Artur Męski

Model Checking for Reaction and Multi-Agent Systems

PhD dissertation

Supervisor:
prof. dr hab. inż. **Wojciech Penczek**
Institute of Computer Science, PAS

Institute of Computer Science
Polish Academy of Sciences

Warsaw, 2019

Acknowledgments

Firstly, I would like to thank Wojciech Penczek for his help, support, infectious enthusiasm over the years it has taken me to complete this thesis. This research would not have been possible without him trusting in me by agreeing to be my supervisor.

I am grateful to Grzegorz Rozenberg for his invaluable comments and for initially suggesting the idea of applying model checking to reaction systems. My thanks also go to Maciej Koutny for inviting me to Newcastle and providing an environment in which many ideas of this thesis were conceived.

This research would not have been possible without Agata Półrola introducing me to Wojciech, explaining to me how to write my first paper, for coaching me for my initial conference presentations and for all the conversations that we had. I was fortunate to have a position at the Institute of Computer Science of Polish Academy of Sciences, where I was able to call Michał Knapik and Maciek Szreter both colleagues and very good friends.

Mariusz Jarocki's belief in me, and in offering me my first academic job, cannot be understated, and my appreciation goes to Mariusz Frydrych for introducing me to many interesting problems and for the subsequent discussions.

My gratitude also goes to Andrew Jones for his support, for reading drafts of this thesis and also for being a true friend.

And last but by no means least, to my family for their support, to my daughter for her patience, and my wife for her love and understanding.

The study was partially funded by the European Union, European Social Fund. Project PO KL *"Information technologies: Research and their interdisciplinary applications"*, agreement UDA-POKL.04.01.01-00-051/10-00.

Contents

Abstract

Reaction systems are a formal model inspired by the functioning of living cells. This model allows for specifying and analysing computational processes in which reactions operate on sets of molecules.

The behaviour of a reaction system is determined by the interactions of its reactions, which are based on the mechanisms of facilitation and inhibition. The formal treatment of reaction systems is qualitative and there is no direct representation of the number of molecules involved in reactions.

In this thesis we introduce formal verification methods for reaction systems. Behaviour of reaction systems depends on interactions of the system with its environment. We show extensions of reaction systems that allow for generating the behaviours of the environment, which are considered realistic or relevant for the verification. We introduce an extension of reaction systems, which allows for direct quantitative modelling of discrete concentration levels. We also propose multi-agent reaction systems that allow for modelling of distributed and multi-agent systems in the reaction systems setting.

To allow for specifying properties of reaction systems and their extensions we introduce temporal logics for reaction systems: rsCTL, which is a branching-time temporal logic, and rsLTL, which is a linear-time temporal logic. Additionally, to enable reasoning about temporal and epistemic properties of multi-agent reaction systems we propose rsCTLK, which is a temporal-epistemic logic. Operators of the introduced logics allow for selecting the desired environment behaviours.

We define parametric reaction systems where reactions may be defined partially, by using parameters in place of ordinary reaction elements. We demonstrate a parameter synthesis approach which under a set of rsLTL specifications calculates a valuation for the parameters which meets the provided specification.

For the defined formalisms we propose symbolic model checking methods based on binary decision diagrams and satisfiability modulo theories. We implement and evaluate these methods experimentally. For the experimental results we introduce a number of scalable reaction systems modelled using the proposed extensions. Finally, we introduce a verification toolkit for reaction systems, which implements the methods proposed in this thesis.

Streszczenie

Systemy reakcji to formalny model zainspirowany funkcjonowaniem komórek żywych i pozwalający na specyfikowanie oraz analizę procesów obliczeniowych, w których reakcje operują na zbiorach molekuł.

Zachowanie systemu reakcji jest określone przez interakcje jego reakcji, które oparte są na mechanizmie umożliwiania i inhibicji. Systemy reakcji w ujęciu formalnym są modelem jakościowym, w którym nie istnieje bezpośrednia reprezentacja liczby molekuł biorących udział w reakcjach.

W niniejszej rozprawie wprowadzamy metody weryfikacji formalnej dla systemów reakcji. Ponieważ zachowanie tych systemów zależy od interakcji ze środowiskiem, w którym funkcjonują, wprowadzamy rozszerzenia systemów reakcji pozwalające na generowanie zachowań środowiska, które uważane są za realistyczne i znaczące dla procesu weryfikacji. Definiujemy również rozszerzenie, które pozwala na bezpośrednie modelowanie zależności ilościowych przez wprowadzenie dyskretnych poziomów koncentracji molekuł. Ponadto, proponujemy formalizm pozwalający na modelowanie systemów rozproszonych oraz wieloagentowych.

W celu umożliwienia specyfikowania własności systemów reakcji oraz ich rozszerzeń, wprowadzamy specjalizowane logiki temporalne: rsCTL do określania własności czasu rozgałęzionego oraz rsLTL do określania własności czasu liniowego. Ponadto, aby pozwolić jednocześnie na określanie własności temporalnych oraz epistemicznych, proponujemy logikę temporalno-epistemiczną rsCTLK. Operatory wprowadzonych logik pozwalają na określanie zachowań środowiska.

W rozprawie definiujemy również pojęcie parametrycznych systemów reakcji, gdzie reakcje mogą być zdefiniowane jedynie częściowo, przez użycie parametrów w miejscu zbiorów występujących w reakcjach. Demonstrujemy również metodę pozwalającą na syntezę wartościowań dla parametrów przy założeniach specyfikowanych za pomocą formuł logiki rsLTL.

Dla zdefiniowanych formalizmów, proponujemy metody symbolicznej weryfikacji modelowej, które oparte są o zastosowanie binarnych diagramów decyzyjnych oraz teorii spełnialności (ang. *satisfiability modulo theories*). Przedstawione metody zostały zaimplementowane i poddane ewaluacji eksperymentalnej. Na potrzeby opracowania wyników eksperymentalnych, wprowadzone zostały skalowalne modele systemów, które zamodelowane zostały z wykorzystaniem zdefiniowanych rozszerzeń systemów reakcji. Implementacje metod zaproponowanych w niniejszej rozprawie stanowią narzędzie do weryfikacji systemów reakcji.

Chapter 1

Introduction

Reaction systems are a formal model for processes inspired by living cells. They capture the basic mechanisms responsible for the dynamic behaviour of living cells. The biochemical interactions of living cells are based on the mechanisms of facilitation and inhibition: reactions and their products, may facilitate or inhibit each other. The simplicity of reaction systems comes from the fact that they model the reactions, states, and dynamic processes using finite sets, and so they directly capture qualitative aspects of systems. There is no direct representation of the number of molecules involved in biochemical reactions, nor the number of molecules present in the current system state. For applications in which correct behaviour of the verified system is crucial, it is essential to provide methods for their analysis. Despite the simplicity of reaction systems, they are capable of representing large numbers of complex processes and understanding the properties of these systems might be challenging.

Model checking is a method that allows for verifying whether a 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 mathematical logic. Model checking is fully automatic and it does not require expert knowledge of verification techniques.

Multi-agent systems are distributed systems composed of interacting autonomous agents, where the agents communicate with each other to solve problems. Their applications can be found in domains such as, e.g., robotics, biology, logistics, and defence systems.

In this doctoral dissertation we introduce formal verification methods based on model checking, which will allow for verification of reaction systems and multi-agent systems modelled in the reaction systems setting.

1.1 Research hypothesis

In this thesis we introduce model checking methods for reaction systems, introduce logics for specifying properties of reaction systems, and propose extensions of reaction systems that facilitate verification. We state the following research hypothesis:

> Formal verification methods based on model checking will allow for verification of reaction systems.

1.2 Related work

In this section we provide an overview of the related work.

Model checking. Model checking was introduced independently by Clarke and Emerson [Clarke and Emerson, 1981, Clarke *et al.*, 1986] and by Queille and Sifakis [Queille and Sifakis, 1982]. The idea of model checking consists of representing a system to be verified in a form of a transition system (model), representing a specification as a temporal logic formula, and checking automatically whether the formula holds in the model. Unfortunately, the practical applicability of the approach is usually limited due to the *state explosion problem*: the analysed state space of the verified system grows significantly for large concurrent systems.

One of the methods used to alleviate the above problem is to apply symbolic model checking [McMillan, 1993], where the state-space is not represented explicitly, and binary decision diagrams (BDD) [Bryant, 1986] are used to store the states of the system and perform operations on them.

Bounded model checking (BMC) [Biere *et al.*, 1999a] is an efficient verification method whose main idea consists in considering a model truncated up to a specific depth. There exist numerous papers which deal with that approach in the context of SAT-based verification for existential temporal properties: a model checking problem on a fraction of the model is translated into the SAT problem, which is then performed using a SAT-solver. Alternatively, the problem can be translated into a *satisfiability modulo theory* (SMT), which is a generalisation of the Boolean satisfiability problem, where some functions and predicate symbols have interpretations from the underlying theory [Armando *et al.*, 2006, Kroening and Strichman, 2016]. Another approach to BMC involves using BDDs [Cabodi *et al.*, 2002, Jones and Lomuscio, 2010, Męski *et al.*, 2011a]. Originally, model checking was introduced for computation tree logic (CTL) [Clarke *et al.*, 1986] and model checking for linear-time temporal logic (LTL) specifications was given in [Lichtenstein and Pnueli, 1985].

There also exist model checking methods for verification of timed systems modelled using time Petri nets and timed automata [Penczek and Półrola, 2006].

An SMT-based BMC method for verification of reachability in bounded time Petri nets was introduced in [Półrola *et al.*, 2014]. BDD and SAT-based BMC methods for verification of distributed time Petri nets and properties expressed in LTL and CTL without the next-step operator were proposed and compared in [Męski *et al.*, 2011b].

Parameter synthesis. In this thesis we also tackle the problem of parameter synthesis (or parametric model checking). In practice, the analysed system or the verified property of the system may be specified partially: parameters might be used in place of the unspecified or unknown elements in the system or in the formula expressing the verified property. The parameter synthesis problem consists in calculating the values for the parameters, given some constraints. Parametric timed automata which extend ordinary timed automata with parameters used in place of unspecified timing constraints were introduced by Alur *et al.* in [Alur *et al.*, 1993b]. In [Hune *et al.*, 2002] the authors present an extension of the UPPAAL [Bengtsson *et al.*, 1995, Behrmann *et al.*, 2004] model checker, which allows for parameter synthesis of linear parameter constraints of parametric timed automata. A parameter synthesis method for a parametric variant of action-restricted CTL (ARCTL) [Pecheur and Raimondi, 2006b] and Kripke structures was proposed in [Knapik *et al.*, 2015]. In [Jones *et al.*, 2012] an approach to synthesis for a temporal-epistemic logic and multi-agent systems, where groups of agents occurring in the formulae are synthesised is considered.

Comprehensive overviews of model checking and other verification methods can be found in [Clarke *et al.*, 1999, Huth and Ryan, 2004, Baier and Katoen, 2008, Grumberg and Veith, 2008].

Reaction systems. The formalism of reaction systems was introduced by Ehrenfeucht and Rozenberg in [Ehrenfeucht and Rozenberg, 2007b].

An important strand of research focuses on applying reaction systems to modelling of different kinds of problems and real-world systems. The paper [Brijder *et al.*, 2011a] presents an introduction to reaction system, proposes a reaction systems model for a binary counter system, and demonstrates how to translate any transition system into a reaction system. In [Corolli *et al.*, 2012] 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 thesis as a benchmark, was described in [Azimi *et al.*, 2014a] together with properties that specify its behaviour. Different reaction system models of the biochemical reactions for self-assembly of vimentin tetramers into intermediate filaments are studied in [Azimi *et al.*, 2015b]. Boolean networks are a formalism used for simulating gene regulatory networks. A translation of Boolean networks into reaction systems was presented in [Barbuti *et al.*, 2018].

There exist extensions of the basic model of reaction systems that allow for

modelling of different classes of systems. The paper [Ehrenfeucht and Rozenberg, 2009] introduces reaction systems with measurements, which allow for expressing quantitative dependencies and are used to introduce time into reaction systems. In reaction systems entities vanish if they are not sustained by reactions. Reaction systems with duration [Brijder *et al.*, 2011c] relax this property by allowing entities to decay in a specified number of steps. In evolving reaction systems [Ehrenfeucht *et al.*, 2017a] the set of the available reactions is allowed to change with each state transition. Quantum and probabilistic reaction systems were proposed in [Hirvensalo, 2012]. In reaction automata [Okubo *et al.*, 2012b], which are a model of computation inspired by reaction systems, only one reaction can take place at a time. In [Okubo *et al.*, 2012b] it is also shown that this model is Turing universal. Reaction automata were studied further in [Okubo *et al.*, 2012a, Okubo, 2014, Okubo and Yokomori, 2015, Okubo and Yokomori, 2016]. A model of exploration systems [Ehrenfeucht and Rozenberg, 2014] extending the framework of reaction systems was defined by introducing zoom structures for representing a depository of knowledge of a discipline of science. The model was also studied in [Ehrenfeucht and Rozenberg, 2015].

There also exist other biologically-inspired modelling formalisms. The most notable ones include chemical reaction networks [Horn and Jackson, 1972], Boolean automata networks [Kauffman, 1969, Shmulevich and Dougherty, 2010], and membrane systems [Paun, 2002]. Simulating reaction systems by membrane systems was discussed in [Alhazov *et al.*, 2016].

A significant part of the research on reaction systems has been focused on their mathematical properties. Minimal reaction systems are defined with reactions using the minimal number of reactants or inhibitors. The paper [Ehrenfeucht *et al.*, 2012b] provided a classification of functions defined by reaction systems with minimal resources. This classification was refined in [Teh and Atanasiu, 2017]. According to the definition of reaction system, the minimal number of resources required to define a reaction is exactly two. The consequences of assuming the minimal number of resources to be three were studied for *almost minimal* reaction systems in [Salomaa, 2013b]. In [Salomaa, 2014] it was demonstrated that everything generated by an arbitrary reaction system can be also generated in three steps by a minimal reaction system, while [Salomaa, 2015] provided a similar result for two steps. Minimal reaction systems with duration were studied in [Salomaa, 2017].

Checking if an entity appears at a given step m in at least one state sequence of a reaction system is the occurrence problem. The problem of convergence is the universal version of that problem and it amounts to checking if the entity appears at the m^{th} step of all the state sequences. These two problems were studied in [Salomaa, 2013a, Salomaa, 2013b, Formenti *et al.*, 2015].

Different classes and properties of reaction systems related to their state sequences were studied in [Salomaa, 2012a, Salomaa, 2012b], while [Ehrenfeucht and Rozenberg, 2007a] tackled the problem of modules and events in reaction systems.

A method for representing reactions of reaction systems using trees was shown

in [Brijder *et al.*, 2012]. Predictors characterise the environment of reaction systems, which leads to production of a certain molecule after a given number of step. Predictors for reaction systems were introduced in [Brijder *et al.*, 2010]. Dynamic causalities in reaction systems together with different notions for representing predictors were studied in [Barbuti *et al.*, 2016a]. These notions were investigated further in [Barbuti *et al.*, 2017] and [Barbuti *et al.*, 2016b], where in the latter it is assumed that the environment provides molecules according to what is expressed by a temporal logic formula.

The complexity of problems related to computing preimages in reaction systems was studied in [Dennunzio *et al.*, 2015c, Dennunzio *et al.*, 2015b]. Several combinatorial problems related to reaction systems are discussed in [Dennunzio *et al.*, 2014, Dennunzio *et al.*, 2015a]. In [Dennunzio *et al.*, 2016] it is shown that the complexity of the reachability problem is PSPACE-complete in reaction systems for several classes of resource-bounded reaction systems and the complexity is lower when a certain restriction is applied.

The dynamics of reaction systems may result in, e.g., existence of fixed points, attractors, and cycles. Problems related to detecting such properties were investigated in [Formenti *et al.*, 2014a, Formenti *et al.*, 2014b].

Verification of reaction systems. In [Azimi *et al.*, 2015a] the authors investigate the property of mass conservation using dependency graphs, and introduce a simulator for reaction systems. A reaction systems simulator focusing on efficiency and implemented on graphics processing units (GPU) is proposed in [Nobile *et al.*, 2017]. A web-based version of the simulator introduced in [Azimi *et al.*, 2015a] is presented in [Ivanov *et al.*, 2018]. The relationship between reaction systems and synchronous digital circuits is studied in [Shang *et al.*, 2019], leading to a simulation method for reaction systems based on translating them into hardware circuits that are emulated using field-programming gate arrays (FPGA). In [Azimi *et al.*, 2016] the authors define several biologically inspired properties for reaction systems and study their verification problems together with their computational complexity. The defined problems include mass conservation, invariants, steady states, stationary processes, elementary fluxes, and periodicity. Steady states of reaction systems are introduced in [Azimi *et al.*, 2016]. A polynomial time algorithm for finding all the steady states of a constrained reaction system is given in [Azimi, 2017]. In [Azimi *et al.*, 2017] the authors introduce a reaction systems framework for dealing with properties important in biology, such as multiple steady states, bistability, limit cycle oscillation, and period-doubling bifurcation.

Multi-agent systems. In this dissertation we also tackle aspects of verification of multi-agent systems. Interpreted systems [Fagin *et al.*, 2003] are the most commonly used formalism for modelling multi-agent systems for model checking. Temporal-epistemic properties of multi-agent systems can be expressed using CTL*K [van der Meyden and Wong, 2003], which is a logic combining epistemic operators with

temporal operators of CTL* [Emerson and Halpern, 1986]. A symbolic model checking method for interpreted systems and CTLK specifications was proposed in [Raimondi and Lomuscio, 2005] and a bounded model checking approach for multi-agent systems and temporal-epistemic properties was put forward in [Penczek and Lomuscio, 2003]. A comparison of methods for bounded model checking and LTLK specifications for interpreted systems, based on BDDs and SAT was given in [Męski *et al.*, 2014b]. There also exist other modelling formalisms for multi-agent systems, e.g., modular interpreted systems [Jamroga and Ågotnes, 2007, Jamroga *et al.*, 2013].

1.3 Contributions

The high-level outcomes of the research work presented in this thesis are:

- model checking methods for reaction systems,
- a parametric model checking approach for reaction synthesis,
- a reaction systems model checking toolkit implementing the introduced verification methods.

Below we provide a detailed overview of the contributions of the individual chapters.

Chapter 3. We introduce rsCTL, computation tree logic for reaction systems, which is a logic for specifying properties of reaction systems, as well as a method for verifying these properties. The processes of reaction systems are guided by the context sequences (which model interactions 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 symbolic model checking. We also provide complexity results for the problem of model checking reaction systems and prove that the complexity of rsCTL model checking is PSPACE-complete.

Chapter 4. We extend the basic reaction systems to allow for modelling of distributed and multi-agent systems, and provide support for different synchronisation schemes. The formalism allows for modelling of systems employing both synchronous and asynchronous execution semantics. We demonstrate how the formalism of multi-agent reaction systems can be applied to modelling of multi-agent systems.

The model brings together advantages of reaction systems, membrane systems [Alhazov, 2006, Kleijn and Koutny, 2011] and tissue systems [Paun, 2002, Păun and Rozenberg, 2002, Paun *et al.*, 2010] to model networks of cells. This is achieved

by extending the basic reaction systems model with the idea of compartmentalisation derived from membrane and tissue systems which allows for modelling of distributed systems.

Further, we introduce *context automata*, which represent the influence of the bigger system in which the reaction system models are situated. We also introduce an extended notion of context automata allowing conditional generation of contexts (based on the current state of a reaction system).

To enable specifying temporal-epistemic properties of multi-agent reaction systems we introduce a new logic for reaction systems: rsCTLK, which combines rsCTL (Chapter 3) with CTLK [Penczek and Lomuscio, 2003]. We introduce model checking for multi-agent reaction systems and properties expressed in rsCTLK. The proposed method is implemented using binary decision diagrams for symbolic model checking and the method is evaluated experimentally. Additionally, we prove that model checking for rsCTLK is PSPACE-complete.

Chapter 5. We introduce a formalism supporting quantitative modelling by considering reaction systems with discrete concentrations of entities.

Although reaction systems with discrete concentrations are semantically equivalent to the original qualitative reaction systems, they provide much more succinct representations in terms of the number of entities being used, and allow for more efficient verification. Our experimental results show a significant improvement in the execution times in favour of reaction systems with discrete concentrations.

We define a variant of linear-time temporal logic (rsLTL) interpreted over models of reaction systems with discrete concentrations. We provide an encoding into SMT, together with a bounded model checking method, and present experimental results showing the efficiency of verification for reaction systems with discrete concentrations.

Chapter 6. In practical applications, a reaction system may have only partially specified reactions, where reactants, inhibitors, or products might be initially unknown. In such situations, we propose to use parameters in place of the unspecified reaction parts. We develop a reaction mining approach where the missing inputs are computed automatically. To develop such an approach, we introduce reaction systems with parameters. The main result is a methodology that calculates the valuations of the parameters in such a way that the resulting reaction system satisfies a given rsLTL formula when operating in a given external environment. Intuitively, such a formula might correspond to a number of observations (runs) of the behaviour of a partially specified system.

We provide a suitable encoding of parametric reaction systems in SMT, and propose a procedure based on bounded model checking for solving the synthesis problem. We also provide experimental results demonstrating the scalability of this method and its potential applications.

Chapter 7. We present a toolkit for verification of reaction systems which combines all the implementations prepared for the preceding chapters. The toolkit consists of two separate modules, which are responsible for supporting BDD-based and SMT-based verification methods. An overview of the modules and their architecture is provided. We also introduce RSSL, which is an input language for specifying reaction systems and their properties.

1.3.1 Publications

I was the lead author of the following papers:

- *"Model checking temporal properties of reaction systems"*, on which Chapter 3 is based. The co-authors of this paper were: Wojciech Penczek and Grzegorz Rozenberg. It was published in Information Sciences, 2015 ([Męski *et al.*, 2015]). This paper is an extended version of [Męski *et al.*, 2014a] published as Technical Report 1028 of Institute of Computer Science, Polish Academy of Sciences, April 2014.
- *"Model Checking for Temporal-Epistemic Properties of Distributed Reaction Systems"*, on which Chapter 4 is based. The co-authors of this paper were: Maciej Koutny and Wojciech Penczek. It was published as a technical report of School of Computing, University of Newcastle upon Tyne ([Męski *et al.*, 2019]).
- *"Verification of Linear-Time Temporal Properties for Reaction Systems with Discrete Concentrations"*, on which Chapter 5 is based. The co-authors of this paper were: Maciej Koutny and Wojciech Penczek. It was published in Fundamenta Informaticae, 2017 ([Męski *et al.*, 2017]). This paper is an extended version of the paper *"Towards Quantitative Verification of Reaction Systems"*, published in the proceedings of Unconventional Computation and Natural Computation – 15th International Conference, UCNC 2016, Manchester, UK, July 11-15, 2016 ([Męski *et al.*, 2016]).
- *"Reaction Mining for Reaction Systems"*, on which Chapter 6 is based. The co-authors of this paper were: Maciej Koutny and Wojciech Penczek. It was published in the proceedings of Unconventional Computation and Natural Computation – 17th International Conference, UCNC 2018, Fontainebleau, France, June 25-29, 2018 ([Męski *et al.*, 2018]).

Citations

The paper [Męski *et al.*, 2015], on which Chapter 3 is based, was cited in papers on the following topics:

1. Dependency graphs for investigation of the mass conservation property, and simulation of reaction systems [Azimi *et al.*, 2015a].

2. Analysis of dynamic causalities using predictors involving temporal logic formulae [Barbuti *et al.*, 2016b].
3. Verification problems for reaction systems and their computational complexity [Azimi *et al.*, 2016].
4. An overview of research on reaction systems [Ehrenfeucht *et al.*, 2017b].
5. Simulation of reaction systems using graphics processing units [Nobile *et al.*, 2017].
6. Equivalence notions for reaction systems [Kleijn *et al.*, 2018].
7. Web-based reaction systems simulator [Ivanov *et al.*, 2018].
8. Reaction systems which communicate with their environment [Bottoni *et al.*, 2019].
9. Translation of reaction systems into a process algebra [Brodo *et al.*, 2019].
10. Relationship between synchronous digital circuits and reaction systems [Shang *et al.*, 2019].

1.3.2 Collaborations

Grzegorz Rozenberg suggested the idea of introducing model checking for reaction systems and for the verification of context-independent sequences of reaction systems. This resulted in defining the model checking approach presented in Chapter 3.

Maciej Koutny helped with the formalisation of the translation of context restricted reaction systems with discrete concentrations into context restricted reaction systems. The translation was presented in Section 5.1. He also participated in the initial discussions about the definition of multi-agent reaction systems introduced in Chapter 4. The generalisation of context automata which follows the idea of state-aware context controllers of [Kleijn *et al.*, 2018] was suggested by Maciej Koutny. This has resulted in Definition 4.2.6 which defines extended context automata.

1.4 Structure of this thesis

Chapter 1 provides an introduction to the subject of this thesis, describes its most important contributions, and gives an overview of the literature.

Chapter 2 introduces reaction systems and basic notions used in the remainder of the thesis.

Chapter 3 defines rsCTL, a branching time temporal logic for reaction systems. The logic is interpreted over the models for context restricted reaction systems which generalise standard reaction systems by controlling context sequences. A translation from the context restricted reaction systems into Boolean formulae is defined. The Boolean formulae encoding is then used to implement

symbolic model checking for rsCTL. The proposed method is implemented into a model checking tool using binary decision diagrams. The chapter concludes with an experimental evaluation which validates various reaction systems.

Chapter 4 focuses on multi-agent reaction systems, an extension of reaction systems for modelling of distributed and multi-agent systems. To allow for expressing temporal-epistemic properties of multi-agent reaction systems we introduce rsCTLK. The model checking problem for rsCTLK is defined together with a symbolic model checking method based on binary decision diagrams. Finally, for an implementation of the proposed verification method experimental results are provided.

Chapter 5 introduces reaction systems with discrete concentrations that allow for direct quantitative modelling. A linear-time temporal logic for reaction systems, rsLTL, is introduced. We define the bounded model checking problem for rsLTL as well as a translation into SMT. The presented SMT-based bounded model checking method is evaluated by comparing quantitative reaction systems modelling approach for model checking with the direct modelling approach of reaction systems with discrete concentrations.

Chapter 6 introduces parametric reaction systems in which reactions can be defined partially. To allow for specifying restrictions on parameters and relationships between them, a language of parameter constrains is introduced. We propose an approach to parameter synthesis, given a set of observations expressed in rsLTL. Using the example of the mutual exclusion problem, we demonstrate how to synthesise a malicious reaction whose presence invalidates the integrity of the system. We provide an experimental evaluation with an analysis of the efficiency of the synthesis method and the SMT encoding, which is compared with the non-parametric encoding (Chapter 5).

Chapter 7 presents a reaction systems verification toolkit which implements the methods developed in the dissertation. It gives an overview of the architecture and the relationships between the modules of the system. A language for specifying reaction system models and their properties is also introduced.

Chapter 8 provides a summary, concluding remarks, and suggests possible future research directions.

Chapter 2

Preliminaries

In this chapter we recall some basic notions for reaction systems that are used in this dissertation.

2.1 Reaction systems

First of all, we recall the notion of a reaction. This section is based on [Brijder *et al.*, 2011a].

In this thesis, by $\mathbb{Z}$ we denote the set of integers and by $\mathbb{N}$ the set of natural numbers including 0.

Definition 2.1.1. A *reaction* is a triple $b = (R, I, P)$ such that R, I, P are finite nonempty sets with $R \cap I = \varnothing$.

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 R_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 ($R \neq \varnothing$), each reaction may be inhibited ($I \neq \varnothing$), and if a reaction takes place then this creates a "material" effect – something is produced ($P \neq \varnothing$).

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 2.1.2. 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 $I_b \cap T = \varnothing$. The *result* of b on T, denoted by $res_b(T)$, is defined by: $res_b(T) = P_b$ if $en_b(T)$, and $res_b(T) = \varnothing$ otherwise.

2. For $B \subseteq rac(Z)$, the result of B on T, denoted by $res_B(T)$, is defined by $res_B(T) = \bigcup\{res_b(T) \mid b \in B\}$.

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 in 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 $R_b \cap I_b = \varnothing$ – in this way we do not consider "trivial reactions", i.e., reaction that are never enabled.

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

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

Definition 2.1.3. 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 2.1.4. Consider the set $S = \{1, 2, 3, 4\}$ and the following set A of reactions from $rac(S)$:

- $a_1 = (\{1, 4\}, \{2\}, \{1, 2\})$,
- $a_2 = (\{2\}, \{4\}, \{1, 3, 4\})$,
- $a_3 = (\{1, 3\}, \{2\}, \{1, 2\})$, 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., [Ehrenfeucht *et al.*, 2012a]).

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., [Ehrenfeucht *et al.*, 2012a]) 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 2.1.5. Let $\mathcal{R} = (S, A)$ be a RS and let $n \geq 1$.

1. An (n-step) *interactive process* of $\mathcal{R}$ is a pair $\pi = (\gamma, \delta)$ of finite sequences of finite sets such that:
   (a) $\gamma = (C_0, C_1, \cdots, C_n)$ and $\delta = (D_0, D_1, \cdots, D_n)$,
   (b) $C_0, C_1, \cdots, C_n \subseteq S$,
   (c) $D_0, D_1, \cdots, D_n \subseteq S$, with $D_0 = \varnothing$, and
   (d) $D_i = res_A(D_{i-1} \cup C_{i-1})$ for all $i \in \{1, \cdots, n\}$.
2. The *state sequence* of π is the sequence $\tau = (W_0, W_1, \cdots, W_n)$ 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 $(C_1, C_2, \cdots, C_n)$ 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 provided 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).

2.2 Examples of reaction systems

In this section we provide examples of reaction systems found in literature.

2.2.1 Basic examples

Example 2.2.1. Now we provide a reaction system implementation of an n-bit cyclic binary counter (BC) given in [Brijder *et al.*, 2011b, Section 4.1]. 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 defined as:

$$S_n = \{p_0, \ldots, p_{n-1}, inc, dec\}.$$

The entities $\{p_0, \ldots, p_{n-1}\}$ allow to provide a set representation of the 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-ten) is represented by the set $\{p_0, p_1, p_3\}$. Thus, for each $W \subseteq S_n$, the value represented by W equals

$$\sum_{p_i \in (W \setminus \{inc, dec\})} 2^i.$$

The entities *inc* and *dec* are used to represent the instructions to increment by one, and to decrement by one the value of the current state of the counter. If one attempts to increment and decrement 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:

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

Finally the desired RS is defined as $\mathcal{R}^n_{\text{BC}} = (S_n, B_n)$. □

Example 2.2.2. We consider here an implementation by a RS of a small generic regulatory system[1] GRS discussed in [Ehrenfeucht *et al.*, 2012a, 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. For example, h could represent a high level of radiation that may inhibit a reaction.

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 GRS does not specify such inhibitions, this inhibition is expressed by the dummy inhibitor h.

[1]The definition of these biological terms is out of scope of this thesis; we refer the motivated reader to [Sneppen, 2014].

– $(\{x\}, \{Y, Z\}, \{\widehat{x}\})$ 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 or Z are 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 GRS) formalised by the dummy h.

Consequently, the reaction system for the regulatory system is modelled by $\mathcal{R}_{\text{GRS}} = (S, A)$, where: $A = A_x \cup A_y \cup A_z \cup A_Q$. □

2.2.2 Heat shock response model

Example 2.2.3. A qualitative model of the eukaryotic heat shock response[2] (HSR) was introduced in [Azimi *et al.*, 2014b]. 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.

Next, we define the reaction system $\mathcal{R}_{\text{HSR}} = (S, A)$ modelling HSR. The background set is defined as follows:

$$S = \{hsp, hsf, hsf_2, hsf_3, hse, mfp, prot, hsf_3{:}hse, hsp{:}hsf, hsp{:}mfp, stress, nostress, h\}.$$

The entities used in the model are summarised in Table 2.1. The entities *stress* and *nostress* indicate, respectively, presence and absence of an environmental stressor triggering heat shock response. For instance, *stress* might correspond to temperature greater than 42 °C, and *nostress* to temperature lower than 37 °C.

The set A of the reactions is composed of the following elements:

[2] For an in-depth study of the problem we refer the interested reader to [Voellmy and Boellmann, 2007].

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 heat shock$^\diamondsuit$
$nostress$	absence of heat shock$^\diamondsuit$
h	dummy inhibitor

Table 2.1: Summary of the entities used in the heat shock response model. The entities used in the context sets are marked with $\diamondsuit$.

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

– $(\{hsp{:}mfp\}, \{h\}, \{hsp, prot\})$.

□

2.3 Summary

In this section we recalled some basic notions related to reaction systems. We have also given some examples of modelling with reaction systems found in literature. In the following chapters we use these examples to demonstrate the verification methods introduced in this thesis.

Chapter 3

Model checking for rsCTL

In this chapter we introduce a model checking method for a branching-time logic and reaction systems.

3.1 Controlling context sequences

For the purpose of model checking we need to decide, which context sequences should be considered in the verification. This requires a method in which one would be able to select these context sequences, which are relevant for the system under analysis. Considering all the possible context sequences generated by the entire background set could result in unnecessarily large state spaces, which would have a negative impact on the efficiency of the verification algorithms.

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 3.1.1. A *simple context restricted reaction system*, *SCRRS* for short, is a triple SCR-$\mathcal{R} = (S, A, \mathcal{E})$ 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 3.1.2. Let SCR-$\mathcal{R} = (S, A, \mathcal{E})$ be an SCRRS. An *initialised context restricted reaction system*, *ICRRS* for short, is a pair ICR-$\mathcal{R} = (\text{SCR-}\mathcal{R}, S_0)$, where $S_0 \subseteq 2^S$ is the set of *initial states* such that $S_0 \neq \varnothing$.

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

Definition 3.1.3. Let SCR-$\mathcal{R} = (S, A, \mathcal{E})$ be a SCRRS and let $n \geq 0$ be an integer. An (n-step) *interactive process* in SCR-$\mathcal{R}$ is a pair $\pi = (\gamma, \delta)$ of finite sequences of finite sets such that:

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

The above definition differs from Definition 2.1.5 by restricting in (2) the proper context sets to be subsets of $\mathcal{E}$.

To define an interactive process in ICRRS the definition for SCRRS is augmented with a restriction on the initial context set.

Definition 3.1.4. Let ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ be a ICRRS and let $n \geq 0$ be an integer. An (n-step) *interactive process* in ICR-$\mathcal{R}$ is a pair $\pi = (\gamma, \delta)$ of finite sequences of finite sets such that:

1. $\gamma = (C_0, C_1, \cdots, C_n)$ and $\delta = (D_0, D_1, \cdots, D_n)$,
2. $C_0 \in S_0$, $C_1, \cdots, C_n \subseteq \mathcal{E}$,
3. $D_0, D_1, \cdots, D_n \subseteq S$, $D_0 = \varnothing$, and
4. $D_i = res_A(D_{i-1} \cup C_{i-1})$ for all $i \in \{1, \cdots, n\}$.

Next, we give a simple example of an ICRRS.

Example 3.1.5. Consider the ICRRS ICR-$\mathcal{R}_1 = \big((S, A, \{4\}), \big\{\{1, 4\}\big\}\big)$, where (S, A) is the RS $\mathcal{R}_1$ from Example 2.1.4.

Let $\gamma = (\{1, 4\}, \varnothing, \{4\}, \{4\})$ and $\delta = (\varnothing, \{1, 2\}, \{1, 3, 4\}, \{1, 2\})$. Then, $\pi = (\gamma, \delta)$ is a 3-step interactive process of $\mathcal{R}_1$. It is also an interactive process of ICR-$\mathcal{R}_1$, 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 SCRRS $(S, A, \{1, 3\})$, because the context set $\{4\}$ is not a subset of $\{1, 3\}$. □

3.2 Examples

Example 3.2.1. Here we define an ICRRS for the reaction system defined in Example 2.2.2. 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 is defined as

$$\text{ICR-}\mathcal{R}_{\text{GRS}} = \big((S, A, \{U\}), \big\{\{x, y\}\big\}\big),$$

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

Example 3.2.2. Now we provide an ICRRS that uses the RS implementing an n-bit cyclic binary counter and defined in Example 2.2.1 as $\mathcal{R}^n_{\text{BC}} = (S_n, B_n)$. The desired ICRRS is then defined as ICR-$\mathcal{R}^n_{\text{BC}} = ((S_n, B_n, \mathcal{E}), \{\varnothing\})$ 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 ICR-$\mathcal{R}^n_{\text{BC}}$) 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. □

3.3 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.

3.3.1 Syntax and semantics

Let ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ be an ICRRS, and $\mathcal{PV}$ be a nonempty set of propositional variables. Without loss of generality, we assume that $\mathcal{PV} = S$, i.e., we will be using the entities of S as propositional variables. The language of *computation tree logic for reaction systems*, rsCTL for short, is defined by the following grammar:

$$\phi := \wp \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathsf{E}_\Psi \mathsf{X}\phi \mid \mathsf{E}_\Psi \mathsf{G}\phi \mid \mathsf{E}_\Psi[\phi \mathsf{U} \phi],$$

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

The operators of rsCTL are composed of the path quantifier E_Ψ and temporal operators (X, G, U). The path quantifier E_Ψ 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 $\mathsf{X}\phi$ operator means 'in the next state ϕ holds', $\mathsf{G}\phi$ means 'in each state of the path (globally) ϕ holds'. The $\phi \mathsf{U} \psi$ 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 3.4.

Definition 3.3.1. 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 $\phi \in \{\neg\phi_1, \mathsf{E}_\Psi \mathsf{X}\phi_1, \mathsf{E}_\Psi \mathsf{G}\phi_1\}$, then $\mathsf{d}(\phi) = \mathsf{d}(\phi_1) + 1$,
- if $\phi \in \{\phi_1 \vee \phi_2, \phi_1 \wedge \phi_2, \mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]\}$, then $\mathsf{d}(\phi) = max(\{\mathsf{d}(\phi_1), \mathsf{d}(\phi_2)\}) + 1$.

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

Definition 3.3.2. 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 $\phi = \neg\phi_1$, then $\mathsf{c}(\phi) = \mathsf{c}(\phi_1)$,
- if $\phi \in \{\neg\phi_1, \mathsf{E}_\Psi \mathsf{X}\phi_1, \mathsf{E}_\Psi \mathsf{G}\phi_1\}$, then $\mathsf{c}(\phi) = max(\{|\Psi|, \mathsf{c}(\phi_1)\})$,
- if $\phi \in \{\phi_1 \vee \phi_2, \phi_1 \wedge \phi_2, \mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]\}$, then $\mathsf{c}(\phi) = max(\{|\Psi|, \mathsf{c}(\phi_1), \mathsf{c}(\phi_2)\})$.

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

Definition 3.3.3. Let ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ be an ICRRS. Then, the model for ICR-$\mathcal{R}$ is defined as $\mathcal{M}(\text{ICR-}\mathcal{R}) = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ where:

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

The set of the initial states defined in (2) consists of the results of applying the reactions to the initial context sequences.

For simplicity, in the sequel of this chapter we fix ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ and its model $\mathcal{M}_{\text{ICR-}\mathcal{R}} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$. Additionally, each element $(w, \alpha, w') \in \longrightarrow$ is denoted by $w \xrightarrow{\alpha} w'$.

The next lemma follows immediately from Definition 3.3.3. It states that the transition relation is serial, i.e., every state of the model has a successor.

Lemma 3.3.4. *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 the models for CTL [Clarke *et al.*, 1999], the paths are defined as sequences of states. However, in the models for rsCTL, the paths contain additional elements, which represent context sets. Thus, they are defined as sequences of states interleaved with *actions*, which are the subsets of the set $\mathcal{E}$.

Definition 3.3.5. Let $\Psi \subseteq 2^{\mathcal{E}}$ and $\Psi \neq \varnothing$. A *path* over Ψ is an infinite sequence $\sigma = (w_0, \alpha_0, w_1, \alpha_1, \dots)$ of states and actions such that $w_i \xrightarrow{\alpha_i} w_{i+1}$ and $\alpha_i \in \Psi$ for each $i \geq 0$.

The set of all the paths over Ψ is denoted by Π_Ψ^{inf}. 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

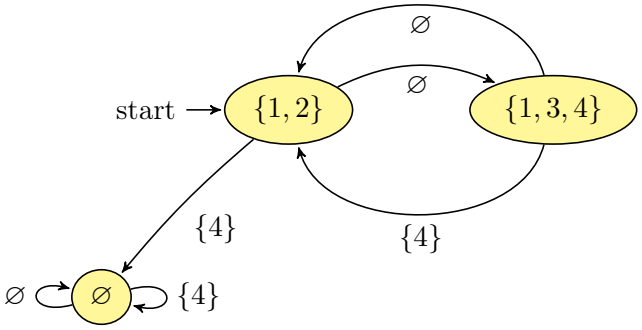

Figure 3.1: The reachable part of the model for ICR-$\mathcal{R}_1$ from Example 3.1.5

by $\sigma_a(i)$. By $\Pi_\Psi^{inf}(w)$ we denote the set of all the paths over Ψ that start in $w \in \mathbb{W}$, that is, $\Pi_\Psi^{inf}(w) = \{\sigma \in \Pi_\Psi^{inf} \mid \sigma_s(0) = w\}$.

Let $w, w' \in \mathbb{W}$ and $\Psi \subseteq 2^{\mathcal{E}}$. We say that w' is a Ψ-successor of w (denoted by $w \longrightarrow_\Psi w'$) 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 3.3.6. Let ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ be an ICRRS and let $\mathcal{M}_{\text{ICR-}\mathcal{R}} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ be the model for ICR-$\mathcal{R}$. We say that a state $w \in \mathbb{W}$ is *reachable* over $\Psi \subseteq 2^{\mathcal{E}}$ in $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ if there exists $w' \in \mathbb{W}_0$ and a path $\sigma \in \Pi_\Psi^{inf}(w')$ such that $\sigma_s(i) = w$ for some $i \geq 0$.

Example 3.3.7. Consider the ICRRS ICR-$\mathcal{R}_1$ from Example 3.1.5. Then, the reachable part of the model $\mathcal{M}_{\text{ICR-}\mathcal{R}_1}$ is shown in Figure 3.1. The states of the model are depicted as the ellipsis. □

Now we are ready to define the semantics of rsCTL.

Definition 3.3.8. Let $\mathcal{M}_{\text{ICR-}\mathcal{R}} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ be a model and $w \in \mathbb{W}$ be a state of $\mathcal{M}_{\text{ICR-}\mathcal{R}}$. The fact that ϕ holds in the state w of the model $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ is denoted by $\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \phi$, where the relation $\models$ is defined recursively as follows:

$$
\begin{array}{lll}
\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \wp & \text{iff} & \wp \in L(w) \text{ for } \wp \in \mathcal{PV}, \\
\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \neg\phi & \text{iff} & \mathcal{M}_{\text{ICR-}\mathcal{R}}, w \not\models \phi, \\
\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \phi \vee \psi & \text{iff} & \mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \phi \text{ or } \mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \psi, \\
\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \phi \wedge \psi & \text{iff} & \mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \phi \text{ and } \mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \psi, \\
\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \mathsf{E}_\Psi \mathsf{X}\phi & \text{iff} & (\exists \sigma \in \Pi_\Psi^{inf}(w))\ \mathcal{M}_{\text{ICR-}\mathcal{R}}, \sigma_s(1) \models \phi, \\
\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \mathsf{E}_\Psi \mathsf{G}\phi & \text{iff} & (\exists \sigma \in \Pi_\Psi^{inf}(w))(\forall i \geq 0)\big(\mathcal{M}_{\text{ICR-}\mathcal{R}}, \sigma_s(i) \models \phi\big), \\
\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \mathsf{E}_\Psi[\phi \mathsf{U} \psi] & \text{iff} & (\exists \sigma \in \Pi_\Psi^{inf}(w))(\exists i \geq 0)\big(\mathcal{M}_{\text{ICR-}\mathcal{R}}, \sigma_s(i) \models \psi \\
 & & \quad \text{and } (\forall 0 \leq j < i)\ \mathcal{M}_{\text{ICR-}\mathcal{R}}, \sigma_s(j) \models \phi\big).
\end{array}
$$

We define now derived operators, which also introduce the universal path quantifier A_Ψ meaning 'for all the paths over Ψ':

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

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

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

The language of our logic resembles the language of action-restricted CTL (ARCTL) of [Pecheur and Raimondi, 2006a]. 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 $\Psi = \{\varnothing\}$. 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 3.7) 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. In Chapter 4 we formally define a logic extending rsCTL, in which path quantifiers use propositional formulae built over the entities.

3.3.2 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 where all the context sets are empty.

Lemma 3.3.9. *Let ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ be an ICRRS and $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ be the model for ICR-$\mathcal{R}$. There exists a path σ such that $\sigma_a(i) \subseteq \sigma_s(i)$ for each $i \geq 0$, if and only if, there exists a path σ' such that $\sigma'_s(i) = \sigma_s(i)$ and $\sigma'_a(i) = \varnothing$ for each $i \geq 0$.*

Proof. First, we assume that there exists a path σ such that $\sigma_a(i) \subseteq \sigma_s(i)$ for each $i \geq 0$. We construct the path σ' with the same states as in σ, i.e., $\sigma'_s(i) = \sigma_s(i)$ for each $i \geq 0$. Next, we show the states of σ' can be reached by following actions that are empty context sets only, i.e., $\sigma'_s(i) \xrightarrow{\varnothing} \sigma'_s(i+1)$ for each $i \geq 0$. From the definition of the transition relation and the fact that $\sigma_a(i) \subseteq \sigma_s(i)$, it follows that $res_A(\sigma_s(i) \cup \sigma_a(i)) = res_A(\sigma_s(i) \cup \varnothing) = res_A(\sigma_s(i))$. 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 $\sigma'_s(i) \xrightarrow{\varnothing} \sigma'_s(i+1)$, i.e., $\sigma'_a(i) = \varnothing$ for each $i \geq 0$. 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 3.3.10. For the ICRRS ICR-$\mathcal{R}_{\text{GRS}}$ defined in Example 3.2.1 we can describe the following properties interpreted in the model $\mathcal{M}_{\text{ICR-}\mathcal{R}_{\text{GRS}}}$, i.e., they must hold in the initial states of the model:

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

$$\mathsf{EF}Q.$$

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):

$$\mathsf{AG}\left(Q \Rightarrow (\mathsf{AX}\neg\widehat{y})\right).$$

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:

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

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

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

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

$$\mathsf{A}_{\Psi}\mathsf{G}(\neg Q), \text{ where } \Psi = \{\alpha \subseteq \mathcal{E} \mid U \notin \alpha\} = \{\varnothing\}.$$

   The set Ψ contains only $\varnothing$, since this is the only possible context set when we disallow U.

6. There exists a context-independent sequence (see Lemma 3.3.9) over which the state where X and Y are present is reachable:

$$\mathsf{E}_{\{\varnothing\}}\mathsf{F}(X \wedge Y).$$

□

Example 3.3.11. For the ICRRS ICR-$\mathcal{R}_{\text{BC}}$ from Example 3.2.2 we can describe the following properties interpreted according to their validity in $\mathcal{M}_{\text{ICR-}\mathcal{R}_{\text{BC}}}$:

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:

$$\mathsf{AG}\left((\neg b_0 \wedge \cdots \wedge \neg b_n) \Rightarrow \mathsf{E}_{\{\{inc\},\{dec\}\}}\mathsf{F}(b_0 \wedge \cdots \wedge b_n)\right).$$

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:

$$\mathsf{AG}\left((b_0 \wedge \cdots \wedge b_n) \Rightarrow \mathsf{A}_{\{\{inc\}\}}\mathsf{X}(\neg b_0 \wedge \cdots \wedge \neg b_n)\right).$$

□

3.4 Verification of rsCTL properties

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 3.6.

To be able to verify rsCTL properties of a given ICRRS ICR-$\mathcal{R}$, we need the set of the reachable states of the model $\mathcal{M}_{\text{ICR-}\mathcal{R}}$. 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 $\tau : 2^W \longrightarrow 2^W$ 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 inductively 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 [Tarski, 1955] that if τ is monotone and W is a finite set, then there exist $m, n \leq |W|$ such that $\tau^m(\varnothing)$ 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 $\mathcal{M}_{\text{ICR-}\mathcal{R}} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ be a model. From Definition 3.3.3 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 $W \subseteq \mathbb{W}$:

$$\mathsf{post}_\Psi(W) \stackrel{def}{=} \{w' \in \mathbb{W} \mid (\exists w \in W)\ w \longrightarrow_\Psi w')\} \text{ where } \Psi \subseteq 2^{\mathcal{E}}.$$

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

$$\mathit{Reach}(\text{ICR-}\mathcal{R}) = \mu X.\big(\mathbb{W}_0 \cup X \cup \mathsf{post}_{2^\mathcal{E}}(X)\big).$$

Algorithm 1: The algorithm for computing the set $\mathit{Reach}(\text{ICR-}\mathcal{R})$

1: $X := \mathbb{W}_0$
2: $X_p := \varnothing$
3: **while** $X \neq X_p$ **do**
4: $\quad X_p := X$
5: $\quad X := X \cup \mathsf{post}_{2^\mathcal{E}}(X)$
6: **end while**
7: **return** X

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

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

$$\mathsf{pre}^{\exists}_\Psi(W) = \{w \in \mathit{Reach}(\text{ICR-}\mathcal{R}) \mid (\exists w' \in W)\ w \longrightarrow_\Psi w')\}.$$

Let ϕ_1, ϕ_2 be some rsCTL formulae. We define the following sets:

$$[\![\neg\phi_1]\!] \stackrel{def}{=} \mathit{Reach}(\text{ICR-}\mathcal{R}) \setminus [\![\phi_1]\!],$$
$$[\![\phi_1 \wedge \phi_2]\!] \stackrel{def}{=} [\![\phi_1]\!] \cap [\![\phi_2]\!],$$
$$[\![\phi_1 \vee \phi_2]\!] \stackrel{def}{=} [\![\phi_1]\!] \cup [\![\phi_2]\!],$$
$$[\![\mathsf{E}_\Psi \mathsf{X}\phi_1]\!] \stackrel{def}{=} \mathsf{pre}^{\exists}_\Psi([\![\phi_1]\!]).$$

The remaining operators are defined as the following fixed points:

$$[\![\mathsf{E}_\Psi \mathsf{G}\phi_1]\!] \stackrel{def}{=} \nu X.\big([\![\phi_1]\!] \cap \mathsf{pre}^{\exists}_\Psi(X)\big),$$
$$[\![\mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]]\!] \stackrel{def}{=} \mu X.\big([\![\phi_2]\!] \cup ([\![\phi_1]\!] \cap \mathsf{pre}^{\exists}_\Psi(X))\big).$$

In the case of $[\![\mathsf{E}_\Psi \mathsf{G}\phi_1]\!]$ the greatest fixed point computation is involved, which in each iteration removes states that do not have a Ψ-predecessor in which ϕ_1 is satisfied. In the case of $[\![\mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]]\!]$ the least fixed point computation is involved, such that in each iteration the Ψ-predecessors, in which ϕ_1 is satisfied, are added to the set of states in which ϕ_2 is satisfied. See Algorithm 2 and 3 for the pseudo-code

Algorithm 2: Procedure $\mathsf{checkEG}(\Psi, \phi_1)$

1: $X := \mathit{Reach}(\text{ICR-}\mathcal{R})$, $X_p := \varnothing$
2: $Y_\phi := \mathsf{check}_{rs\text{CTL}}(\phi)$
3: **while** $X \neq X_p$ **do**
4: $\quad X_p := X$
5: $\quad X := \left(Y_{\phi_1} \cap \mathsf{pre}^{\exists}_{\Psi}(X)\right)$
6: **end while**
7: **return** X

Algorithm 3: Procedure $\mathsf{checkEU}(\Psi, \phi_1, \phi_2)$

1: $X := \varnothing$, $X_p := \mathit{Reach}(\text{ICR-}\mathcal{R}))$
2: $Y_{\phi_1} := \mathsf{check}_{rs\text{CTL}}(\phi_1)$, $Y_{\phi_2} := \mathsf{check}_{rs\text{CTL}}(\phi_2)$
3: **while** $X \neq X_p$ **do**
4: $\quad X_p := X$
5: $\quad X := Y_{\phi_2} \cup (Y_{\phi_1} \cap \mathsf{pre}^{\exists}_{\Psi}(X))$
6: **end while**
7: **return** X

of the procedures implementing the described fixed point computations. The overall procedure $\mathsf{check}_{rs\text{CTL}(\phi)}$ for computing the set of states in which a given rsCTL formula ϕ holds is outlined in Algorithm 4.

3.4.1 Complexity analysis

We consider now the complexity of $\mathsf{check}_{rs\text{CTL}}(\phi)$. 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 $\mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]$ 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 ϕ_2 holds, and all the Ψ-predecessors of the states in X, in which ϕ_1 also holds (Line 5 of Algorithm 3). Firstly, we consider the complexity of computing the intersection $Y_{\phi_1} \cap \mathsf{pre}^{\exists}_{\Psi}(X)$. The operation could be implemented as finding all $w \in [\![\phi_1]\!]$ by checking if $w' = res_A(w \cup \alpha)$ for each $w \in [\![\phi_1]\!]$, $\alpha \in \Psi$, and $w' \in X$.

Algorithm 4: Procedure $\mathsf{check}_{rs\mathrm{CTL}}(\phi)$

1: **if** $\phi \in \mathcal{PV}$ **then**
2: **return** $\{w \in \mathbb{W} \mid \phi \in w\} \cap \mathit{Reach}(\text{ICR-}\mathcal{R})$ // *This is because* $\mathcal{PV} = S$
3: **else if** $\phi = \neg\phi_1$ **then**
4: **return** $\mathit{Reach}(\text{ICR-}\mathcal{R}) \setminus \mathsf{check}_{rs\mathrm{CTL}}(\phi_1)$
5: **else if** $\phi = \phi_1 \vee \phi_2$ **then**
6: **return** $\mathsf{check}_{rs\mathrm{CTL}}(\phi_1) \cup \mathsf{check}_{rs\mathrm{CTL}}(\phi_2)$
7: **else if** $\phi = \mathsf{E}_\Psi \mathsf{X}\phi_1$ **then**
8: **return** $\mathsf{pre}^{\exists}_{\Psi}(\mathsf{check}_{rs\mathrm{CTL}}(\phi_1))$
9: **else if** $\phi = \mathsf{E}_\Psi \mathsf{G}\phi_1$ **then**
10: **return** $\mathsf{checkEG}(\Psi, \phi_1)$
11: **else if** $\phi = \mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]$ **then**
12: **return** $\mathsf{checkEU}(\Psi, \phi_1, \phi_2)$
13: **end if**

This involves three nested loops iterating over $[\![\phi_1]\!], X \subseteq 2^S$ 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_{ϕ_2} 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 $[\![\mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]]\!]$ is $\mathcal{O}(2^{|S|} \cdot (2^{2\cdot|S|} + |\Psi|) \cdot |A| \cdot |S|)$. In the case of $\mathsf{E}_\Psi \mathsf{G}\phi_1$, in each iteration of the loop we find all the Ψ-predecessors of the states in X, in which ϕ_1 also holds (Line 5 of Algorithm 2). Similarly as in the case of $\mathsf{E}_\Psi \mathsf{G}\phi_1$, this is the same as finding all $w \in [\![\phi_1]\!]$ by checking if $w' = res_A(w \cup \alpha)$ for each $w \in [\![\phi_1]\!]$, $\alpha \in \Psi$, 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 $\mathcal{O}(2^{|S|} \cdot (2^{2\cdot|S|} + \mathsf{c}(\phi)) \cdot |A| \cdot |S| \cdot \mathsf{d}(\phi))$.

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

Lemma 3.4.1. *Given an ICRRS* ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ *and an rsCTL formula* ϕ*, the problem of deciding whether* $\mathcal{M}_{\text{ICR-}\mathcal{R}} \models \phi$ *is PSPACE-hard.*

Proof. The proof is by reduction of QSAT[1], which is a known PSPACE-complete problem [Stockmeyer and Meyer, 1973], to the rsCTL model checking problem. The construction used for the reduction is similar to the one of [Laroussinie *et al.*, 2004] for timed automata and TCTL. Let $PV = \{x_1, x_2, \ldots, x_n\}$ be a set of propositional

[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).

variables, β be a Boolean formula over PV and in 3-CNF, and

$$\gamma = \mathcal{Q}_1 x_1 \mathcal{Q}_2 x_2 \dots \mathcal{Q}_n x_n \beta$$

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

$$\overline{PV} = \{\bar{x} \mid x \in PV\}$$

and assume that β is represented as the conjunction of m clauses:

$$\beta = c_1 \wedge c_2 \wedge \dots \wedge c_m$$

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 $0 < i \leq m$, $1 \leq j \leq 3$. For each clause c we define:

$$vars(c) = \{0 < k \leq n \mid x_k \in PV \text{ is in } c\},$$
$$\overline{vars}(c) = \{0 < k \leq n \mid \bar{x}_k \in \overline{PV} \text{ is in } c\}.$$

Next, we define the ICRRS, which we use for the translation. Let $V = \{p_1, \bar{p}_1, \dots, p_n, \bar{p}_n\}$ be a set of the entities that represent the propositional variables and their negations, and $C = \{\hat{c}_1, \hat{c}_2, \dots, \hat{c}_m\}$ 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 $S = V \cup C \cup \{t, h\}$. We define the following sets of reactions:

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

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., $\mathcal{E} = \{p_1, \bar{p}_1, \dots, p_n, \bar{p}_n\}$. We assume that the set of the initial states contains only the empty set, i.e., $S_0 = \{\varnothing\}$. Then, the ICRRS for the described QSAT problem and the formula γ is defined as ICR-$\mathcal{R}_{\text{QSAT}} = ((S, A, \mathcal{E}), S_0)$. We define the following rsCTL formulae for all $1 \leq i \leq n$:

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

$$\phi_{n+1} = \mathsf{E}_{\{\varnothing\}}\mathsf{X}t.$$

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 $0 < i \leq n$. 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.

Finally, it is easy to see that γ is true if and only if $\mathcal{M}_{\text{ICR-}\mathcal{R}_{\text{QSAT}}} \models \phi_1$. □

Lemma 3.4.2. *Given an ICRRS* ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ *and an rsCTL formula* ϕ*, the problem of deciding whether* $\mathcal{M}_{\text{ICR-}\mathcal{R}} \models \phi$ *is in* PSPACE*.*

Proof. To prove that the problem is in PSPACE we show that there exists a nondeterministic algorithm for deciding whether $\mathcal{M}_{\text{ICR-}\mathcal{R}} \models \phi$ that requires at most polynomial space in the size of the input, i.e., the formula ϕ and the ICRRS ICR-$\mathcal{R}$. The proof is similar to the one of [Alur *et al.*, 1993a] for TCTL interpreted over timed graphs.

The algorithm uses the recursive procedure *label*(w, ϕ), which returns *true* iff $\mathcal{M}_{\text{ICR-}\mathcal{R}}, w \models \phi$, 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 $\mathcal{O}\big(|S| \cdot \mathsf{d}(\phi)\big)$.

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

The nondeterministic procedure for checking $\phi = \mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]$ 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*.

The procedure for checking $\phi = \mathsf{E}_\Psi \mathsf{G}\phi_1$ in $w \subseteq S$ is outlined in Algorithm 6. Similarly 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*.

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 Algorithm 5 and Algorithm 6.

Algorithm 5: Nondeterministic procedure for checking $\mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]$

1: $\widehat{w} := w$
2: *checking*:
3: **if** $label(\widehat{w}, \phi_2)$ **then**
4: **return** *true*
5: **end if**
6: **if** $\neg label(\widehat{w}, \phi_1)$ **then**
7: **return** *false*
8: **end if**
9: *guessing*:
10: guess $\widehat{w}' \subseteq S$
11: guess $\alpha \in \Psi$
12: **if** $\widehat{w}' \neq res_A(\widehat{w} \cup \alpha) \vee \neg label(\widehat{w}', \phi_1)$ **then**
13: **goto** *guessing*
14: **else**
15: $\widehat{w} := \widehat{w}'$
16: **goto** *checking*
17: **end if**

To check if $\mathcal{M}_{\text{ICR-}\mathcal{R}} \models \phi$, 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 $\mathcal{O}(\mathsf{d}(\phi))$ 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 $\mathcal{O}\big(|S| \cdot \mathsf{d}(\phi)\big)$. However, if we assume that we include the size of the formula in our space complexity considerations, then the procedure needs space $\mathcal{O}\big((|S| + \mathsf{c}(\phi)) \cdot \mathsf{d}(\phi)\big)$. Therefore, by Savitch's theorem, the deterministic algorithm can be implemented in polynomial space.

□

The following theorem follows directly from Lemma 3.4.1 and Lemma 3.4.2:

Theorem 3.4.3. *The model checking problem for rsCTL is PSPACE-complete.*

This rsCTL verification method can be performed symbolically using binary decision diagrams (BDDs) [Bryant, 1986]. The operations on sets of states, such as union, intersection, and difference, can be carried out efficiently on BDDs representing Boolean functions.

Algorithm 6: Nondeterministic procedure for checking $\mathsf{E}_\Psi \mathsf{G}\phi_1$

1: $\widehat{w} := w$
2: $L := \mathit{false}$
3: **if** $\neg \mathit{label}(\widehat{w}, \phi_1)$ **then**
4: **return** *false*
5: **end if**
6: *guessing*:
7: guess $\widehat{w}' \subseteq S$
8: guess $\alpha \in \Psi$
9: **if** $\widehat{w}' \neq \mathit{res}_A(\widehat{w} \cup \alpha) \vee \neg \mathit{label}(\widehat{w}', \phi_1)$ **then**
10: **goto** *guessing*
11: **end if**
12: **if** $\neg L$ **then**
13: guess $\gamma \in \{\mathit{true}, \mathit{false}\}$
14: **if** γ **then**
15: $\widehat{w}_l := \widehat{w}'$
16: $L := \mathit{true}$
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*

3.5 Bounded model checking using BDDs

Bounded model checking (BMC) is a method, which allows to check existential properties on a fragment of the verified model. Typically, BMC is implemented by reduction to the satisfiability problem (SAT) (we demonstrate such a translation in Chapter 5). However, it is also possible to implement bounded model checking using BDDs [Copty *et al.*, 2001, Cabodi *et al.*, 2002], which does not involve reducing the problem to SAT.

We define rsECTL, which is the existential fragment of rsCTL, where negation can be applied to propositional variables only. This fragment of rsCTL is defined by the following grammar:

$$\phi := \wp \mid \neg\wp \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathsf{E}_\Psi \mathsf{X}\phi \mid \mathsf{E}_\Psi \mathsf{G}\phi \mid \mathsf{E}_\Psi[\phi \mathsf{U} \phi],$$

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

The rsECTL formulae are interpreted in structures, which are obtained from rsCTL models by restricting the set of states. We call these structures *submodels* and

define them as follows.

Definition 3.5.1. Let ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ be an ICRRS and let $\mathcal{M}_{\text{ICR-}\mathcal{R}} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ be the model for ICR-$\mathcal{R}$. Let $U \subseteq \mathbb{W}$ be a subset of the states of $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ containing all the initial states, i.e., $\mathbb{W}_0 \subseteq \mathsf{U}$. The *submodel* $sub_U(\mathcal{M}_{\text{ICR-}\mathcal{R}})$ generated from $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ by U is defined as follows:

$$sub_U(\mathcal{M}_{\text{ICR-}\mathcal{R}}) = (U, \mathbb{W}_0, \longrightarrow_U, L_U),$$

where:

- $\longrightarrow_U = \longrightarrow \cap (U \times 2^{\mathcal{E}} \times U)$,
- $L_U : U \longrightarrow 2^{\mathcal{PV}}$ is defined by $L_U(w) = L(w)$ for all $w \in U$.

Introducing the above restriction on the transition relation may result in sequences, which are not paths in the sense of Definition 3.3.5. This follows from the fact that some states in the submodel may not have successors, which are also in U. This was not the case for the models of rsCTL (Definition 3.3.3) and Lemma 3.3.4 does not hold for submodels. The semantics of rsCTL considers only paths, which are infinite sequences. To be able to interpret rsECTL formulae in submodels, we extend the semantics to take into consideration also maximal finite paths. Firstly, we define a path to be an infinite or a maximal finite sequence.

Definition 3.5.2. Let ICR-$\mathcal{R}$ be an ICRRS and $\mathcal{M}_{\text{ICR-}\mathcal{R}} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ be the model for ICR-$\mathcal{R}$. Let $U \subseteq \mathbb{W}$ such that $sub_U(\mathcal{M}_{\text{ICR-}\mathcal{R}})$ is the submodel of $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ generated by U, and let $\Psi \subseteq 2^{\mathcal{E}}$ such that $\Psi \neq \varnothing$.

- An *finite path* over Ψ in $sub_U(\mathcal{M}_{\text{ICR-}\mathcal{R}})$ is a finite sequence $\sigma = (w_0, \alpha_0, w_1, \alpha_1, \ldots, \alpha_{n-1}, w_n)$ of length $n \in \mathbb{N}$, such that $w_i \xrightarrow{\alpha_i}_U w_{i+1}$, $\alpha_i \in \Psi$ for each $0 \leq i < n$ and the sequence is maximal, i.e., there does not exist $w_{n+1} \in U$ and $\alpha_n \in \Psi$ such that $w_n \xrightarrow{\alpha_n}_U w_{n+1}$.
- An *infinite path* over Ψ in $sub_U(\mathcal{M}_{\text{ICR-}\mathcal{R}})$ is an infinite sequence $\sigma = (w_0, \alpha_0, w_1, \alpha_1, \ldots)$ of states and actions such that $w_i \xrightarrow{\alpha_i}_U w_{i+1}$ and $\alpha_i \in \Psi$ for each $i \geq 0$.

If a sequence σ is a finite or an infinite path over Ψ, we simply call it a *path* over Ψ. We define the length of σ as

$$len(\sigma) = \begin{cases} n+1 & \text{if } \sigma \text{ is finite,} \\ \omega & \text{if } \sigma \text{ is infinite.} \end{cases}$$

Definition 3.5.3. Let σ be a path over $\Psi \subseteq 2^{\mathcal{E}}$ and let $a \in \mathbb{N}$, $b \in \mathbb{Z}$. We define $non^+(b)$ such that $non^+(b) = 1$ if $b \leq 0$ and $non^+(b) = 0$ if $b > 0$. The set of the indices of σ restricted with a and b is defined as follows:

$$I_\sigma[a|b] = \begin{cases} \{i \in \mathbb{N} \mid a \leq i < min(\{len(\sigma), non^+(b) \cdot len(\sigma) + b\})\} & \text{if } \sigma \text{ is finite,} \\ \{i \in \mathbb{N} \mid i \geq a\} & \text{if } \sigma \text{ is infinite.} \end{cases}$$

Example 3.5.4. Let us consider a finite path such that $len(\sigma) = 6$. We can take the path indices obtained by skipping the first two path indices and truncate the path length to 4, by taking the set $I_\sigma[2|4] = \{2, 3\}$. Using negative values for b allows us to skip trailing indices. We can skip the first and the last two path indices, e.g., $I_\sigma[1|{-2}] = \{1, 2, 3\}$. To skip only the first index, we can simply use 0 in place of the argument b, i.e., $I_\sigma[1|0] = \{1, 2, 3, 4, 5\}$. This is the same as $I_\sigma[1|len(\sigma)]$.

For $I_\sigma[0|len(\sigma)]$ and $I_\sigma[0|0]$ we simply write I_σ. The set of all the paths over Ψ is denoted by Π_Ψ. For each $i \in I_\sigma$, the i^{th} state of the path σ is denoted by $\sigma_s(i)$, and for each $i \in I_\sigma[0|{-1}]$ 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, $\Pi_\Psi(w) = \{\sigma \in \Pi_\Psi \mid \sigma_s(0) = w\}$. The set of all the infinite paths over Ψ that start in $w \in \mathbb{W}$ is defined as $\Pi^{inf}_\Psi(w) = \{\sigma \in \Pi_\Psi(w) \mid len(\sigma) = \omega\}$.

Definition 3.5.5. Let ICR-$\mathcal{R}$ be an ICRRS and $\mathcal{M}_{\text{ICR-}\mathcal{R}} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ be the model for ICR-$\mathcal{R}$. Let $U \subseteq \mathbb{W}$ such that $\mathbb{W}_0 \subseteq U$, and $\mathcal{M} = sub_U(\mathcal{M}_{\text{ICR-}\mathcal{R}})$, i.e., $\mathcal{M}$ is the submodel generated from $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ by U. The fact that an rsECTL formula ϕ holds in a state $w \in U$ of $\mathcal{M}$ is denoted by $\mathcal{M}, w \models \phi$, where the relation $\models$ is defined recursively as follows:

$$
\begin{array}{lll}
\mathcal{M}, w \models \wp & \text{iff} & \wp \in L(w) \text{ for } \wp \in \mathcal{PV}, \\
\mathcal{M}, w \models \neg\wp & \text{iff} & \wp \notin L(w) \text{ for } \wp \in \mathcal{PV}, \\
\mathcal{M}, w \models \phi \vee \psi & \text{iff} & \mathcal{M}, w \models \phi \text{ or } \mathcal{M}, w \models \psi, \\
\mathcal{M}, w \models \phi \wedge \psi & \text{iff} & \mathcal{M}, w \models \phi \text{ and } \mathcal{M}, w \models \psi, \\
\mathcal{M}, w \models \mathsf{E}_\Psi \mathsf{X}\phi & \text{iff} & (\exists \sigma \in \Pi_\Psi(w))\ 1 \in I_\sigma \text{ and } \mathcal{M}, \sigma_s(1) \models \phi, \\
\mathcal{M}, w \models \mathsf{E}_\Psi \mathsf{G}\phi & \text{iff} & (\exists \sigma \in \Pi^{inf}_\Psi(w))(\forall i \in I_\sigma)(\mathcal{M}, \sigma_s(i) \models \phi), \\
\mathcal{M}, w \models \mathsf{E}_\Psi[\phi \mathsf{U} \psi] & \text{iff} & (\exists \sigma \in \Pi_\Psi(w))(\exists i \in I_\sigma)\big(\mathcal{M}, \sigma_s(i) \models \psi \\
 & & \quad \text{and } (\forall 0 \leq j < i)\ \mathcal{M}, \sigma_s(j) \models \phi\big).
\end{array}
$$

It is easy to check that the above semantics for infinite paths is identical to the one of Definition 3.3.8. When considering finite paths, the main difference is the requirement that $\mathsf{E}_\Psi \mathsf{G}\phi$ can be satisfied only on an infinite path. In Definition 3.3.8, when referring to the i^{th} state of the path σ the value of i is only required to be non-negative. Here, we limit the scope of i by writing $i \in I_\sigma$, which handles finite and infinite paths.

We describe Algorithm 7 that is used to perform bounded model checking on submodels of $\mathcal{M}_{\text{ICR-}\mathcal{R}}$. The computation stops when the verified formula holds, or all the submodels generated from $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ were obtained.

The algorithm operates on submodels of the model $\mathcal{M}_{\text{ICR-}\mathcal{R}}$ to verify an rsECTL formula ϕ. The complete set of reachable states is obtained by computing the least fixed point according to Algorithm 1. The set of reachable states *Reach* of the model initially contains only the initial states $\mathbb{W}_0$ (line 1). In each iteration (except the last one, when ϕ holds) the set *Reach* is extended with the successors of the states in *Reach* (line 8). Until the algorithm reaches the fixed point, in each iteration it checks if ϕ holds in the submodel $sub_{Reach}(\mathcal{M}_{\text{ICR-}\mathcal{R}})$ (Line 4). If

Algorithm 7: The bounded model checking algorithm used with BDDs

```
1: Reach := 𝕎_0
2: Reach_p := ∅
3: while Reach ≠ Reach_p do
4:    if 𝕎_0 ⊆ [[sub_Reach(M_ICR-R), φ]] then
5:       return true
6:    end if
7:    Reach_p := Reach
8:    Reach := Reach ∪ post_2ε(Reach)
9: end while
10: return false
```

the formula holds, the algorithm terminates returning *true*. For the purpose of checking if the fixed point has been reached i.e., if the set *Reach* has changed from the previous iteration (line 3), we keep its last value in $Reach_p$. When we reach the fixed point, the loop terminates (line 9) and the algorithm returns *false*.

The presented BMC algorithm is complete in the sense that it always terminates with a result indicating whether the verified formula holds in the model.

The following lemma states that we can verify rsECTL formulae on submodels of the original model.

Lemma 3.5.6. *Let ICR-$\mathcal{R}$ be an ICRRS and $\mathcal{M} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ be the model for ICR-$\mathcal{R}$. Let ϕ be an rsECTL formula and let $w \in \mathbb{W}$ be a state of $\mathcal{M}$. Then, $\mathcal{M}, w \models \phi$ iff there exists $U \subseteq \mathbb{W}$ such that $w \in U$ and $sub_U(\mathcal{M}), w \models \phi$.*

Proof. $\rightarrow$: In this direction, follows simply for $U = \mathbb{W}$.
$\leftarrow$: The converse follows by induction on the length of the formula ϕ. The base case is straightforward as the lemma follows directly for the propositional variables and their negations. We assume the statement holds for the subformulae of ϕ. Let $U \subseteq \mathbb{W}$, $w \in U$ and $sub_U(\mathcal{M}), w \models \phi$.

1. Let $\phi = \phi_1 \vee \phi_2$. By the semantics of rsECTL we have $sub_U(\mathcal{M}), w \models \phi_1$ or $sub_U(\mathcal{M}), w \models \phi_2$. By the induction hypothesis and the definition of submodel (Definition 3.5.1), the state w exists also in the model $\mathcal{M}$, and $\mathcal{M}, w \models \phi_1$ or $\mathcal{M}, w \models \phi_2$, thus $\mathcal{M} \models \phi_1 \vee \phi_2$.
2. Let $\phi = \phi_1 \wedge \phi_2$. By the semantics of rsECTL we have $sub_U(\mathcal{M}), w \models \phi_1$ and $sub_U(\mathcal{M}), w \models \phi_2$. By the induction hypothesis and the definition of submodel, the state w exists also in the model $\mathcal{M}$, and $\mathcal{M}, w \models \phi_1$ and $\mathcal{M}, w \models \phi_2$, thus $\mathcal{M} \models \phi_1 \wedge \phi_2$.
3. Let $\phi = \mathsf{E}_\Psi \mathsf{X} \phi_1$. By the semantics of rsECTL there exists a path $\sigma \in \Pi_\Psi(w)$ in $sub_U(\mathcal{M})$ such that $1 \in I_\sigma$ and $sub_U(\mathcal{M}), \sigma_s(1) \models \phi_1$. By the induction hypothesis and the definition of submodel, there exists a path σ' in $\mathcal{M}$ such that σ is the prefix of σ', i.e., $\sigma_s(i) = \sigma'_s(i)$ for each $i \in I_\sigma$ and $\sigma_a(i) = \sigma'_a(i)$

for each $i \in I_\sigma[0|-1]$. Therefore, $1 \in I_{\sigma'}$, $\mathcal{M}, \sigma'_s(1) \models \phi_1$, and $\sigma'_s(0) = w$. It follows that $\mathcal{M}, w \models \mathsf{E}_\Psi \mathsf{X} \phi_1$.

4. Let $\phi = \mathsf{E}_\Psi \mathsf{G} \phi_1$. By the semantics of rsECTL, there exists an infinite path $\sigma \in \Pi^{inf}_\Psi(w)$ in $sub_U(\mathcal{M})$ such that $sub_U(\mathcal{M}), \sigma_s(i) \models \phi_1$ for each $i \in I_\sigma$. By the induction hypothesis and the definition of submodel, the path σ also exists in $\mathcal{M}$, and $\mathcal{M}, \sigma_s(i) \models \phi_1$ for each $i \in I_\sigma$ and $\sigma_s(0) = w$, thus $\mathcal{M}, w \models \mathsf{E}_\Psi \mathsf{G} \phi_1$.

5. Let $\phi = \mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]$. By the semantics of rsECTL there exists a path $\sigma \in \Pi_\Psi(w)$ in $sub_U(\mathcal{M})$ and there exists $i \in I_\sigma$ such that $sub_U(\mathcal{M}), \sigma_s(i) \models \phi_2$ and $sub_U(\mathcal{M}), \sigma_s(j) \models \phi_1$ for each $0 \leq j < i$. By the induction hypothesis and the definition of submodel, there exists a path σ' in $\mathcal{M}$ such that σ is the prefix of σ'. Therefore, $\mathcal{M}, \sigma'_s(i) \models \phi_2$ and $\mathcal{M}, \sigma'_s(j) \models \phi_1$ for each $0 \leq j < i$, and $\sigma'_s(0) = w$, thus $\mathcal{M}, w \models \mathsf{E}_\Psi[\phi_1 \mathsf{U} \phi_2]$.

□

3.6 Encoding ICRRS into Boolean formulae

In this section we provide an encoding of the initialised context restricted reaction systems into Boolean formulae, which we use for symbolic model checking. Firstly, we define the *symbolic model checking problem for rsCTL*.

Let ICR-$\mathcal{R}$ be an ICRRS and ϕ be an rsCTL formula. The symbolic model checking problem for rsCTL consists in deciding whether $\mathcal{M}_{\text{ICR-}\mathcal{R}} \models \phi$; however, the model is not represented explicitly. Instead, we use Boolean formulae to encode the states and the transition relation of $\mathcal{M}_{\text{ICR-}\mathcal{R}}$. To verify the formula ϕ, we traverse the model according to the algorithms defined in Section 3.4.

The general idea of the encoding into Boolean formulae is similar to the one introduced for reaction systems in [Ehrenfeucht and Rozenberg, 2007b], but it differs in how the encoding of the transitions between states is defined.

Let ICR-$\mathcal{R} = ((S, A, \mathcal{E}), S_0)$ be an ICRRS and $\mathcal{M}_{\text{ICR-}\mathcal{R}} = (\mathbb{W}, \mathbb{W}_0, \longrightarrow, L)$ 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 $\mathbf{P} = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$. The actions representing context entities are encoded with the variables $\mathbf{P}^{\mathcal{E}} = \{\mathtt{p}^{\mathcal{E}}_1, \ldots, \mathtt{p}^{\mathcal{E}}_n\}$. To encode the successors in the transition relation, we use the set $\mathbf{P'} = \{\mathtt{p}'_1, \cdots, \mathtt{p}'_n\}$ of the primed variables. The set of reactions producing $e \in S$ is defined as $Prod(e) = \{a \in A \mid e \in P_a\}$. For brevity, we use the following vectors of variables: $\overline{\mathtt{p}} = (\mathtt{p}_1, \ldots, \mathtt{p}_n)$, $\overline{\mathtt{p}}^{\mathcal{E}} = (\mathtt{p}^{\mathcal{E}}_1, \ldots, \mathtt{p}^{\mathcal{E}}_n)$, $\overline{\mathtt{p}}' = (\mathtt{p}'_1, \ldots, \mathtt{p}'_n)$. 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 $1 \leq i \leq n$. 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:

$$\mathsf{St}_w(\overline{\mathbf{p}}) = \Big(\bigwedge_{e \in w} \mathtt{m}(e) \Big) \wedge \Big(\bigwedge_{e \in (S \setminus w)} \neg \mathtt{m}(e) \Big).$$

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

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

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

$$\mathsf{Ct}_\alpha(\overline{\mathbf{p}}^{\mathcal{E}}) = \Big(\bigwedge_{e \in \alpha} \mathtt{m}^{\mathcal{E}}(e) \Big) \wedge \Big(\bigwedge_{e \in (S \setminus \alpha)} \neg \mathtt{m}^{\mathcal{E}}(e) \Big).$$

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:

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

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 $\mathit{Prod}(e) \neq \varnothing$, then

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

- if $\mathit{Prod}(e) = \varnothing$, then

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

Entity production. The function $\mathsf{Pr}_e\big(\overline{\mathbf{p}}, \overline{\mathbf{p}}^{\mathcal{E}}, \overline{\mathbf{p}}'\big)$ 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.

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

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

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

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.

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

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 3.6.1. *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 [McMillan, 1993] 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 3.4). 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. □

3.7 Experimental results

We implemented a model checker for ICRRS, based on the encoding presented in Section 3.6, 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 3.3. It was implemented using the C++ programming language, and it uses CUDD library [Somenzi, 1994] for operations on BDDs.

In the experimental evaluation we use our implementation with the parameters summarised in Table 3.1.

Partitioning of transition relation The encoded transition relation of the verified system can be stored and applied in two different ways: by using a

parameter	description
x	partitioned transition relation
z	reordering of BDD variables

Table 3.1: Parameters of the model checking tool

monolithic or *partitioned* encoding [Burch *et al.*, 1991]. 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 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 Rudell's sifting algorithm [Rudell, 1993] implemented in the CUDD library.

Bounded model checking The implementation uses the BDD-based bounded model checking heuristic for testing rsECTL formulae, as described in Section 3.5. 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. This feature is enabled automatically when verifying rsECTL formulae.

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

3.7.1 Heat shock response model

Firstly, we test our implementation using the heat shock response (HSR) model described in Example 2.2.3 and introduced in [Azimi *et al.*, 2014b]. We define the ICRRS modelling HSR as follows:

$$\text{ICR-}\mathcal{R}_{\text{HSR}} = ((S, A, \{stress, nostress\}), S_0),$$

where S and A are as defined in Example 2.2.3, and

$$S_0 = \{\{hsf, prot, hse, nostress\}, \{hse, prot, hsp{:}hsf, stress\}, \{hsp, prot, hsf_3{:}hse, mfp, hsp{:}mfp, nostress\}\}.$$

The definition of S_0 corresponds to the initial context sets used in the processes analysed in [Azimi *et al.*, 2014b]. In the same paper, six properties essential to 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. The following properties are specified:

P1. *mass-conservation*: if *hse* or hsf_3:*hse* is present, then always *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;

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

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 25MB of memory.

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

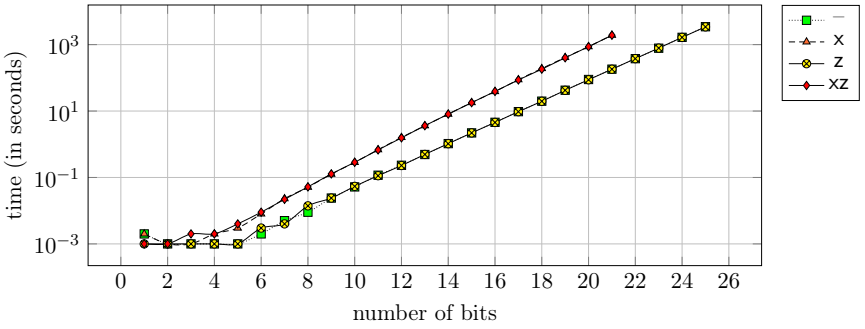

Figure 3.2: Verification results for BC and ψ_1: execution time

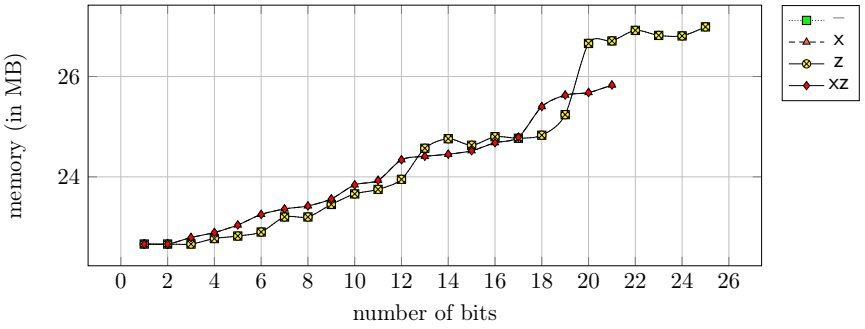

Figure 3.3: Verification results for BC and ψ_1: memory consumption

3.7.2 Binary counter

We test our implementation by verifying properties of the bit counter system (BC) from Example 3.2.2 using the following formulae:

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

The first two formulae that we use in our benchmarks were specified in Example 3.3.11. 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. 3.2–3.3), ψ_2 (Fig. 3.4–3.5), and ψ_4 (Fig. 3.8–3.9) 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

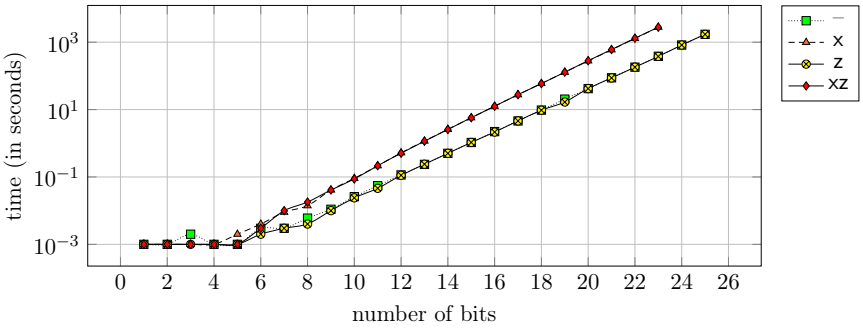

Figure 3.4: Verification results for BC and ψ_2: execution time

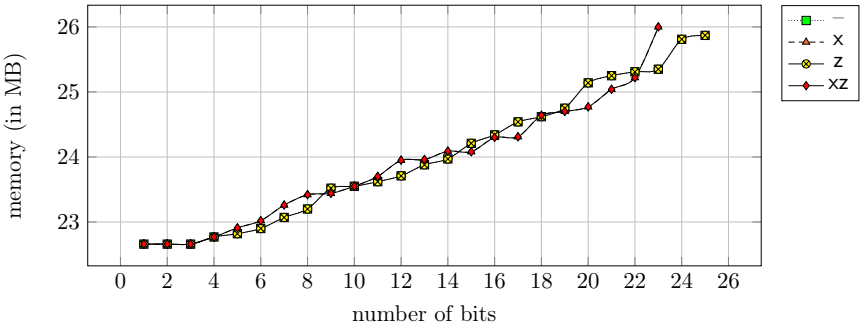

Figure 3.5: Verification results for BC and ψ_2: memory consumption

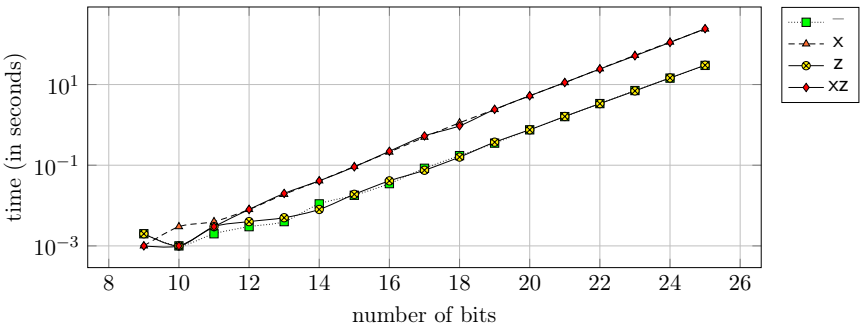

Figure 3.6: Verification results for BC and ψ_3: execution time

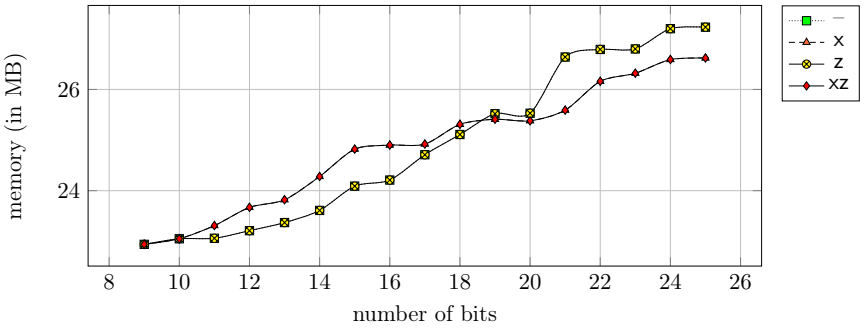

Figure 3.7: Verification results for BC and ψ_3: memory consumption

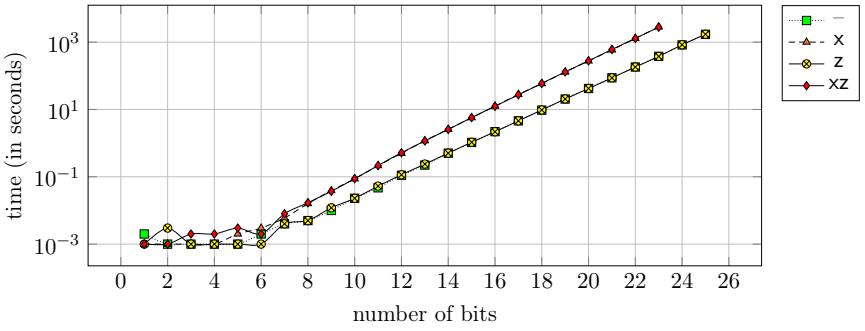

Figure 3.8: Verification results for BC and ψ_4: execution time

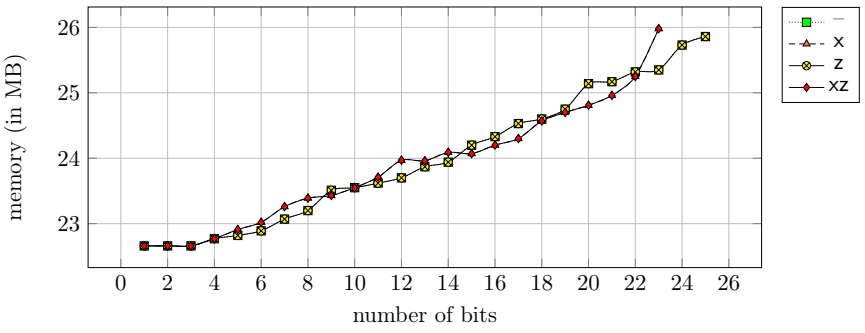

Figure 3.9: Verification results for BC and ψ_4: memory consumption

consumption improvements. In the case of ψ_3 (Figure 3.6–3.7), 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.

3.7.3 Mutual exclusion protocol

Here we use ICRRS to define a mutual exclusion protocol (MUTEX). 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 be any subset of $\{act_1, \ldots, act_n\}$. 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. 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 $i \neq j$,
- $(\{req_i\}, \{act_i\}, \{req_i\})$,
- $(\{req_i, act_i\}, \{act_j \mid j \in \{1, \ldots, n\} \text{ and } j \neq i\} \cup \{lock\}, \{in_i, lock\})$,
- $(\{in_i, act_i\}, \{s\}, \{out_i, done\})$,
- $(\{in_i\}, \{act_i\}, \{in_i\})$.

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

$$\text{ICR-}\mathcal{R}^n_{\text{MUTEX}} = ((S_n, A_n, \{act_1, \ldots, act_n\}), \{\{out_1, \ldots, out_n\}\}).$$

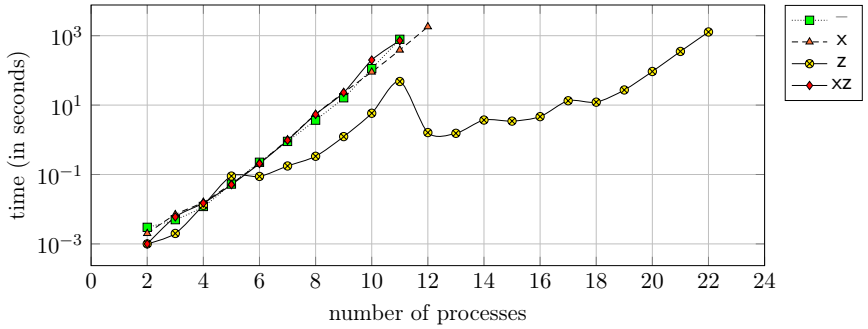

Figure 3.10: Verification results for MUTEX and ψ_1: execution time

In our experiments, we test the following formulae:

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

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 Fig. 3.10–3.15. Our implementation 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.

3.7.4 Abstract pipeline system

We consider an abstract system (APS) which resembles the pipeline protocol of [Peled, 1993]. 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 ICR-$\mathcal{R}_p^n$ = $((S_n, A_n, \mathcal{E}_n), S_0)$, where the background set S_n is defined as:

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

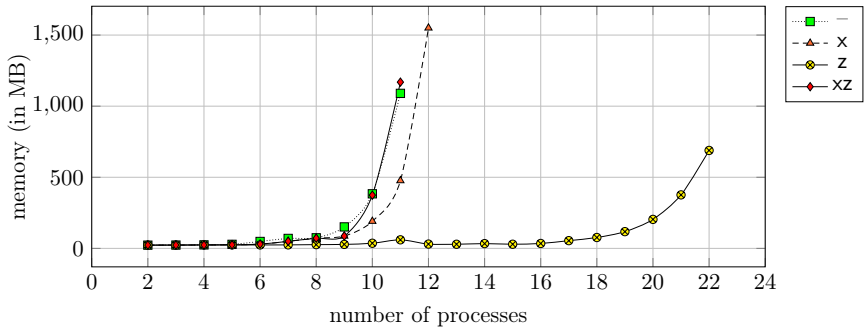

Figure 3.11: Verification results for MUTEX and ψ_1: memory consumption

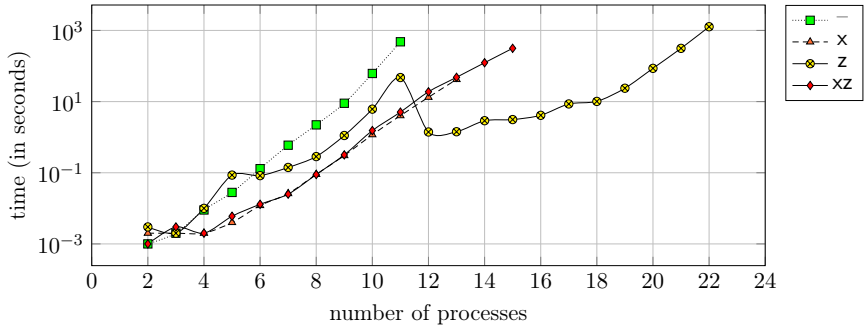

Figure 3.12: Verification results for MUTEX and ψ_2: execution time

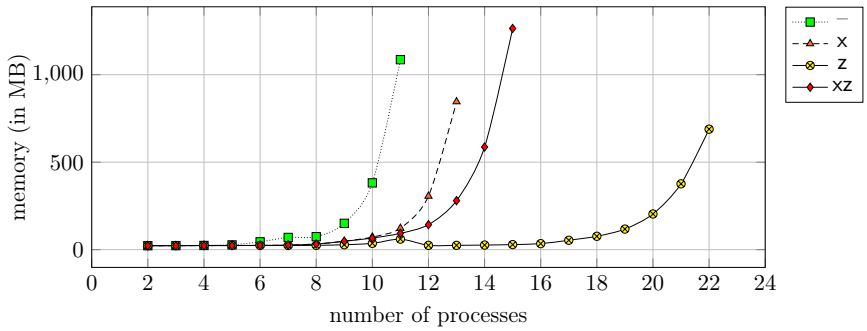

Figure 3.13: Verification results for MUTEX and ψ_2: memory consumption

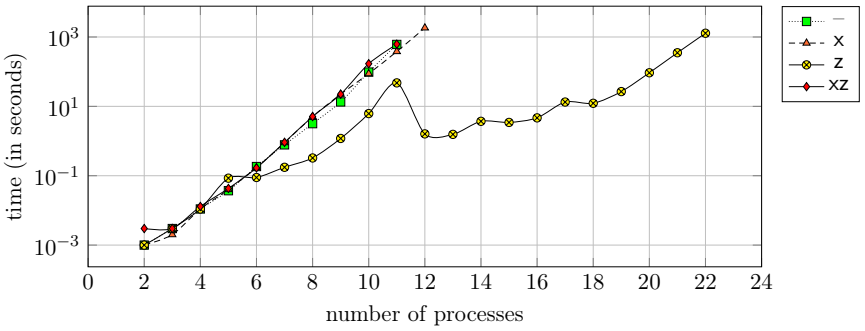

Figure 3.14: Verification results for MUTEX and ψ_3: execution time

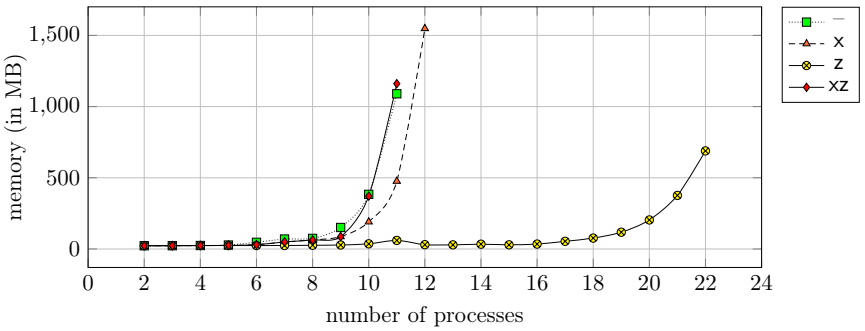

Figure 3.15: Verification results for MUTEX and ψ_3: memory consumption

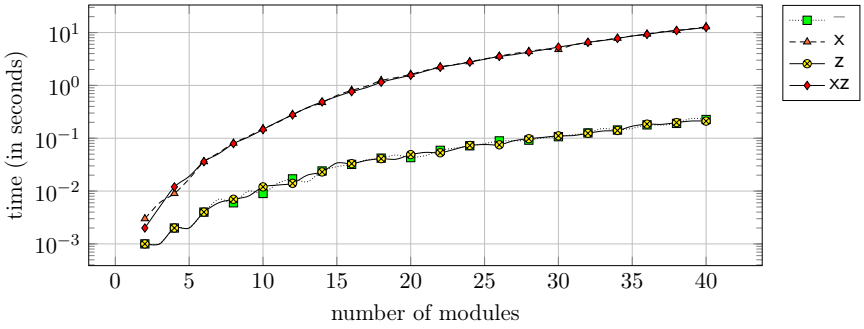

Figure 3.16: Verification results for APS variant 1: execution time

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

$$\begin{aligned} T =& \big\{(\{x\},\{s\},\{a_1\})\big\},\\ M_i =& \big\{(\{a_i\},\{s\},\{y_i\}),(\{y_i\},\{s\},\{y_i\}),\\ & (\{a_i\},\{s\},\{b_i\}),(\{b_i\},\{s\},\{c_i\}),(\{c_i\},\{s\},\{d_i\}),\\ & (\{d_i,y_i\},\{s\},\{a_{i+1}\})\big\} \text{ for } i \in \{1,\ldots,n\},\\ R_n =& \big\{(\{a_{n+1},y_1,\ldots,y_n\},\{s\},\{r\})\big\}. \end{aligned}$$

The reaction of T initialises the process of producing x, the reactions of M_i for $0 < i \leq n$ 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.

The set of reactions of ICR-$\mathcal{R}^n_{\text{APS}}$ 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. $\mathcal{E}_n = \{s\} \cup \{a_i \mid 0 < i \leq n \text{ and } i \text{ is even}\}$.

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

Fig. 3.16–3.17 and Fig. 3.18–3.19 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.

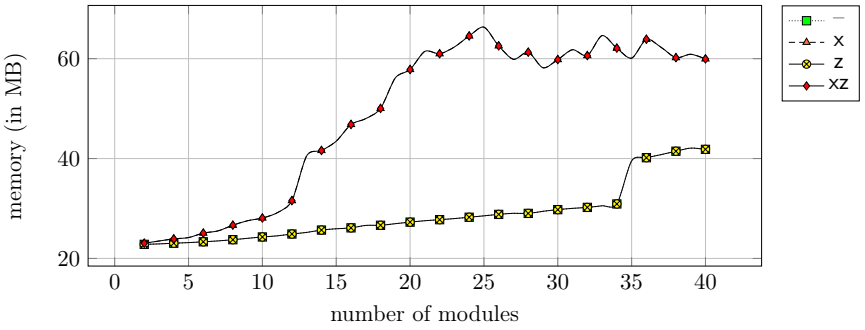

Figure 3.17: Verification results for APS variant 1: memory consumption

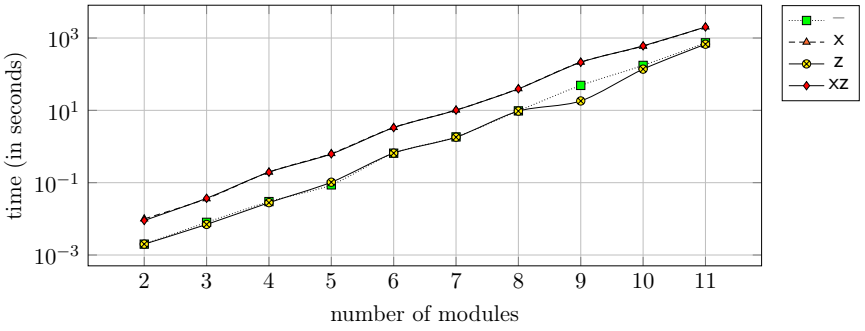

Figure 3.18: Verification results for APS variant 2: execution time

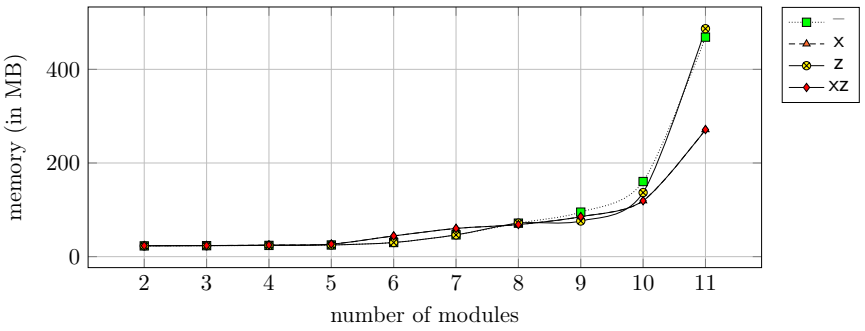

Figure 3.19: Verification results for APS variant 2: memory consumption

3.7.5 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.

3.8 Concluding remarks

We introduced a branching-time 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. We 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 was implemented. This allowed for an experimental evaluation, which proved, despite the complexity of the verification problem, that our approach is promising for possible applications of the proposed verification method.

Chapter 4

Model checking for rsCTLK

In this chapter, we propose an extension of reaction systems that allows for the modelling of distributed and multi-agent systems. We introduce rsCTLK, a logic for specifying temporal-epistemic properties and define its model checking problem.

4.1 Context automata

The method for generating context sequences in context-restricted reaction systems described in Chapter 3 assumes that, for the context sets of the proper context sequences, we always allow the subsets of a fixed subset of the background set. This means that one may be forced to model behaviours that will be irrelevant in reality. For instance, we may want to restrict the context sequences in such a way that some entities never appear at the same time. For example, in the HSR system used in Section 3.7.1, it would be desired for *stress* and *nostress* to never occur simultaneously, since they contradict each other. As demonstrated in Chapter 3, these unrealistic processes can be ignored by selecting only the relevant context sets when specifying the properties of the system in rsCTL. However, to solve this problem at the level of the representation of the system, we use finite automata to generate context sequences.

Definition 4.1.1. A *context automaton* (CA) over Σ, is a triple $\mathfrak{A} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$, where:

- Σ is a finite set of labels,
- $\mathcal{Q}$ is a finite set of *locations*,
- $\mathfrak{q}^{init} \in \mathcal{Q}$ is the *initial location*,
- and $R \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a *transition relation* labelled with elements of Σ.

We assume that R is serial, i.e., for all $\mathfrak{q} \in \mathcal{Q}$ there exists $c \in \Sigma$ and $\mathfrak{q}' \in \mathcal{Q}$ such that $(\mathfrak{q}, c, \mathfrak{q}') \in R$.

Definition 4.1.2. A *context restricted reaction system* (CRRS) is a pair

$$\text{CR-}\mathcal{R} = (\mathcal{R}, \mathfrak{A})$$

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 sequences of states of its *interactive processes*.

Definition 4.1.3. An n-step *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 $z_0 = \mathfrak{q}^{init}$
- $C_0, C_1, \ldots, C_n, D_0, D_1, \ldots, D_n \subseteq S$ with $D_0 = \varnothing$
- $(z_i, C_i, z_{i+1}) \in R$, for every $i \in \{0, \ldots, n-1\}$
- $D_i = res_A(D_{i-1} \cup C_{i-1})$, for every $i \in \{1, \ldots, n\}$.

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 behaviour of CR-$\mathcal{R}$ by recording the states obtained by the evolution of the reaction system $\mathcal{R}$ in the environment modelled by the context automaton $\mathfrak{A}$.

In the following chapters we use context automata to generate context sequences (Section 2.1). In the next section, we show how this notion can be further extended to introduce *extended context automata*, where for the transitions in the context automaton additional conditions on the current state of the reaction system are specified.

4.2 Multi-agent reaction systems

A multi-agent reaction system models a system that consists of many *agents*. Each agent has its own set of reactions defined over a common background set.

Definition 4.2.1. Let $\mathcal{A}$ be an nonempty finite set, the elements of which are called *agents*. A *multi-agent reaction system* (MARS) is a tuple $\mathcal{D} = (S, \{A_i\}_{i \in \mathcal{A}})$, where S is a finite nonempty set, and $A_i \subseteq rac(S)$ for each $i \in \mathcal{A}$.

As with reaction systems, given a MARS $\mathcal{D}$, we refer to S as $\mathcal{D}$'s *background set*. Throughout this chapter, $\mathcal{A} = \{1, \ldots, m\}$ is a fixed set of agents, unless it is specified otherwise. 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 $\mathcal{D}$.

At each transition from a global state to its successor, the environment provides a MARS with a *context*. Each context contains a set of entities for each agent, and specifies the activated agents. 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 provided by the environment, and $\mathbf{C}^a$ denotes the *activated* agents.

The evolution of a MARS in an unconstrained environment is captured by a suitably redefined notion of an interactive process, where the activated agents combine (share) their local states to derive the next local states.

For dealing with tuples we use the following notation: if $x = (x_1, x_2, \ldots, x_n)$ is a tuple, then $x[i] = x_i$, for every $i \leq n$. With $\overline{\varnothing}^m$ we denote the tuple consisting of m empty sets.

We now introduce the notion of an interactive process for MARS.

Definition 4.2.2. Let $\mathcal{D} = (S, \{A_i\}_{i \in \mathcal{A}})$ be a MARS. 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 $\delta = (\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n)$,
- $\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_n \in Ct_{\mathcal{D}}$,
- $\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \in St_{\mathcal{D}}$ with $\mathbf{D}_0 = \overline{\varnothing}^m$,
- for each $k \in \{0, \ldots, n-1\}$ and each $i \in \mathcal{A}$:

$$\mathbf{D}_{k+1}[i] = \begin{cases} res_{A_i}\left(\mathbf{C}_k^c[i] \cup \bigcup_{j \in \mathbf{C}_k^a} \mathbf{D}_k[j]\right) & \text{if } i \in \mathbf{C}_k^a \\ \mathbf{D}_k[i] & \text{if } i \notin \mathbf{C}_k^a \end{cases}.$$

Now we give an intuition for the above definition. For an activated agent $i \in \mathcal{A}$, i.e., $i \in \mathbf{C}_k^a$, to calculate its local successor state $\mathbf{D}_{k+1}[i]$, we take its context set $\mathbf{C}_k^c[i]$ and the union of all the local states of the activated agents (state sharing). If $i \in \mathcal{A}$ is not activated, i.e., $i \notin \mathbf{C}_k^a$, then the local state $\mathbf{D}_k[i]$ remains unchanged, i.e., $\mathbf{D}_{k+1}[i] = \mathbf{D}_k[i]$.

Example 4.2.3. Let us consider a simple example where two agents synchronise to produce an entity by combining their local states. Let $\mathcal{D} = (S, \{A_1, A_2, A_3\})$, where:

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

The entity $\star$ is used as a dummy inhibitor. Consider the 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	$((\{e_1\}, \varnothing, \{e_4\}), \{1, 3\})$	$\overline{\varnothing}^3$
1	$(\overline{\varnothing}^3, \{1, 2\})$	$(\{e_2\}, \varnothing, \{e_5\})$
2	$(\overline{\varnothing}^3, \varnothing)$	$(\varnothing, \{e_3\}, \{e_5\})$

We begin with the state $\mathbf{D}_0 = \overline{\varnothing}^3$ and the context $\mathbf{C}_0 = ((\{e_1\}, \varnothing, \{e_4\}), \{1, 3\})$, where the agents 1 and 3 are active and *receive* e_1 and e_4, respectively. By combining the local states of 1 and 3 we get $\varnothing$. As the result, we get the state $\mathbf{D}_1 = (\{e_2\}, \varnothing, \{e_5\})$, where $res_{A_1}(\{e_1\} \cup \varnothing) = \{e_2\}$ is the local state of the agent 1 and $res_{A_3}(\{e_4\} \cup \varnothing) = \{e_5\}$ is the local state of the agent 3. The local state of the inactive agent 2 remains the same as in $\mathbf{D}_0$, i.e., $\varnothing$.

In the next step, i.e., to obtain $\mathbf{D}_2$, we consider $\mathbf{C}_1 = (\overline{\varnothing}^3, \{1, 2\})$ and $\mathbf{D}_1$ calculated in the previous step. The context $\mathbf{C}_1$ indicates that 1 and 2 are the active agents and they receive no entities, i.e., their the context sets are empty. To calculate the local states of $\mathbf{D}_2$, we need to combine the local states of the active agents from $\mathbf{D}_1$, obtaining the set $\{e_2, e_5\}$. For each active agent, we take the union of its context set with the combined local states. As the result, we get the state $\mathbf{D}_2 = (\varnothing, \{e_3\}, \{e_5\})$, where $res_{A_1}(\varnothing \cup \{e_2, e_5\}) = \varnothing$ is the new local state of the agent 1 and $res_{A_2}(\varnothing \cup \{e_2, e_5\}) = \{e_3\}$ is the new local state of the agent 2. The local state of the agent 3 remains unchanged from $\mathbf{D}_1$, i.e., it retains the previously produced entity e_5 without any reaction being executed locally. Notice that e_2 is required for e_3 to be produced in 2 and the entity is provided by 1 by sharing the local states of the active agents.

The context $\mathbf{C}_2$ is defined only for completeness and it could be used only to calculate the successor of $\mathbf{D}_2$. □

Interactive processes for MARS capture all possible evolutions of a MARS. In practice, however, the behaviour of an environment is constrained and may depend on the current state of the MARS. 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 the agents.

Definition 4.2.4. The *state constraints* $\mathcal{SC}_\mathcal{D}$ are Boolean formulae with propositions in the form of $i.ent$, where $i \in \mathcal{A}$ and $ent \in S$. Their grammar is as follows:

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

Definition 4.2.5. 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:

- $\mathbf{W} \models_{sc} \mathit{true}$,
- $\mathbf{W} \models_{sc} i.ent$ iff $ent \in \mathbf{W}[i]$,
- $\mathbf{W} \models_{sc} \neg\mathfrak{sc}$ 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 4.2.6. An *extended context automaton* (ECA) over a MARS $\mathcal{D}$, 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

– $R \subseteq \mathcal{Q} \times \mathcal{SC}_{\mathcal{D}} \times Ct_{\mathcal{D}} \times \mathcal{Q}$ is a *transition relation.*

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

We say that $\mathfrak{E}$ is *progressive* if for all $\mathfrak{q} \in \mathcal{Q}$ and $\mathbf{W} \in St_{\mathcal{D}}$ there exist $\mathfrak{sc} \in \mathcal{SC}_{\mathcal{D}}$, $\mathbf{C} \in Ct_{\mathcal{D}}$, and $\mathfrak{q}' \in \mathcal{Q}$ such that $(\mathfrak{q}, \mathfrak{sc}, \mathbf{C}, \mathfrak{q}') \in R$ and $\mathbf{W} \models_{sc} \mathfrak{sc}$.

We formalise the notion of a MARS evolving in an environment provided by a progressive ECA.

Definition 4.2.7. A *context-restricted multi-agent reaction system* (CRMARS) is a pair CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ such that $\mathcal{D}$ is a MARS 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 we write $\mathcal{D}(\mathcal{S})$ for $\mathbf{W}$ and $\mathfrak{E}(\mathcal{S})$ for $\mathfrak{q}$.

In CA (Definition 4.1.1) we assume that the transition relation is serial. This allows us to avoid the situation where 'the environment dies', i.e., where the context automaton stops providing contexts to the reaction system. Similarly, given a CRMARS, we assume that the ECA is progressive. In ECAs, for a transition to be enabled, the state constraint associated with the transition must be satisfied. Checking progressiveness of an ECA amounts to checking, if for each location and each $\mathbf{W} \in St_{\mathcal{D}}$ there exists a transition for which its state constraint is satisfied. This means that checking if an ECA is progressive could be involved. Therefore, we take a different approach and instead of checking if a given ECA is progressive, we construct a corresponding progressive ECA.

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

In the above construction, we add a special location $\bot$ with a self-loop, and for each transition we add an additional transition with the state constraint negated. The added transitions lead to $\bot$, provide empty contexts, and do not activate any agents.

The following result follows immediately from the above construction.

Lemma 4.2.8. *$prg(\mathfrak{E})$ is a progressive ECA, for every $\mathfrak{E}$.*

Definition 4.2.9. Let CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ be a CRMARS such that $\mathcal{D} = (S, \{A_i\}_{i \in \mathcal{A}})$ and $\mathfrak{E} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$. An n-step *interactive process in* CR-$\mathcal{D}$ is $\pi = (\zeta, \gamma, \delta)$, where:

– $\zeta = (\mathfrak{q}_0, \mathfrak{q}_1, \ldots, \mathfrak{q}_n)$, $\gamma = (\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_n)$, and $\delta = (\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n)$,
– $\mathfrak{q}_0, \mathfrak{q}_1, \ldots, \mathfrak{q}_n \in \mathcal{Q}$ with $\mathfrak{q}_0 = \mathfrak{q}^{init}$,
– $\mathbf{C}_0, \mathbf{C}_1, \ldots, \mathbf{C}_n \in Ct_{\mathcal{D}}$,
– $\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \in St_{\mathcal{D}}$ with $\mathbf{D}_0 = \overline{\varnothing}^m$,
– for each $k \in \{0, \ldots, n-1\}$ there exists $\mathfrak{sc} \in \mathcal{SC}_{\mathcal{D}}$ such that $\mathbf{D}_k \models_{sc} \mathfrak{sc}$ and $(\mathfrak{q}_k, \mathfrak{sc}, \mathbf{C}_k, \mathfrak{q}_{k+1}) \in R$ and for each $i \in \mathcal{A}$:

$$\mathbf{D}_{k+1}[i] = \begin{cases} res_{A_i}(\mathbf{C}_k^c[i] \cup \bigcup_{j \in \mathbf{C}_k^a} \mathbf{D}_k[j]) & \text{if } i \in \mathbf{C}_k^a, \\ \mathbf{D}_k[i] & \text{if } i \notin \mathbf{C}_k^a. \end{cases}$$

Example 4.2.10. We now introduce the train-gate-controller system (TGC) modelled using CRMARS. It is a commonly considered benchmark when dealing with multi-agent systems [Lomuscio *et al.*, 2009, Męski *et al.*, 2014b]. It consists of $m \geq 2$ trains trying to access a tunnel and a controller that regulates access to the tunnel. At any given time at most one train is allowed in the tunnel. The set of agents is defined as $\mathcal{A} = \{t_1, \ldots, t_m\}$. In this model, the controller is modelled in the context automaton and there is no interaction between the agents. We define the background set of the MARS modelling the system:

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

For each train $i \in \mathcal{A}$ we define the set of its reactions as $A_i = \{a_1^{tr}, a_2^{tr}, a_3^{tr}, a_4^{tr}, a_5^{tr}\}$, where the reactions are defined as follows:

– $a_1^{tr} = (\{out\}, \{\star\}, \{approach\})$,
– $a_2^{tr} = (\{approach\}, \{req\}, \{req\})$,
– $a_3^{tr} = (\{allowed\}, \{\star\}, \{in\})$,
– $a_4^{tr} = (\{in\}, \{\star\}, \{out, leave\})$,
– $a_5^{tr} = (\{req\}, \{in\}, \{req\})$.

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

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* (a_1^{tr}). Any train may request access to the tunnel by producing the entity *req* (a_2^{tr}). If access to the tunnel is requested and, 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 has entered the tunnel (a_3^{tr}). When a train leaves the tunnel it produces *leave* and *out* (a_4^{tr}). 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 (a_5^{tr}).

To define a context-restricted multi-agent 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:

1. $\mathfrak{q}_0 \xrightarrow{true,((\{out\},\ldots,\{out\}),\{t_1,\ldots,t_m\})} \mathfrak{q}_{green}$,
2. for each $t_i \in \mathcal{A}$:
   (a) $\mathfrak{q}_{green} \xrightarrow{\mathfrak{sc},(\overline{\varnothing}^m,\{t_i\})} \mathfrak{q}_{green}$, where $\mathfrak{sc} = \neg \bigvee_{j\in\mathcal{A}} t_j.req$,
   (b) $\mathfrak{q}_{green} \xrightarrow{t_i.req,\mathbf{C}} \mathfrak{q}_{red}$, where: $\mathbf{C}^a = \{t_i\}$ and for each $t_j \in \mathcal{A}$:
   $$\mathbf{C}^c[j] = \begin{cases} \{allowed\} & j = i, \\ \varnothing & j \neq i, \end{cases}$$
   (c) $\mathfrak{q}_{red} \xrightarrow{\mathfrak{sc},(\overline{\varnothing}^m,\{t_i\})} \mathfrak{q}_{red}$, where $\mathfrak{sc} = \neg \bigvee_{t_j\in\mathcal{A}} t_j.leave$,
   (d) $\mathfrak{q}_{red} \xrightarrow{t_i.leave,(\overline{\varnothing}^m,\{t_i\})} \mathfrak{q}_{green}$.

The transition (1) in the context automaton provides the set $\{out\}$ as the initial context sets to all the agents. When the trains are not requesting access to the tunnel, in (2a) the automaton nondeterministically selects and provides the empty context set to an agent which executes its actions. The transition (2b) allows one of the trains to enter the tunnel, when it is requesting access and the tunnel is empty, i.e., the automaton is in $\mathfrak{q}_{green}$. The mutual exclusion is enforced here by using the activated agents set $\mathbf{C}^a$, where only one of the requesting agents is activated, and it receives the *allowed* entity. In (2c), if the trains are not leaving the tunnel, then the automaton activates and provides the empty context set to a nondeterministically selected agent, allowing it to perform an action. If there is a train leaving the tunnel, the automaton transitions from $\mathfrak{q}_{red}$ to $\mathfrak{q}_{green}$ (2d).

Finally, the CRMARS for TGC is defined as:

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

□

Example 4.2.11. We now present a variant of the model from Example 4.2.10, where the controller is modelled as an agent, and the system relies entirely on the communication between the agents. The system consists of $n \geq 2$ trains. The set of agents is defined as $\mathcal{A} = \{t_1, \ldots, t_n, c\}$ and $m = |\mathcal{A}|$. The agents $t_1, \ldots, t_n$ represent trains and c is the controller agent. We define the background set of the MARS:

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

For each train $i \in \{t_1, \ldots, t_n\}$ we define the set of its reactions as $A_i = \{a_1^{tr}, a_2^{tr}, a_3^{tr}, a_4^{tr}, a_5^{tr}\}$, where the reactions are defined as follows:

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

The set $A_n = \{a_1^{ctr}, a_2^{ctr}\}$ consists of the reactions for the controller agent that are defined as follows:

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

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

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

To define the context-restricted multi-agent 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:

- $\mathfrak{q}_0 \xrightarrow{true,(x,\{t_1,\ldots,t_n,c\})} \mathfrak{q}_1$, where x is an m-tuple such that $x[i] = \{out\}$ for $i \in \{1, \ldots, n\}$ and $x[m] = \varnothing$,
- $\mathfrak{q}_1 \xrightarrow{true,(\overline{\varnothing}^m,\{t_i,c\})} \mathfrak{q}_1$ for all $t_i \in \{t_1, \ldots, t_n\}$.

Such a definition of the context automaton allows for the agents to communicate with the controller in pairs only. Note that $\mathfrak{E}_{\text{TGC}}$ can be regarded as an extended *state-oblivious context controller* [Kleijn *et al.*, 2018]. Finally, the CRMARS 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. The access is provided via a nondeterministic choice in the context automaton, and the requesting agent produces *in*. This results in producing *lock* entity by the controller that 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.

□

4.3 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:

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

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

The aim of the logic is to resemble CTLK [Penczek and Lomuscio, 2003], while retaining the expressive power of rsCTL. The grammar uses the propositional and temporal operators of rsCTL (Chapter 3). In place of propositional variables we use $i.ent$, 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.

In contrast to rsCTL defined in Chapter 3, to restrict the scope of the path quantifiers in this logic we use state constraints. One advantage of such an approach is the ability to obtain compact representations for families of sets of entities (sets of the allowed actions) under path quantifiers.

Definition 4.3.1. Let ϕ be an rsCTLK formula. Then, $\mathsf{d}(\phi)$ is the *depth of* ϕ and is defined recursively as follows:

- if $\phi = i.ent$, where $i \in \mathcal{A}$ and $ent \in S$, 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 $\phi \in \{\phi' \vee \phi'', \phi' \wedge \phi'', \mathsf{E}_{\mathfrak{sc}}[\phi'\mathsf{U}\phi'']\}$, then $\mathsf{d}(\phi) = max(\{\mathsf{d}(\phi'), \mathsf{d}(\phi'')\}) + 1$.

Definition 4.3.2. Let CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ where $\mathcal{D} = (S, \{A_i\}_{i\in\mathcal{A}})$ is a multi-agent reaction system and $\mathfrak{E} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ is an extended context automaton over $\mathcal{D}$. 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. $\mathcal{S}_{init} = \big(\mathfrak{q}^{init}, (\varnothing, \ldots, \varnothing)\big) \in St_{\text{CR-}\mathcal{D}}$ 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 $k \in \mathcal{A}$:
   - $\mathbf{S}'[k] = res_{A_i}\left(\mathbf{C}^c[k] \cup \bigcup_{j\in\mathbf{C}^a} \mathbf{S}[j]\right)$ 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 4.3.2 and seriality of the transition relation of $\mathfrak{E}$.

Lemma 4.3.3. *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 4.3.4. 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 $\mathcal{S}_i \xrightarrow{\mathbf{C}_i} \mathcal{S}_{i+1}$ and $\mathbf{C}_i^c \models_{sc} \mathfrak{sc}$ for each $i \geq 0$.

The set of all the paths over $\mathfrak{sc}$ is denoted by $\Pi^{inf}_{\mathfrak{sc}}$. 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 $\Pi^{inf}_{\mathfrak{sc}}(\mathcal{S})$ we denote the set of all the paths over $\mathfrak{sc}$ that start in $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$, i.e., $\Pi^{inf}_{\mathfrak{sc}}(\mathcal{S}) = \{\sigma \in \Pi^{inf}_{\mathfrak{sc}} \mid \sigma_s(0) = w\}$.

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 an $\mathfrak{sc}$-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 4.3.5. 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.,

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

We also write $Reach(\mathcal{M}_{\text{CR-}\mathcal{D}})$ to denote $Reach(\text{CR-}\mathcal{D})$.

Definition 4.3.6. 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:

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

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

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 4.3.7. 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 rsCTLK 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:

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

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

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

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 say 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 4.3.8. Here we specify some rsCTLK properties of the TGC system presented in Example 4.2.10.

1. It is possible for each train to eventually enter the tunnel, in one step from receiving the *allowed* entity in the context:

$$\phi_1 = \bigwedge_{t_i \in \{t_1,\ldots,t_m\}} \mathsf{EF}\left(\mathsf{E}_{t_i.allowed}\mathsf{X}\left(t_i.in\right)\right).$$

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:

$$\phi_2 = \mathsf{AG}\left(t_i.in \implies \mathrm{K}_{t_i}\left(\bigwedge_{\substack{t_j \in \{t_1,\ldots,t_m\},\\ i \neq j}} \neg t_j.in\right)\right).$$

3. We take all the states of all the paths such that the trains do not receive the *allowed* entity, with the exception of the i^{th} train. In these states, the i^{th} train knows that no other train is in the tunnel:

$$\phi_3 = \mathsf{A}_{\mathfrak{sc}}\mathsf{G}\left(\mathrm{K}_{t_i}\left(\bigwedge_{\substack{t_j \in \{t_1,\ldots,t_m\}\\ j \neq i}} \neg t_j.in\right)\right),$$

where:

$$\mathfrak{sc} = \bigwedge_{\substack{t_j \in \{t_1,\ldots,t_m\}\\ j \neq i}} \neg t_j.allowed.$$

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

$$\phi_4 = \mathsf{AG}\left(\mathrm{K}_{t_i} \bigwedge_{\substack{t_j, t_k \in \{t_1,\ldots,t_m\}\\ j<k}} \neg\left(t_j.in \wedge t_k.in\right)\right).$$

5. 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:

$$\phi_5 = \mathsf{AG}\left(t_i.in \implies \mathrm{C}_{\{t_1,\ldots,t_m\}}\left(\bigwedge_{\substack{t_j \in \{t_1,\ldots,t_m\}\\ i \neq j}} \neg t_j.in\right)\right).$$

□

4.4 Model checking for rsCTLK

In this section we describe a model checking method for rsCTLK, which is based on computing fixed points. To calculate the set of the reachable states and the results for temporal operators we use algorithms similar to the ones presented in

Section 3.4. The difference is in how the set of successor and predecessor states are defined. In this section we restrict the sets of states to only those, which are obtained via transitions restricted with state constraints, whereas in Chapter 3 we used sets of actions. 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.

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 8: The algorithm for computing the set $\mathit{Reach}(\text{CR-}\mathcal{D})$

1: $X := \mathcal{S}_{init}$
2: $X_p := \emptyset$
3: **while** $X \neq X_p$ **do**
4: $\quad X_p := X$
5: $\quad X := X \cup \mathsf{post}_{Ct_{\mathcal{D}}}(X)$
6: **end while**
7: **return** X

Algorithm 8 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:

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

Let ϕ_1, ϕ_2 be some rsCTLK formulae. For the non-epistemic formulae of rsCTLK

the sets of states in which they hold are defined similarly as in Section 3.4:

$$\begin{aligned}
[\![\neg\phi_1]\!] &\stackrel{def}{=} \mathit{Reach}(\text{CR-}\mathcal{D}) \setminus [\![\phi_1]\!],\\
[\![\phi_1 \vee \phi_2]\!] &\stackrel{def}{=} [\![\phi_1]\!] \cup [\![\phi_2]\!],\\
[\![\phi_1 \wedge \phi_2]\!] &\stackrel{def}{=} [\![\phi_1]\!] \cap [\![\phi_2]\!],\\
[\![\mathsf{E}_{\mathfrak{sc}}\mathsf{X}\phi_1]\!] &\stackrel{def}{=} \mathsf{pre}^{\exists}_{\mathfrak{sc}}([\![\phi_1]\!]),\\
[\![\mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi_1]\!] &\stackrel{def}{=} \nu X.\big([\![\phi_1]\!] \cap \mathsf{pre}^{\exists}_{\mathfrak{sc}}(X)\big),\\
[\![\mathsf{E}_{\mathfrak{sc}}[\phi_1\mathsf{U}\phi_2]]\!] &\stackrel{def}{=} \mu X.\big([\![\phi_2]\!] \cup ([\![\phi_1]\!] \cap \mathsf{pre}^{\exists}_{\mathfrak{sc}}(X))\big).
\end{aligned}$$

The above definitions differ from the ones presented in Section 3.4 in how the predecessors are defined – here we restrict the set of predecessors with state constraints. See Algorithm 2 and 3 in Chapter 3 for the pseudo-code of the procedures for calculating $[\![\mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi_1]\!]$ and $[\![\mathsf{E}_{\mathfrak{sc}}[\phi_1\mathsf{U}\phi_2]]\!]$, respectively.

For a group of agents $\Gamma \subseteq \mathcal{A}$ we introduce the set of Γ-*neighbours* of the states in W:

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

The sets for the epistemic operators [Fagin *et al.*, 2003] are defined as follows:

$$\begin{aligned}
[\![\overline{\mathrm{K}}_i\phi_1]\!] &\stackrel{def}{=} \mathsf{nb}_{\{i\}}([\![\phi_1]\!]),\\
[\![\overline{\mathrm{C}}_\Gamma\phi_1]\!] &\stackrel{def}{=} \mu X.\,([\![\phi_1]\!] \cup \mathsf{nb}_\Gamma(X))\,.
\end{aligned}$$

To compute the set of states for $\overline{\mathrm{K}}_i\phi_1$ we find all the i-neighbours of the states in which ϕ_1 holds. In the case of $\overline{\mathrm{C}}_\Gamma\phi_1$, to obtain the set of states in which the formula holds, we calculate the least fixed point. The procedures for $[\![\overline{\mathrm{K}}_i\phi_1]\!]$ and $[\![\overline{\mathrm{C}}_\Gamma\phi_1]\!]$ are presented in Algorithm 9 and 10, respectively.

Algorithm 9: $\mathsf{checkNK}(i, \phi_1)$

1: $X := \mathsf{check}_{rs\text{CTLK}}(\phi_1)$
2: **return** $\mathsf{nb}_{\{i\}}(X)$

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 11.

Definition 4.4.1. Given CR-$\mathcal{D}$ and an rsCTLK formula ϕ, *rsCTLK model checking* is the problem of deciding whether $\mathcal{M}_{\text{CR-}\mathcal{D}} \models \phi$.

Lemma 4.4.2. *The rsCTLK model checking problem is* PSPACE*-hard.*

Algorithm 10: $\mathsf{checkNC}(\Gamma, \phi_1)$

$X := \varnothing$, $X_p := \mathit{Reach}(\text{CR-}\mathcal{D})$
$Y_{\phi_1} = \mathsf{check}_{rs\text{CTLK}}(\phi_1)$
while $X \neq X_p$ **do**
  $X_p := X$
  $X := \mathsf{nb}_\Gamma(Y_{\phi_1} \cup X)$
end while
return X

Algorithm 11: $\mathsf{check}_{rs\text{CTLK}}(\phi)$

if $\phi = i.ent$ **then**
  return $\{\mathcal{S} \in \mathit{St}_{\text{CR-}\mathcal{D}} \mid ent \in \mathcal{D}(\mathcal{S})[i]\} \cap \mathit{Reach}(\text{CR-}\mathcal{D})$
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 $\phi = \overline{\mathrm{K}}_i\phi_1$ **then**
  return $\mathsf{checkNK}(i, \phi_1)$
else if $\phi = \overline{\mathrm{C}}_\Gamma\phi_1$ **then**
  return $\mathsf{checkNC}(\Gamma, \phi_1)$
end if

Proof. Follows from the fact that QSAT can be reduced to the rsCTLK model checking problem. It is easy to see that every initialised context restricted reaction system (ICRRS) can be translated into CRMARS with a single agent and rsCTL is a subset of rsCTLK. Therefore, the reduction is similar to the one for ICRRS and rsCTL (Chapter 3). □

Lemma 4.4.3. *The rsCTLK 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 for rsCTL and initialised context-restricted reaction systems.

The algorithm uses a recursive procedure $label(\mathcal{S}, \phi)$, 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 procedures for checking $\phi = \mathsf{E}_{\mathfrak{sc}}[\phi_1 \mathsf{U} \phi_2]$ and $\phi = \mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi_1$ in $\mathcal{S} \subseteq St_{\text{CR-}\mathcal{D}}$ are outlined in Algorithm 12 and 13, respectively. The algorithms are similar to the ones presented in the proof of Lemma 3.4.2. However, the successor state is guessed differently: the algorithms nondeterministically select a state and a context $\mathbf{C} \in Ct_{\mathcal{D}}$ of a path over $\mathfrak{sc}$ (e.g., Line 10 of Algorithm 12), 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 a context and a state are selected again.

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 14. 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 15. 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, the procedure nondeterministically selects a state $\widehat{\mathcal{S}}_r$ of that sequence. The guessing of the sequence stops when $\widehat{\mathcal{S}}_r$ is the newly guessed state. For simplicity of the

Algorithm 12: Nondeterministic procedure for checking $\mathsf{E}_{\mathfrak{sc}}[\phi_1 \mathsf{U} \phi_2]$

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 $\widehat{\mathcal{S}}' \in St_{\text{CR-}\mathcal{D}}$ and $\mathbf{C} \in Ct_{\mathcal{D}}$
11: **if** $\mathbf{C}^c \not\models_{sc} \mathfrak{sc}$ **then**
12: **goto** *guessing*
13: **else if** $\neg label(\widehat{\mathcal{S}}', \phi_1)$ **then**
14: **goto** *guessing*
15: **else if** $\big((\mathcal{S}, \mathbf{C}, \widehat{\mathcal{S}}') \notin \longrightarrow\big)$ **then**
16: **goto** *guessing*
17: **end if**
18: $\widehat{\mathcal{S}} := \widehat{\mathcal{S}}'$
19: **goto** *checking*

presentation, this part of the procedure is not included in Algorithm 12 and Algorithm 13. 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 CRMARS 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 4.4.2 and Lemma 4.4.3.

Theorem 4.4.4. *The rsCTLK model checking problem is* PSPACE*-complete.*

Algorithm 13: Nondeterministic procedure for checking $\mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi_1$

$\widehat{\mathcal{S}} := \mathcal{S}$
$L := \mathit{false}$
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 14: Nondeterministic procedure for checking $\overline{\mathsf{K}}_i\phi_1$

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 15: Nondeterministic procedure for checking $\overline{\mathrm{C}}_\Gamma \phi_1$

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*

4.5 Bounded model checking using BDDs

Similarly as for rsCTL (Section 3.5), we provide BDD-based bounded model checking for a fragment of rsCTLK. We define rsECTLK, which is the existential fragment of rsCTLK, where negation can be applied only to the entities belonging to agents. This fragment of rsCTLK is defined by the following grammar:

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

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

Definition 4.5.1. Let CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ be a CRMARS and let

$$\mathcal{M}_{\text{CR-}\mathcal{D}} = (St_{\text{CR-}\mathcal{D}}, \mathcal{S}_{init}, \longrightarrow)$$

be the model for CR-$\mathcal{D}$. Let $U \subseteq St_{\text{CR-}\mathcal{D}}$ be a subset of the states of $\mathcal{M}_{\text{CR-}\mathcal{D}}$ containing all the initial states, i.e., $\mathcal{S}_{init} \subseteq \mathsf{U}$. The *submodel* $sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}})$ generated from $\mathcal{M}_{\text{CR-}\mathcal{D}}$ by U is defined as $sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}}) = (U, \mathcal{S}_{init}, \longrightarrow_U)$, where: $\longrightarrow_U = \longrightarrow \cap\, (U \times Ct_\mathcal{D} \times U)$.

Similarly to BMC for rsCTL (Section 3.5), with the above restriction some states in the submodel may not have successors, which are also in U. Therefore, we define a path to be an infinite or a maximial finite sequence.

Definition 4.5.2. Let CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ be a CRMARS and $\mathcal{M}_{\text{CR-}\mathcal{D}}$ be the model for CR-$\mathcal{D}$. Let $U \subseteq St_{\text{CR-}\mathcal{D}}$ such that $sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}})$ is the submodel of $\mathcal{M}_{\text{CR-}\mathcal{D}}$ generated by U.

– A *finite path* over $\mathfrak{sc} \in \mathcal{SC}_\mathcal{D}$ in $sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}})$ is a finite sequence $\sigma = (\mathcal{S}_0, \mathbf{C}_0, \mathcal{S}_1, \mathbf{C}_1, \ldots, \mathbf{C}_{n-1}, \mathcal{S}_n)$ of length $n \in \mathbb{N}$, such that $\mathcal{S}_i \xrightarrow{\mathbf{C}_i}_U \mathcal{S}_{i+1}$, $\mathbf{C}_i \models_{sc} \mathfrak{sc}$ for $0 \leq i < n$ and the sequence is maximal, i.e., there does not exist $\mathbf{C}_n \in Ct_\mathcal{D}$ and $\mathcal{S}_{n+1} \in U$ such that $\mathbf{C}_n \models_{sc} \mathfrak{sc}$ and $\mathcal{S}_n \xrightarrow{\mathbf{C}_i}_U \mathcal{S}_{n+1}$.

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

If a sequence σ is a finite or an infinite path over $\mathfrak{sc} \in \mathcal{SC}_{\mathcal{D}}$, we simply call it a *path* over $\mathfrak{sc} \in \mathcal{SC}_{\mathcal{D}}$. We define the length of σ as

$$len(\sigma) = \begin{cases} n+1 & \text{if } \sigma \text{ is finite,} \\ \omega & \text{if } \sigma \text{ is infinite.} \end{cases}$$

We use notation for the sets of path indices introduced in Definition 3.5.3 and we write I_σ for $I_\sigma[0|len(\sigma)]$ and $I_\sigma[0|0]$.

The set of all the paths over $\mathfrak{sc}$ is denoted by $\Pi_{\mathfrak{sc}}$. For each $i \in I_\sigma$, the i^{th} state of the path σ is denoted by $\sigma_s(i)$, and for each $i \in I_\sigma[0|-1]$ 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., $\Pi_{\mathfrak{sc}}(\mathcal{S}) = \{\sigma \in \Pi_{\mathfrak{sc}} \mid \sigma_s(0) = w\}$. The set of all the infinite paths over $\mathfrak{sc}$ that start in $\mathcal{S}$ is defined as $\Pi^{inf}_{\mathfrak{sc}}(\mathcal{S}) = \{\sigma \in \Pi_{\mathfrak{sc}}(\mathcal{S}) \mid len(\sigma) = \omega\}$.

The epistemic indistinguishability relation uses the definition of the reachable states. The set $Reach(sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}}))$ of the reachable states of the submodel $sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}})$ is defined as $Reach(sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}})) \stackrel{def}{=} Reach(\mathcal{M}_{\text{CR-}\mathcal{D}}) \cap U$. For each agent $i \in \mathcal{A}$, the *epistemic indistinguishability* relation $\sim_i \subseteq U \times U$ is defined by: $\mathcal{S} \sim_i \mathcal{S}'$ iff $\mathcal{D}(\mathcal{S})[i] = \mathcal{D}(\mathcal{S}')[i]$ and $\mathcal{S}, \mathcal{S}' \in Reach(sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}}))$.

Definition 4.5.3. Let CR-$\mathcal{D} = (\mathcal{D}, \mathfrak{E})$ be a CRMARS and $\mathcal{M}_{\text{CR-}\mathcal{D}}$ be the model for CR-$\mathcal{D}$. Let $U \subseteq St_{\text{CR-}\mathcal{D}}$ such that $sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}})$ is the submodel of $\mathcal{M}_{\text{CR-}\mathcal{D}}$ generated by U. The fact that an rsECTLK formula ϕ holds in $\mathcal{S} \in U$ is denoted by $sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}}), \mathcal{S} \models \phi$, or simply by $\mathcal{S} \models \phi$ when $sub_U(\mathcal{M}_{\text{CR-}\mathcal{D}})$ is clear from the context. The relation $\models$ is defined recursively as follows:

$$\begin{array}{lll}
\mathcal{S} \models i.ent & \text{iff} & ent \in \mathcal{D}(\mathcal{S})[i], \\
\mathcal{S} \models \neg i.ent & \text{iff} & ent \notin \mathcal{D}(\mathcal{S})[i], \\
\mathcal{S} \models \phi \vee \psi & \text{iff} & \mathcal{S} \models \phi \text{ or } \mathcal{S} \models \psi, \\
\mathcal{S} \models \phi \wedge \psi & \text{iff} & \mathcal{S} \models \phi \text{ and } \mathcal{S} \models \psi, \\
\mathcal{S} \models \mathsf{E}_{\mathfrak{sc}}\mathsf{X}\phi & \text{iff} & (\exists \sigma \in \Pi_{\mathfrak{sc}}(\mathcal{S}))\ 1 \in I_\sigma \text{ and } \sigma_s(1) \models \phi, \\
\mathcal{S} \models \mathsf{E}_{\mathfrak{sc}}\mathsf{G}\phi & \text{iff} & (\exists \sigma \in \Pi^{inf}_{\mathfrak{sc}}(\mathcal{S}))(\forall i \in I_\sigma)(\sigma_s(i) \models \phi), \\
\mathcal{S} \models \mathsf{E}_{\mathfrak{sc}}[\phi \mathsf{U} \psi] & \text{iff} & (\exists \sigma \in \Pi_{\mathfrak{sc}}(\mathcal{S}))(\exists i \in I_\sigma)\big(\sigma_s(i) \models \psi \\
 & & \quad \text{and } (\forall 0 \leq j < i)\ \sigma_s(j) \models \phi\big), \\
\mathcal{S} \models \overline{\mathrm{K}}_i\phi & \text{iff} & (\exists \mathcal{S}' \in St_{\text{CR-}\mathcal{D}})(\mathcal{S} \sim_i \mathcal{S}' \text{ and } \mathcal{S}' \models \phi), \\
\mathcal{S} \models \overline{\mathrm{C}}_\Gamma\phi & \text{iff} & (\exists \mathcal{S}' \in St_{\text{CR-}\mathcal{D}})(\mathcal{S} \sim^{\mathrm{C}}_\Gamma \mathcal{S}' \text{ and } \mathcal{S}' \models \phi).
\end{array}$$

It is easy to check that the above semantics for infinite paths is identical to the one of Definition 4.3.7.

To implement bounded model checking on submodels of $\mathcal{M}_{\text{CR-}\mathcal{D}}$ we use Algorithm 7, which we recall as Algorithm 16, where we use the notation for CRMARS.

Algorithm 16: The BDD-based BMC algorithm for rsECTLK

1: $Reach := \mathcal{S}_{init}$
2: $Reach_p := \varnothing$
3: **while** $Reach \neq Reach_p$ **do**
4: **if** $\mathcal{S}_{init} \subseteq [\![sub_{Reach}(\mathcal{M}_{\text{CR-}\mathcal{D}}), \phi]\!]$ **then**
5: **return** *true*
6: **end if**
7: $Reach_p := Reach$
8: $Reach := Reach \cup \mathsf{post}_{true}(Reach)$
9: **end while**
10: **return** *false*

The algorithm operates on submodels of the model $\mathcal{M}_{\text{CR-}\mathcal{D}}$ to verify an rsECTLK formula ϕ. The following lemma states that we can verify rsECTLK formulae on submodels of the original model.

Lemma 4.5.4. *Let CR-$\mathcal{D}$ be a CRMARS, $\mathcal{M}_{\text{CR-}\mathcal{D}} = (St_{\text{CR-}\mathcal{D}}, \mathcal{S}_{init}, \longrightarrow)$ be the model for CR-$\mathcal{D}$, let ϕ be an rsECTLK formula, and let $\mathcal{S} \in St_{\text{CR-}\mathcal{D}}$ be some state of the model. Then, $\mathcal{M}, \mathcal{S} \models \phi$ iff there exists $U \subseteq St_{\text{CR-}\mathcal{D}}$ such that $\mathcal{S} \in U$ and $sub_U(\mathcal{M}), \mathcal{S} \models \phi$.*

Proof. $\rightarrow$: In this direction, follows simply for $U = St_{\text{CR-}\mathcal{D}}$.
$\leftarrow$: The converse follows by induction on the length of the formula ϕ. The base case is straightforward as the lemma follows directly for $i.ent$ and $\neg i.ent$. We assume the statement holds for the subformulae of ϕ.

Let $U \subseteq St_{\text{CR-}\mathcal{D}}$, $\mathcal{S} \in U$, and $sub_U(\mathcal{M}), \mathcal{S} \models \phi$. We focus on the epistemic operators. For the remaining operators the proof follows as for Lemma 3.5.6.

1. Let $\phi = \overline{\mathrm{K}}_i\phi_1$ and $i \in \mathcal{A}$. By the semantics of rsECTLK we have that there exists $\mathcal{S}' \in St_{\text{CR-}\mathcal{D}}$ such that $\mathcal{S} \sim_i \mathcal{S}'$ and $sub_U(\mathcal{M}), \mathcal{S}' \models \phi_1$. By the induction hypothesis and the definition of submodel (Definition 4.5.1), the states $\mathcal{S}$ and $\mathcal{S}'$ exist in $\mathcal{M}$, and $\mathcal{M}, \mathcal{S}' \models \phi_1$ and $\mathcal{S} \sim_i \mathcal{S}'$, thus $\mathcal{M}, \mathcal{S} \models \overline{\mathrm{K}}_i\phi_1$.
2. Let $\phi = \overline{\mathrm{C}}_\Gamma\phi_1$ and $\Gamma \subseteq \mathcal{A}$. By the semantics of rsECTLK we have that there exists $\mathcal{S}' \in St_{\text{CR-}\mathcal{D}}$ such that $\mathcal{S} \sim_\Gamma^{\mathrm{C}} \mathcal{S}'$ and $sub_U(\mathcal{M}), \mathcal{S}' \models \phi_1$. By the induction hypothesis and the definition of submodel, the states $\mathcal{S}$ and $\mathcal{S}'$ exist in $\mathcal{M}$, and $\mathcal{M}, \mathcal{S}' \models \phi_1$ and $\mathcal{S} \sim_\Gamma^{\mathrm{C}} \mathcal{S}'$, thus $\mathcal{M}, \mathcal{S} \models \overline{\mathrm{C}}_\Gamma\phi_1$.

□

4.6 Boolean encoding

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

This encoding differs from the encoding for the models of rsCTL presented in Chapter 3. For the rsCTLK models we need to represent the local states of all the agents. In this chapter we also control the context sequences via an extended context automaton and this involves encoding the local states of the automaton. Moreover, we represent richer contexts that encode separate context sets for all the agents, but also indicate, which agents are active. To this aim, for each agent we introduce a variable indicating activity of the agent.

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 $\mathfrak{q} \in \mathcal{Q}$ of $\mathfrak{E}$ we write $\mathtt{e}(\mathfrak{q})$, and to denote its primed encoding used to encode that $\mathfrak{q}$ is a successor we write $\mathtt{e}'(\mathfrak{q})$. 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 straightforward.

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 $St_{\text{CR-}\mathcal{D}}$:

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

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:

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

Entity production condition. Let $j \in \mathcal{A}$. 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

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

– 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.

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

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.

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

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

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

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.

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

4.7 Experimental results

The proposed model checking method has been implemented in our reaction systems model checking toolkit. The implementation uses C++ as the implementation

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

Table 4.1: Parameters of the model checking tool

language and the CUDD library [Somenzi, 1994] for creating binary decision diagrams and handling the operations on them. The tool uses the encoding presented in Section 4.6 and implements the algorithms from Section 4.4.

In the experimental evaluation we use our implementation with the parameters summarised in Table 4.1. The implementation provides the same features as described in Section 3.7. Bounded model checking for rsECTLK was described in Section 4.5. For reordering of the BDD variables we use the group sifting algorithm of [Panda and Somenzi, 1995] implemented in the CUDD library. All of the rsCTLK formulae used in our benchmarks hold in the tested models.

4.7.1 Train-gate-controller

We test our implementation by verifying the following properties of the train-gate-controller system from Example 4.2.10 in all the configurations of the described implementation parameters:

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

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

$$\phi_3 = \mathsf{AG}\left(t_1.in \implies \mathrm{K}_{t_1}\left(\bigwedge_{t_j \in \{t_2,\ldots,t_m\}} \neg t_j.in\right)\right),$$

$$\phi_4 = \mathsf{AG}\left(t_1.in \implies \mathrm{C}_{\mathcal{A}}\left(\bigwedge_{t_j \in \{t_2,\ldots,t_m\}} \neg t_j.in\right)\right).$$

The experimental results are presented in Fig. 4.1–4.8. 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. We consider values of the scaling parameter between 0 and 20.

4.7.2 Distributed abstract pipeline

Here we introduce distributed abstract pipeline (DAP) system similar to the abstract pipeline system introduced in Chapter 3.

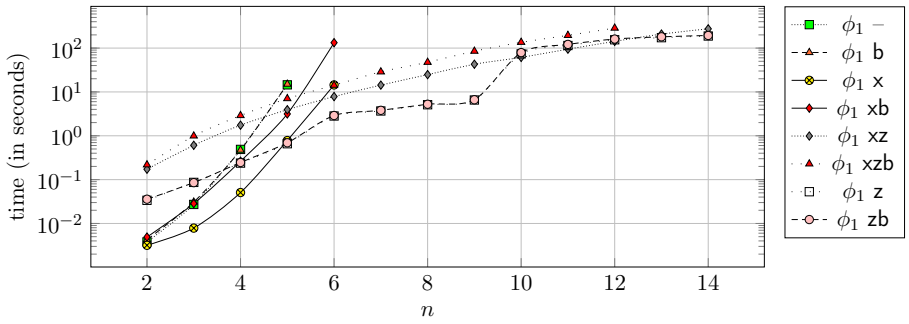

Figure 4.1: Verification results for TGC and ϕ_1: execution time

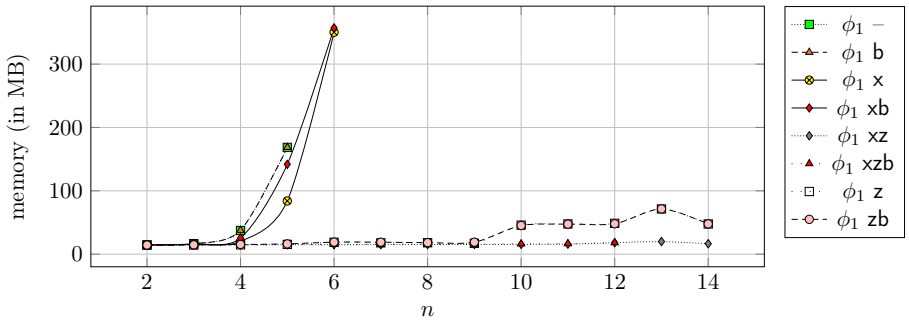

Figure 4.2: Verification results for TGC and ϕ_1: memory consumption

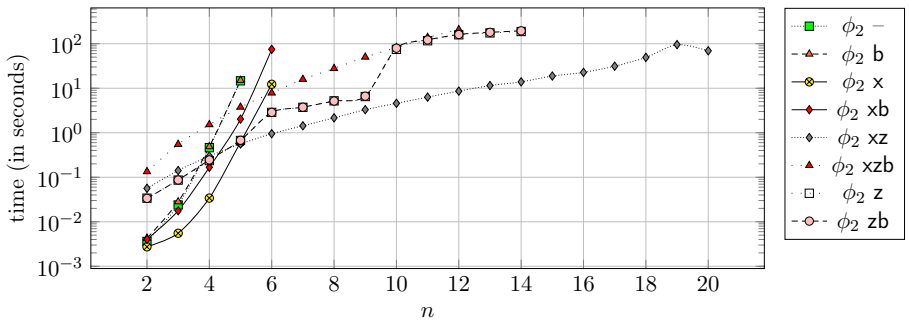

Figure 4.3: Verification results for TGC and ϕ_2: execution time

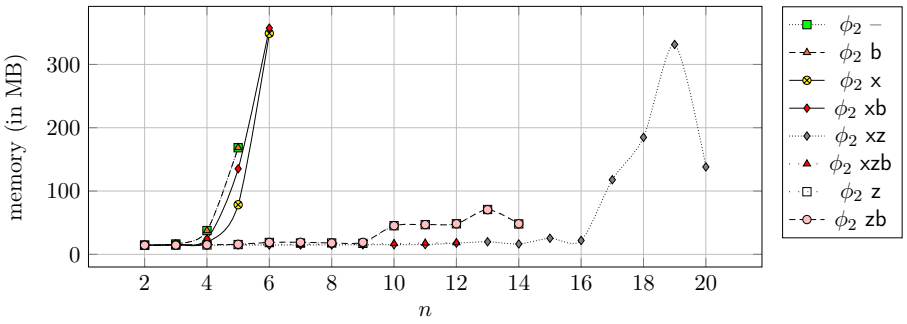

Figure 4.4: Verification results for TGC and ϕ_2: memory consumption

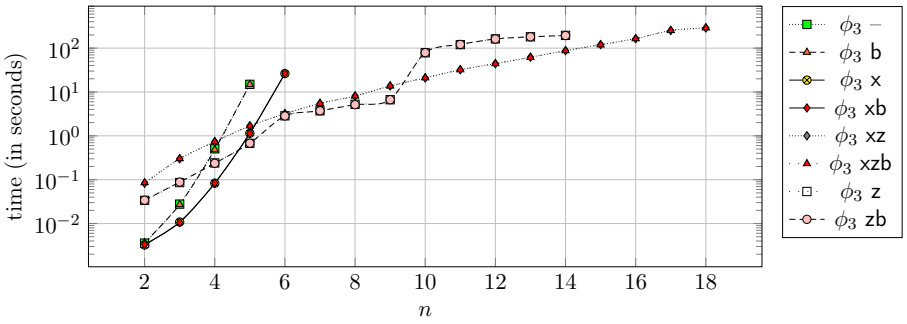

Figure 4.5: Verification results for TGC and ϕ_3: execution time

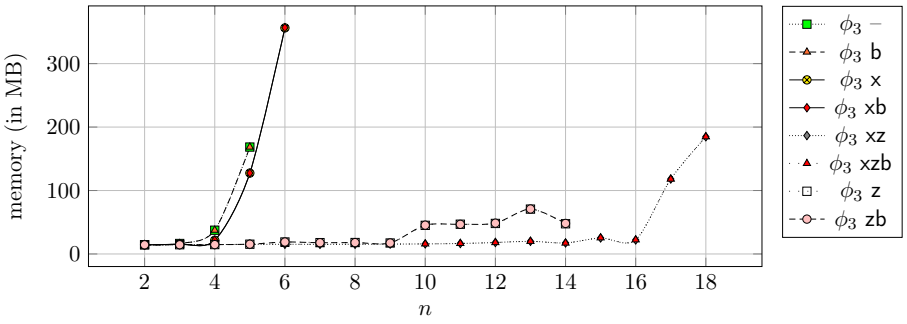

Figure 4.6: Memory consumption for TGC and ϕ_3

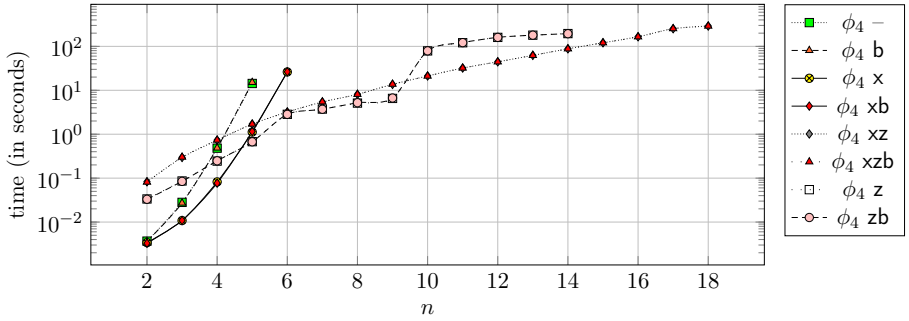

Figure 4.7: Verification results for TGC and ϕ_4: execution time

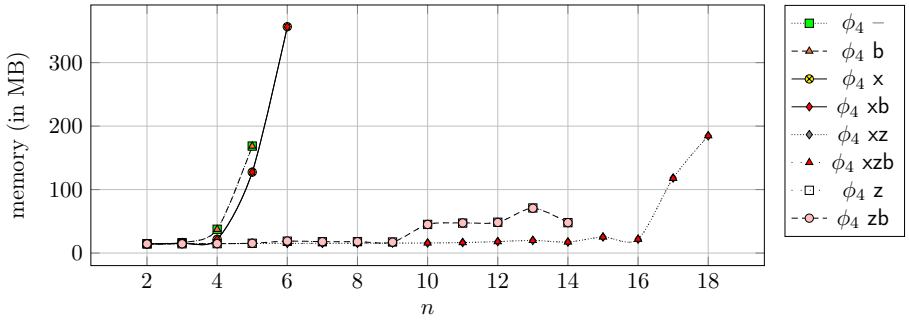

Figure 4.8: Verification results for TGC and ϕ_4: memory consumption

The background set is defined as $S = \{a, b, c, d, y, dy, r, \star\}$. 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 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:

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

The receiver agent has only one reaction: $A_n = \{(\{r\}, \{\star\}, \{r\})\}$. 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-$\mathcal{D}_{dap} = ((S, \{A_i\}_{i\in\mathcal{A}}), prg(\mathfrak{E}_{dap}))$. 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. 4.9–4.12. The execution time limit was set to 10 minutes.

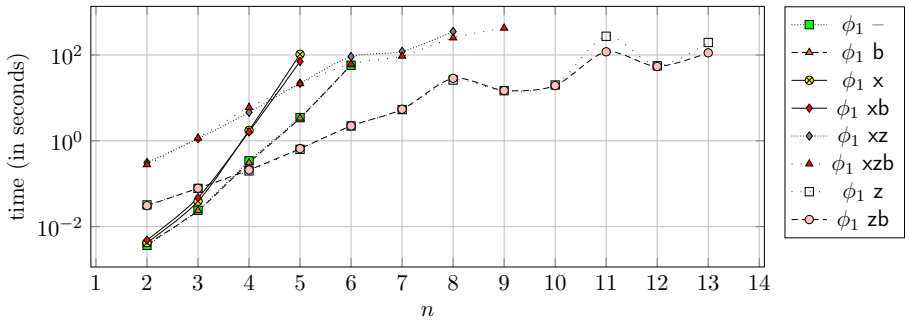

Figure 4.9: Verification results for DAP and ϕ_1: execution time

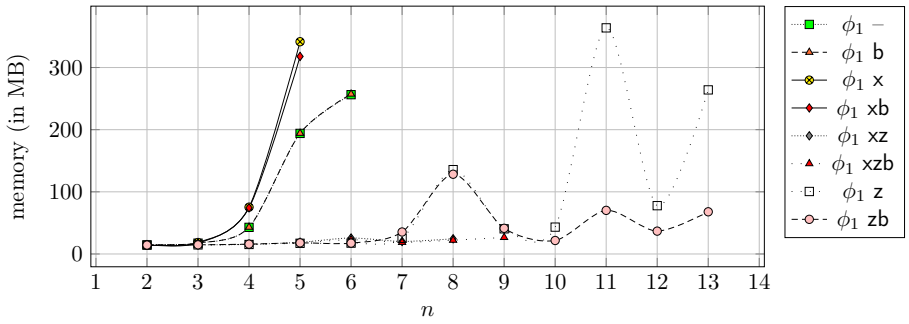

Figure 4.10: Verification results for DAP and ϕ_1: memory consumption

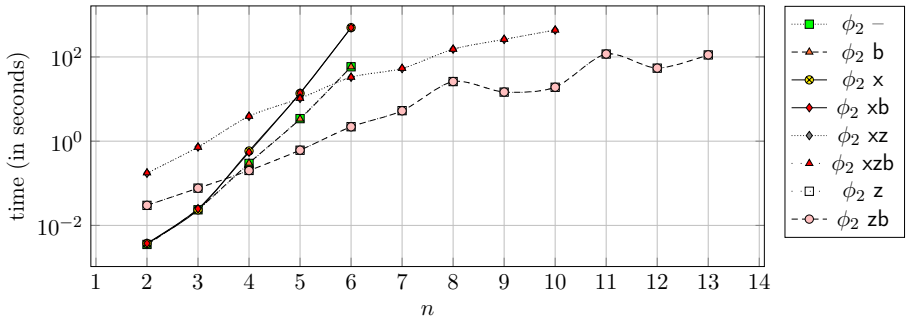

Figure 4.11: Verification results for DAP and ϕ_2: execution time

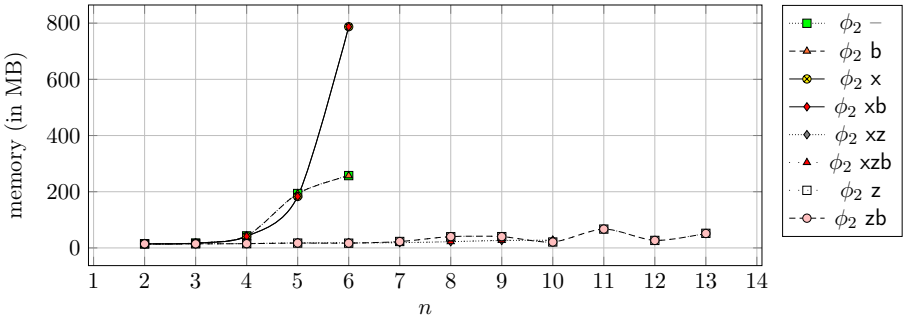

Figure 4.12: Verification results for DAP and ϕ_2: memory consumption

4.7.3 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. 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 but for its witness to be obtained the entire model needs to be explored ($n.r$ is produced in the last step of the generation of the reachable state-space). 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, reordering of BDD variables is the most efficient and using partitioning of the transition relation results in longer execution times. Using BDD reordering heuristic may result in a performance, which could be difficult to predict, e.g., in DAP and ϕ_1 for 11 agents the performance is inferior than for 12 agents. We attribute this behaviour to the implementation of the reordering heuristic in CUDD.

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 tables with the results in the appendix to this chapter.

4.8 Summary

We introduced a generalisation of reaction systems that allows for modelling distributed systems. We also extended the notion of context automata by allowing the behaviour of the environment also depend on the state of the reaction system. To allow for expressing temporal and epistemic properties of multi-agent systems,

we defined rsCTLK, which is a logic combining rsCTL with CTLK. 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.

Appendix

	-		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
15	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 4.2: Verification results for TGC and ϕ_1

	-		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 4.3: Verification results for TGC and ϕ_2

	-		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	-	-	-	-
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 4.4: Verification results 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	14.47	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	-	-	-	-
19	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 4.5: Verification results for TGC and ϕ_4

	-		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
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 4.6: Verification results for DAP and ϕ_1

	-		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
14	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 4.7: Verification results for DAP and ϕ_2

Chapter 5

Model checking for rsLTL

In this chapter we define reaction systems with discrete concentrations and a variant of linear-temporal logic. For the introduced formalisms we provide a bounded model checking method.

5.1 Reaction systems with discrete concentrations

The enabling of some of biochemical reactions encountered in practical applications depends not only on the availability of the necessary reactants and the absence of inhibitors, but also on their concentration levels. To address this aspect in biochemical modelling, we introduce an extension of the basic reaction systems supporting an explicit representation of the discrete concentration levels of entities. The resulting model uses multisets of entities, but otherwise it retains key features of the original framework. The main new idea is that the k^{th} level of concentration of an entity x is represented by a multiset containing k copies of x.

In what follows, a *multiset* over a set X is any mapping $\mathbf{b} : X \to \{0, 1, \dots\}$, and the *empty* multiset $\varnothing_X$ is one which always returns 0. For $\varnothing_X$ we simply write $\varnothing$, when X is clear from the context. If $\mathbf{b}$ is a multiset over X, we write $\mathbf{b} \in \mathcal{B}(X)$, where $\mathcal{B}(X)$ is the set of all multisets over X. For a finite set $\mathbf{B}$ of multisets over X, $\uplus(\mathbf{B})$ is the union of multisets, i.e., is the multiset over X such that $\uplus(\mathbf{B})(x) = \max(\{0\} \cup \{\mathbf{b}(x) \mid \mathbf{b} \in \mathbf{B}\})$, for every $x \in X$. For two multisets, $\mathbf{b}$ and $\mathbf{b}'$, we denote $\mathbf{b} \leq \mathbf{b}'$ if $\mathbf{b}(x) \leq \mathbf{b}'(x)$, for every $x \in X$. The *carrier* of a multiset $\mathbf{b}$ is the set $carr(\mathbf{b}) = \{x \in X \mid \mathbf{b}(x) > 0\}$.

Definition 5.1.1. A *reaction system with discrete concentrations* (RSC) is a pair $\mathcal{C} = (S, A)$, where:

- S is a finite *background* set
- and A is a nonempty finite set of *c-reactions* over the background set.

Each *c-reaction* in A is a triple $a = (\mathbf{r}, \mathbf{i}, \mathbf{p})$ such that $\mathbf{r}$, $\mathbf{i}$, $\mathbf{p}$ are multisets over S with $\mathbf{r}(e) < \mathbf{i}(e)$, for every $e \in carr(\mathbf{i})$.

The multisets $\mathbf{r}$, $\mathbf{i}$, and $\mathbf{p}$ are respectively denoted by $\mathbf{r}_a$, $\mathbf{i}_a$, and $\mathbf{p}_a$ and called the *reactant*, *inhibitor*, and *product concentration levels* of c-reaction a. An entity e is an inhibitor of a whenever $e \in carr(\mathbf{i}_a)$.

A c-reaction $a \in A$ is *enabled* by $\mathbf{t} \in \mathcal{B}(S)$, denoted $en_a(\mathbf{t})$, if $\mathbf{r}_a \leq \mathbf{t}$ and $\mathbf{t}(e) < \mathbf{i}_a(e)$, for every $e \in carr(\mathbf{i}_a)$. The *result* of a on $\mathbf{t}$ is given by $res_a(\mathbf{t}) = \mathbf{p}_a$ if $en_a(\mathbf{t})$, and by $res_a(\mathbf{t}) = \varnothing_S$ otherwise. Then the *result* of A on $\mathbf{t}$ is:

$$res_A(\mathbf{t}) = \biguplus\{res_a(\mathbf{t}) \mid a \in A\} = \biguplus\{\mathbf{p}_a \mid a \in A \text{ and } en_a(\mathbf{t})\}.$$

In the above, $\mathbf{t}$ is a *state* of a biochemical system being modelled such that, for each entity $e \in S$, $\mathbf{t}(e)$ is the *concentration level* of e (e.g., $\mathbf{t}(e) = 0$ indicates that e is not present in the current state, and $\mathbf{t}(e) = 1$ indicates that e is present at its lowest concentration level). A c-reaction a is enabled by $\mathbf{t}$ and can take place if the current concentration levels of all its reactants are at least as high as those specified by $\mathbf{r}_a$, and the current concentration levels of all its inhibitors (i.e., entities in the carrier of $\mathbf{i}_a$) are below the thresholds specified by $\mathbf{i}_a$.

Definition 5.1.2. A *context restricted reaction system with discrete concentrations* (CRRSC) is a pair CR-$\mathcal{C} = (\mathcal{C}, \mathfrak{A})$ such that $\mathcal{C} = (S, A)$ is a reaction system with discrete concentrations, and $\mathfrak{A} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ is a *context automaton* over $\mathcal{B}(S)$.

The dynamic behaviour of CR-$\mathcal{C}$ is then captured by the state sequences of its interactive processes.

Definition 5.1.3. An *interactive process* in CR-$\mathcal{C}$ is $\pi = (\zeta, \gamma, \delta)$, where:

- $\zeta = (\mathfrak{q}_0, \mathfrak{q}_1, \ldots, \mathfrak{q}_n)$, $\gamma = (\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_n)$, and $\delta = (\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_n)$
- $\mathfrak{q}_0, \mathfrak{q}_1, \ldots, \mathfrak{q}_n \in \mathcal{Q}$ with $\mathfrak{q}_0 = \mathfrak{q}^{init}$
- $\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_n, \mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_n \in \mathcal{B}(S)$ with $\mathbf{d}_0 = \varnothing_{\mathcal{B}(S)}$
- $\mathfrak{q}_i, \mathbf{c}_i, \mathfrak{q}_{i+1}) \in R$, for every $i \in \{0, \ldots, n-1\}$
- $\mathbf{d}_i = res_A(\biguplus\{\mathbf{d}_{i-1}, \mathbf{c}_{i-1}\})$, for every $i \in \{1, \ldots, n\}$.

Then the *state sequence* of π is $\tau = (\mathbf{w}_0, \ldots, \mathbf{w}_n) = (\biguplus\{\mathbf{c}_0, \mathbf{d}_0\}, \ldots, \biguplus\{\mathbf{c}_n, \mathbf{d}_n\})$.

An intuitive representation of an interactive process in CRRSC is depicted in Figure 5.1.

A context restricted reaction system with discrete concentrations CR-$\mathcal{C} = (\mathcal{C}, \mathfrak{A})$ is a finite state system since it comprises finitely many c-reactions and finitely many multisets labelling the arcs of its context automaton. More precisely, let $\#_{\text{CR-}\mathcal{C}}(e)$ be the maximum integer assigned to $e \in S$ in all the multisets of entities occurring in both $\mathcal{C}$ and $\mathfrak{A}$. Then, $\mathbf{w}(e) \leq \#_{\text{CR-}\mathcal{C}}(e)$, for all $e \in S$ and all states occurring in the state sequences of the interactive processes in CR-$\mathcal{C}$. (Note that this bound

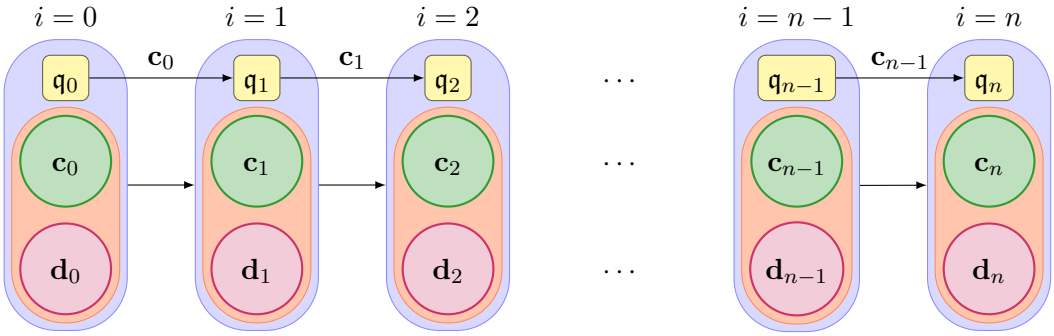

Figure 5.1: Interactive process in CRRSC

can be improved by ignoring the reactant and inhibitor multisets in c-reactions.) Moreover, the behaviour of CR-$\mathcal{C}$ can be simulated by a suitable context restricted reaction system.

To construct such a system, for every $\mathbf{t} \in \mathcal{B}(S)$, we define two sets of entities:

$$\Gamma(\mathbf{t}) = \{e.i \mid e \in S \wedge \mathbf{t}(e) = i > 0\}$$

and

$$\Gamma_{all}(\mathbf{t}) = \{e.i \mid e \in S \wedge 1 \leq i \leq \mathbf{t}(e)\}.$$

The $e.i$'s will be entities of the system we are going to construct. Note that $\Gamma_{all}(\mathbf{t})$ is a *downward-closed* set in the sense that if $e.i \in \Gamma_{all}(\mathbf{t})$ and $i > 1$, then $e.1, \ldots, e.(\text{i-1}) \in \Gamma_{all}(\mathbf{t})$. In fact, Γ_{all} is a bijection from $\mathcal{B}(S)$ to all the downward-closed sets, and its inverse Γ_{all}^{-1} is given by $\Gamma_{all}^{-1}(Z)(e) = \max\{\{0\} \cup \{i \mid e.i \in Z\}\}$, for every $e \in S$. In what follows, Γ_{all} and Γ_{all}^{-1} will be applied component-wise to sequences of respectively multisets and downward-closed sets. For such CR-$\mathcal{C}$, we define the corresponding context restricted reaction system as $\Theta(\text{CR-}\mathcal{C}) = (\mathcal{R}, \mathfrak{A}) = ((S', A'), (\mathcal{Q}, \mathfrak{q}^{init}, R'))$, where:

- $S' = \{e.i \mid e \in S \text{ and } 1 \leq i \leq \#_{\text{CR-}\mathcal{C}}(e)\}$,
- $A' = \{(\Gamma(\mathbf{r}), \Gamma(\mathbf{i}), \Gamma_{all}(\mathbf{p})) \mid (\mathbf{r}, \mathbf{i}, \mathbf{p}) \in A\}$, and
- $R' = \{(z, \Gamma_{all}(\mathbf{c}), z') \mid (z, \mathbf{c}, z') \in R\}$.

It is straightforward to see that $\Theta(\text{CR-}\mathcal{C})$ is well-defined.

As to the complexity of the translation, the number of reactions, states and arrows remains the same. Moreover, the representations of reactions and inhibitors are of the same order. What changes is the size of the background set, in the worst case by the factor $\max\{\#_{\text{CR-}\mathcal{C}}(e) \mid e \in S\}$ as well as the representations of products and contexts (again by the same factor).

We will now investigate a very close correspondence between $\Theta(\text{CR-}\mathcal{C})$ and CR-$\mathcal{C}$. First, we observe that, by the definitions of A' and R', all sets of entities occurring in the interactive processes of $\Theta(\text{CR-}\mathcal{C})$ are downward-closed. Then we obtain that all interactive processes of CR-$\mathcal{C}$ can be simulated by $\Theta(\text{CR-}\mathcal{C})$.

Theorem 5.1.4. *If $\pi = (\zeta, \gamma, \delta)$ is an interactive process in CR-$\mathcal{C}$, then $\pi' = (\zeta, \Gamma_{all}(\gamma), \Gamma_{all}(\delta))$ is an interactive process in Θ(CR-$\mathcal{C}$).*

Proof. It suffices to show for $\mathbf{w}$ in the state sequence of π, $\Gamma_{all}(res_A(\mathbf{w})) = res_{A'}(\Gamma_{all}(\mathbf{w}))$. Suppose $a = (\mathbf{r}, \mathbf{i}, \mathbf{p}) \in A$ and $a' = (\Gamma(\mathbf{r}), \Gamma(\mathbf{i}), \Gamma_{all}(\mathbf{p})) \in A'$. We first observe that a is enabled in $\mathbf{w}$ (i.e., $\mathbf{r} \leq \mathbf{w}$ and $\mathbf{w}(e) < \mathbf{i}(e)$, for all $e \in carr(\mathbf{i})$) iff a' is enabled in $\Gamma_{all}(\mathbf{w})$ (i.e., $\Gamma(\mathbf{r}) \subseteq \Gamma_{all}(\mathbf{w})$ and $\Gamma(\mathbf{i}) \cap \Gamma_{all}(\mathbf{w}) = \varnothing$). Moreover, it is easy to check that $\Gamma_{all}(res_a(\mathbf{w})) = res_{a'}(\Gamma_{all}(\mathbf{w}))$. □

Moreover, all interactive processes of Θ(CR-$\mathcal{C}$) simulate those of CR-$\mathcal{C}$.

Theorem 5.1.5. *If $\pi = (\zeta, \gamma, \delta)$ is an interactive process in Θ(CR-$\mathcal{C}$), then*

$$\pi' = (\zeta, \Gamma_{all}^{-1}(\gamma), \Gamma_{all}^{-1}(\delta))$$

is an interactive process in CR-$\mathcal{C}$.

The proof of Theorem 5.1.5 is similar to the proof of Theorem 5.1.4. We have therefore obtained a one-to-one correspondence between the interactive processes of Θ(CR-$\mathcal{C}$) and CR-$\mathcal{C}$.

Remark 5.1.6. From the point of view of enabling c-reactions, not all concentration levels are important and, consequently, they do not need to be represented in the states of Θ(CR-$\mathcal{C}$). To achieve the desired effect, all one needs to do is redefine Γ_{all}, in the following way:

$$\Gamma'_{all}(\mathbf{t}) = \Gamma(\mathbf{t}) \cup (\Gamma_{all}(\mathbf{t}) \cap \bigcup_{a \in A} \Gamma(\mathbf{r}_a) \cup \Gamma(\mathbf{i}_a)).$$

Note that syntactically CRRS are a subclass of CRRSC, such that all the concentration levels in CRRSC are limited to the value of at most one, that is, for any $\mathbf{t} \in \mathcal{B}(S)$ and for any $e \in carr(\mathbf{t})$ we have $\mathbf{t}(e) = 1$.

When dealing with concentration levels we often need to perform incrementation and decrementation operations. For this we need an additional notation: in the remainder of this dissertation we use the notation $e \mapsto i$ to indicate the multiplicity of an entity e in a multiset of entities, e.g., $\{x \mapsto 1, y \mapsto 2\}$ is a multiset with one copy of x, two copies of y, and nothing else. If the multiplicity of an entity is 1, we may also simply omit the value, e.g., $\{x, y \mapsto 2\}$.

5.2 Linear-time temporal logic for reaction systems

In this section we demonstrate how linear-time temporal logic can be used to express properties of reaction systems. Firstly, for the convenience of specifying multisets over a given set S we introduce the following grammar of *multiset expressions*:

$$\mathfrak{a} ::= true \mid e \sim c \mid e \sim e \mid \neg\mathfrak{a} \mid \mathfrak{a} \vee \mathfrak{a},$$

where $\sim \in \{<, \leq, =, \geq, >\}$, $e \in S$, and $c \in \mathbb{N}$. The set of all the multiset expressions over S is denoted by $BE(S)$. Let $\mathbf{b}$ be a multiset over S. The fact that $\mathfrak{a}$ holds in $\mathbf{b}$ is denoted by $\mathbf{b} \models_b \mathfrak{a}$, where the relation $\models_b$ is defined recursively as follows:

$$
\begin{array}{lll}
\mathbf{b} \models_b \mathit{true} & \text{for any } \mathbf{b} \in \mathcal{B}(S), & \\
\mathbf{b} \models_b e_1 \sim c & \text{iff} & \mathbf{b}(e_1) \sim c, \\
\mathbf{b} \models_b e_1 \sim e_2 & \text{iff} & \mathbf{b}(e_1) \sim \mathbf{b}(e_2), \\
\mathbf{b} \models_b \neg \mathfrak{a} & \text{iff} & \mathbf{b} \not\models_b \mathfrak{a}, \\
\mathbf{b} \models_b \mathfrak{a}_1 \vee \mathfrak{a}_2 & \text{iff} & \mathbf{b} \models_b \mathfrak{a}_1 \text{ or } \mathbf{b} \models_b \mathfrak{a}_2.
\end{array}
$$

Next, we derive the conjunction operator: $\mathfrak{a}_1 \wedge \mathfrak{a}_2 \stackrel{def}{\equiv} \neg(\neg\mathfrak{a}_1 \vee \neg\mathfrak{a}_2)$. Notice that for $\sim$ the entire set of relations is not required since we can use the logical operators to obtain the same expressiveness with a minimal set of those operators.

The language of *reaction systems linear-time temporal logic* (rsLTL, for short) is defined by the following grammar:

$$\phi ::= \mathfrak{a} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \mathsf{X}_{\mathfrak{a}}\phi \mid \phi \mathsf{U}_{\mathfrak{a}}\phi \mid \phi \mathsf{R}_{\mathfrak{a}}\phi,$$

where $\mathfrak{a} \in BE(S)$.

The logic captures requirements imposed on paths in the model of a CRRSC. Intuitively, $\mathsf{X}_{\mathfrak{a}}\phi$ means 'following an action satisfying $\mathfrak{a}$, ϕ holds in the next state', $\phi_1 \mathsf{U}_{\mathfrak{a}} \phi_2$ means 'ϕ_2 holds eventually, and ϕ_1 must hold at every preceding state, following only actions satisfying $\mathfrak{a}$', and $\phi_1 \mathsf{R}_{\mathfrak{a}} \phi_2$ means 'following only actions satisfying $\mathfrak{a}$, ϕ_2 holds up to and including the first state where ϕ_1 holds'.

For a given CR-$\mathcal{C}$ we define its model, which is then used to formally define the semantics of the introduced operators.

Definition 5.2.1. Let CR-$\mathcal{C} = (\mathcal{C}, \mathfrak{A})$, where $\mathcal{C} = (S, A)$ is a reaction system with discrete concentrations, and $\mathfrak{A} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ is a *context automaton* over $\mathcal{B}(S)$. Then, the *model* for CR-$\mathcal{C}$ is a triple $\mathcal{M} = (\mathbb{W}, \mathfrak{w}_0, \longrightarrow)$, where:

1. $\mathbb{W} = \mathcal{B}(S) \times \mathcal{Q}$ is the set of states,
2. $\mathfrak{w}^{init} = (\varnothing, \mathfrak{q}^{init})$ is the initial state,
3. $\longrightarrow \subseteq \mathbb{W} \times \mathcal{B}(S) \times \mathbb{W}$ is the transition relation such that for all $\mathbf{w}, \mathbf{w}', \alpha \in \mathcal{B}(S)$, $\mathfrak{q}, \mathfrak{q}' \in \mathcal{Q}$: $\big((\mathbf{w}, \mathfrak{q}), \alpha, (\mathbf{w}', \mathfrak{q}')\big) \in \longrightarrow$ iff: $(\mathfrak{q}, \alpha, \mathfrak{q}') \in R$ and $\mathbf{w}' = res_A(\mathbb{M}\{\mathbf{w}, \alpha\})$. Each element $(\mathfrak{w}, \alpha, \mathfrak{w}') \in \longrightarrow$ is denoted $\mathfrak{w} \xrightarrow{\alpha} \mathfrak{w}'$.

The paths in rsLTL are defined as sequences of states interleaved with actions, i.e., the context multisets.

Definition 5.2.2. A *path* is an infinite sequence $\sigma = (\mathfrak{w}_0, \alpha_0, \mathfrak{w}_1, \alpha_1, \dots)$ of states and actions such that: $\mathfrak{w}_i \xrightarrow{\alpha_i} \mathfrak{w}_{i+1}$ and $\alpha_i \in \mathcal{B}(S)$ for $i \geq 0$.

Let σ be a path. For each $i \geq 0$, the i^{th} state $\mathfrak{w}_i$ of the path σ is denoted by $\sigma_s(i)$, and the i^{th} action α_i of the path σ is denoted by $\sigma_a(i)$. Let $\sigma_s(i) = (\mathbf{w}_i, q_i)$ for each $i \geq 0$. Then, with $\sigma_b(i)$ and $\sigma_{ca}(i)$ we denote $\mathbf{w}_i$ and q_i, respectively. Let $i \geq 0$, then by σ^i we denote the suffix of σ such that $\sigma^i = (\sigma_s(i), \sigma_a(i), \sigma_s(i+1), \sigma_a(i+1), \dots)$, i.e., $\sigma^i_s(j) = \sigma_s(j+i)$ and $\sigma^i_a(j) = \sigma_a(j+i)$ for each $j \geq 0$. By $\Pi_{\mathcal{M}}$ we denote the set of all the paths of the model $\mathcal{M}$, whereas by $\Pi_{\mathcal{M}}(\mathfrak{w})$ we denote the set of all the paths that start in $\mathfrak{w} \in \mathbb{W}$, that is, $\Pi_{\mathcal{M}}(\mathfrak{w}) = \{\sigma \in \Pi \mid \sigma_s(0) = \mathfrak{w}\}$.

Definition 5.2.3. Let $\mathcal{M} = (\mathbb{W}, \mathtt{w}^{init}, \longrightarrow)$ be a model and $\sigma \in \Pi_{\mathcal{M}}$ be a path of $\mathcal{M}$. The fact that ϕ holds over σ is denoted by $\mathcal{M}, \sigma \models \phi$ (or $\sigma \models \phi$ if $\mathcal{M}$ is implicitly understood), where the relation $\models$ is defined recursively as follows:

$$
\begin{array}{lll}
\sigma \models \mathfrak{a} & \text{iff} & \sigma_b(0) \models_b \mathfrak{a}, \\
\sigma \models \phi_1 \vee \phi_2 & \text{iff} & \sigma \models \phi_1 \text{ or } \sigma \models \phi_2, \\
\sigma \models \phi_1 \wedge \phi_2 & \text{iff} & \sigma \models \phi_1 \text{ and } \sigma \models \phi_2, \\
\sigma \models \mathsf{X}_{\mathfrak{a}} \phi_1 & \text{iff} & \sigma_a(0) \models_b \mathfrak{a} \text{ and } \sigma^1 \models \phi_1, \\
\sigma \models \phi_1 \mathsf{U}_{\mathfrak{a}} \phi_2 & \text{iff} & (\exists j \geq 0)\big(\sigma^j \models \phi_2 \\
 & & \text{and } (\forall 0 \leq l < j)(\sigma^l \models \phi_1 \text{ and } \sigma_a(l) \models_b \mathfrak{a})\big), \\
\sigma \models \phi_1 \mathsf{R}_{\mathfrak{a}} \phi_2 & \text{iff} & (\forall j \geq 0)\big((\sigma^j \models \phi_2 \text{ and } (\forall 0 \leq l < j)(\sigma_a(l) \models_b \mathfrak{a})) \\
 & & \text{or } (\exists 0 \leq l < j)(\sigma^l \models \phi_1)\big).
\end{array}
$$

Next, we derive the following operators: $\mathfrak{a} \Rightarrow \phi \stackrel{def}{\equiv} \neg\mathfrak{a} \vee \phi$, $\mathsf{G}_{\mathfrak{a}}\phi \stackrel{def}{\equiv} \mathit{false}\mathsf{R}_{\mathfrak{a}}\phi$, $\mathsf{F}_{\mathfrak{a}}\phi \stackrel{def}{\equiv} \mathit{true}\mathsf{U}_{\mathfrak{a}}\phi$. Moreover, we assume $\mathfrak{a} = \mathit{true}$ when $\mathfrak{a}$ is unspecified for any of the rsLTL operators, e.g., $\mathsf{F}\phi$ is the same as $\mathsf{F}_{\mathit{true}}\phi$. The fragment of rsLTL where $\mathfrak{a} = \mathit{true}$ for all the multiset expressions $\mathfrak{a}$ is called LTL. Moreover, if the exact concentration levels are irrelevant, we may simply write e for $e > 0$ and $\neg e$ for $e = 0$.

An rsLTL formula holds in a model iff it holds in all the paths starting in its initial state, i.e., $\mathcal{M} \models \phi$ iff $\sigma \models \phi$ for all $\sigma \in \Pi_{\mathcal{M}}(\mathtt{w}^{init})$. Additionally, a formula may also hold existentially in a model, i.e., $\mathcal{M} \models_{\exists} \phi$ iff there exists $\sigma \in \Pi_{\mathcal{M}}(\mathtt{w}^{init})$ s.t. $\sigma \models \phi$.

Given CR-$\mathcal{C}$ and an rsLTL formula ϕ, rsLTL model checking is the problem of deciding whether $\mathcal{M} \models \phi$, where $\mathcal{M}$ is the model for CR-$\mathcal{C}$. The existential rsLTL model checking problem is the problem of deciding whether $\mathcal{M} \models_{\exists} \phi$.

Example 5.2.4. We consider an abstract RSC $\mathcal{C} = (\{x, y, h, m\}, \{a_1, a_2, \ldots, a_6\})$, where:

- $a_1 = (\{y \mapsto 1, x \mapsto 1\}, \{y \mapsto 2, h \mapsto 1\}, \{y \mapsto 2\})$,
- $a_2 = (\{y \mapsto 2, x \mapsto 2\}, \{y \mapsto 3, h \mapsto 1\}, \{y \mapsto 3\})$,
- $a_3 = (\{y \mapsto 3, h \mapsto 1\}, \{y \mapsto 4, h \mapsto 2\}, \{y \mapsto 4\})$,
- $a_4 = (\{y \mapsto 4, h \mapsto 1\}, \{h \mapsto 2\}, \{y \mapsto 3\})$,
- $a_5 = (\{y \mapsto 4, x \mapsto 2\}, \{h \mapsto 1\}, \{y \mapsto 2\})$,
- $a_6 = (\{m \mapsto 1\}, \{y \mapsto 3\}, \{m \mapsto 1\})$.

We define a context automaton $\mathfrak{A} = (\{\mathfrak{q}_0, \mathfrak{q}_1\}, \mathfrak{q}_0, \{r_1, r_2, \ldots, r_7\})$, where:

- $r_1 = \mathfrak{q}_0 \xrightarrow{\{x \mapsto 1, y \mapsto 1, m \mapsto 1\}} \mathfrak{q}_1$,
- $r_2 = \mathfrak{q}_1 \xrightarrow{\{x \mapsto 1\}} \mathfrak{q}_1$,
- $r_3 = \mathfrak{q}_1 \xrightarrow{\{x \mapsto 2\}} \mathfrak{q}_1$,

– $r_4 = \mathfrak{q}_1 \xrightarrow{\{x\mapsto 1, h\mapsto 1\}} \mathfrak{q}_1$,
– $r_5 = \mathfrak{q}_1 \xrightarrow{\{x\mapsto 2, h\mapsto 1\}} \mathfrak{q}_1$,
– $r_6 = \mathfrak{q}_1 \xrightarrow{\{h\mapsto 1\}} \mathfrak{q}_1$,
– $r_7 = \mathfrak{q}_1 \xrightarrow{\{h\mapsto 2\}} \mathfrak{q}_1$.

Finally, we define CR-$\mathcal{C} = (\mathcal{C}, \mathfrak{A})$. Intuitively, the system produces the entities y and m at different concentration levels. If there is y present with the concentration level of exactly one unit, the entity x is provided, and h is not present, then the concentration level of y is increased by one unit, i.e., $y \mapsto 2$ is produced. In the next step, the concentration of y is increased further, but the concentration of x is required to be at the level of two units. The level of y increases from three to four units when the level of h is exactly one unit. Then, if h is being continuously provided at the level of exactly one unit, the concentration of y oscillates between four and three units. When y is present at the concentration level of at least four units, x is at the concentration level of two units, and h is not provided, the level of y drops to two units. Additionally, when m is provided, it is sustained at the level of one unit unless y reaches the level of three units.

Let $\mathcal{M}$ be the model for CRRSC. We formulate the following rsLTL properties interpreted in $\mathcal{M}$:

$$\phi_1 = \mathsf{G}_{x>0}((y = 2) \Rightarrow \mathsf{X}_{x>1}(y \geq 3)),$$
$$\phi_2 = \mathsf{F}_{x>0}((y = 2) \wedge \mathsf{X}_{x>1}(y < 3)),$$
$$\phi_3 = \mathsf{X}((y = 3)\mathsf{R}(m \geq 1)).$$

The formula ϕ_1 states that, globally, when $x > 0$ is supplied in the context (by the context automaton), and if $y = 2$ then in the next state $y \geq 3$, if $x > 1$ is supplied in the context. This property holds existentially in the model, i.e., $\mathcal{M} \models_\exists \phi_1$. However, it does not hold universally, i.e., $\mathcal{M} \not\models \phi_1$. Let us consider $\phi_2 \equiv \neg\phi_1$. The formula holds existentially in the model, i.e., $\mathcal{M} \models_\exists \phi_2$ and ϕ_2 expresses the property for the counterexample of ϕ_1.

The property described by ϕ_3 holds in a path, where: when $y = 3$ holds, it releases $m \geq 1$ from holding, otherwise $m \geq 1$ is required to hold. The release property is required to hold after one step, i.e., by using the next-step operator we skip the first step of the path where $\sigma_b(0) = \varnothing$. □

Next we define additional notation that will be used in complexity considerations for rsLTL.

Definition 5.2.5. Let $\mathfrak{a} \in BE(S)$. Then, $|\mathfrak{a}|$ is the size of $\mathfrak{a}$, which is defined recursively as follows:

– if $\mathfrak{a} = \mathit{true}$, then $|\mathfrak{a}| = 1$,
– if $\mathfrak{a} = e_1 \sim c$ or $\mathfrak{a} = e_1 \sim e_2$, where $e_1, e_2 \in S$ and $c \in \mathbb{N}$, then $|\mathfrak{a}| = 2$,

– if $\mathfrak{a} = \neg\mathfrak{a}'$, then $|\mathfrak{a}| = |\mathfrak{a}'| + 1$,
– if $\mathfrak{a} = \mathfrak{a}' \vee \mathfrak{a}''$, then $|\mathfrak{a}| = |\mathfrak{a}'| + |\mathfrak{a}''|$.

Definition 5.2.6. Let ϕ be an rsLTL formula. Then, $\mathsf{op}(\phi)$ is the number of operators used in ϕ, which is defined recursively as follows:

– if $\phi = \mathfrak{a}$, where $\mathfrak{a} \in BE(S)$, then $\mathsf{op}(\phi) = 0$,
– if $\phi = \mathsf{X}_{\mathfrak{a}}\phi_1$, then $\mathsf{op}(\phi) = \mathsf{op}(\phi_1) + 1$,
– if $\phi \in \{\phi_1 \vee \phi_2, \phi_1 \wedge \phi_2, \phi_1\mathsf{U}_{\mathfrak{a}}\phi_2, \phi_1\mathsf{R}_{\mathfrak{a}}\phi_2\}$, then $\mathsf{op}(\phi) = \mathsf{op}(\phi_1) + \mathsf{op}(\phi_2) + 1$.

Definition 5.2.7. Let ϕ be an rsLTL formula, then $\mathsf{mbe}(\phi)$ is the size of the largest expression $\mathfrak{a} \in BE(S)$ with respect to $|\mathfrak{a}|$ used in ϕ. Then, $\mathsf{mbe}(\phi)$ is defined recursively as follows:

– if $\phi = \mathfrak{a}$, where $\mathfrak{a} \in BE(S)$, then $\mathsf{mbe}(\phi) = 0$,
– if $\phi \in \{\phi_1 \vee \phi_2, \phi_1 \wedge \phi_2\}$, then $\mathsf{mbe}(\phi) = max(\{\mathsf{mbe}(\phi_1), \mathsf{mbe}(\phi_2)\})$,
– if $\phi = \mathsf{X}_{\mathfrak{a}}\phi_1$, then $\mathsf{mbe}(\phi) = max(\{|\mathfrak{a}|, \mathsf{mbe}(\phi_1)\})$,
– if $\phi \in \{\phi_1\mathsf{U}_{\mathfrak{a}}\phi_2, \phi_1\mathsf{R}_{\mathfrak{a}}\phi_2\}$, then $\mathsf{mbe}(\phi) = max(\{|\mathfrak{a}|, \mathsf{mbe}(\phi_1), \mathsf{mbe}(\phi_2)\})$.

5.2.1 rsLTL as LTL

Here we demonstrate how the model checking problem for rsLTL can be translated into LTL model checking. The verification method presented in Section 5.3 encodes CRRSC and rsLTL directly and does not use this translation.

Let ϕ be an rsLTL formula and CR-$\mathcal{C} = (\mathcal{C}, \mathfrak{A})$ where $\mathfrak{A} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ and the set of transitions of $\mathfrak{A}$ is defined as $R = \{t_1, \ldots, t_m\}$. The aim of the translation is to define CR-$\mathcal{C}'$ and an LTL formula ϕ' such that $\mathcal{M}$ and $\mathcal{M}'$ are the models for CR-$\mathcal{C}$ and CR-$\mathcal{C}'$, respectively, and $\mathcal{M} \models \phi$ iff $\mathcal{M}' \models \phi'$.

Intuitively, the translation consists of defining CRRSC that for each context entity provided by the context automaton produces the corresponding entity in the state of RSC. These entities indicate which context entities were provided immediately before the system transitioned to a given state, i.e., via which context the current state was reached. Then, the original rsLTL formula is translated into an LTL formula where all the multiset expressions restricting contexts are expressed as constraints on the states.

First, we define a set of entities corresponding to the transitions of $\mathfrak{A}$:

$$S_\star = \{\star_i \mid t_i \in R\}.$$

Next, we define the set of the entities that are used to distinguish from the ordinary entities the entities that were supplied by the context:

$$S_c = \{\tilde{e} \mid e \in S\}.$$

Alternatively, the set can be made smaller by only selecting the entities that are used in the context automaton, i.e., $S_c = \{\tilde{e} \mid (\exists t \in R)(t = (\mathfrak{q} \xrightarrow{\mathbf{c}} \mathfrak{q}') \wedge e \in carr(\mathbf{c}))\}$. Let CR-$\mathcal{C}' = (\mathcal{C}', \mathfrak{A}')$, then $\mathcal{C}' = (S', A')$ where:

$$S' = S \cup S_\star \cup S_c.$$

The set of reactions of $\mathcal{C}'$ is defined as $A' = A \cup A_c$ where the set A_c consists of the following reactions defined for each transition $t_i \in R$:

$$(\{\star_i \mapsto 1\}, \varnothing_{S'}, \{\tilde{e} \mapsto \mathbf{c}(e) \mid t_i = (\mathfrak{q} \xrightarrow{\mathbf{c}} \mathfrak{q}') \wedge e \in carr(\mathbf{c})\}).$$

The context automaton uses a modified transition relation and is defined as $\mathfrak{A}' = (\mathcal{Q}, \mathfrak{q}^{init}, R')$ where:

$$R' = \{\mathfrak{q} \xrightarrow{\mathbf{c}} \mathfrak{q}' \mid (\exists t_i \in R)(t_i = (\mathfrak{q}_i \xrightarrow{\mathbf{c}_i} \mathfrak{q}_i') \wedge \mathbf{c} = \mathbb{M}(\mathbf{c}_i, \{\star_i \mapsto 1\}))\}.$$

Finally, we define the translation of multiset expressions interpreted over context sets with $repl(\mathfrak{a})$ we denote the expression $\mathfrak{a}$ where each occurrence of $e \in S$ in $\mathfrak{a}$ is replaced with $\tilde{e}$. For an rsLTL formula ϕ we recursively define the translation $tr^{\text{LTL}}(\phi)$ such that $tr^{\text{LTL}}(\phi)$ is an LTL formula.

- if $\phi = \mathfrak{a}$ and $\mathfrak{a} \in BE(S)$, then $tr^{\text{LTL}}(\phi) = \phi$,
- if $\phi = \phi_1 \vee \phi_2$, then $tr^{\text{LTL}}(\phi) = tr^{\text{LTL}}(\phi_1) \vee tr^{\text{LTL}}(\phi_2)$,
- if $\phi = \phi_1 \wedge \phi_2$, then $tr^{\text{LTL}}(\phi) = tr^{\text{LTL}}(\phi_1) \wedge tr^{\text{LTL}}(\phi_2)$,
- if $\phi = \mathsf{X}_\mathfrak{a}\phi_1$, then $tr^{\text{LTL}}(\phi) = \mathsf{X}(repl(\mathfrak{a}) \wedge tr^{\text{LTL}}(\phi_1))$,
- if $\phi = \phi_1 \mathsf{U}_\mathfrak{a} \phi_2$, then $tr^{\text{LTL}}(\phi) = (repl(\mathfrak{a}) \wedge tr^{\text{LTL}}(\phi_1))\mathsf{U} tr^{\text{LTL}}(\phi_2)$,
- if $\phi = \phi_1 \mathsf{R}_\mathfrak{a} \phi_2$, then $tr^{\text{LTL}}(\phi) = tr^{\text{LTL}}(\phi_1)\mathsf{R}\,(repl(\mathfrak{a}) \wedge tr^{\text{LTL}}(\phi_2))$.

Construction of $S_\star$, A_c and $\mathfrak{A}'$ require $\mathcal{O}(|R|)$ steps, while the set S_c can be built in $\mathcal{O}(|S|)$ steps. The translation of the formula ϕ runs in $\mathcal{O}(\mathsf{op}(\phi) \cdot \mathsf{mbe}(\phi))$, since for each temporal operator the associated expression $\mathfrak{a} \in BE(S)$ needs to be re-written using the entities of S_c, each rsLTL operator has a constant number of arguments, and $\mathsf{mbe}(\phi)$ is the largest $\mathfrak{a}$ used in ϕ. Therefore, the translation runs in $\mathcal{O}(|R| + |S| + \mathsf{op}(\phi) \cdot \mathsf{mbe}(\phi))$.

5.2.2 Complexity analysis

In this section we show that rsLTL model checking is PSPACE-complete.

Firstly, we define the reachability problem for CRRSC. Let $n \geq 0$ be an integer. A result $\mathbf{d} \in \mathcal{B}(S)$ is *n-step reachable* in CR-$\mathcal{C}$ if there exists an interactive process $\pi = (\zeta, \gamma, \delta)$ in CR-$\mathcal{C}$ such that $\delta = (\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_n)$ and $\mathbf{d}_n = \mathbf{d}$. We say that $\mathbf{d}$ is *reachable* in CR-$\mathcal{C}$ if there is $n \geq 0$ such that $\mathbf{d}$ is n-step reachable in CR-$\mathcal{C}$.

Lemma 5.2.8. *The reachability problem for CRRSC is PSPACE-hard.*

Proof. The proof is by reduction of a PSPACE-complete problem to the reachability problem for CRRSC. Let us take the problem of reachability of configurations of polynomial-space Turing machines, which is a PSPACE-complete problem [Papadimitriou, 1994]. The presented reduction is similar to the ones of [Formenti *et al.*, 2014a] and [Dennunzio *et al.*, 2016].

Let $TM = (Q, \Sigma, \Gamma, \delta, q_I, q_F)$ be a deterministic single-tape Turing machine, where $Q = \{q_1, \ldots, q_m\}$ is the set of states, $\Sigma = \{0, 1\}$ is the input alphabet, $\Gamma = \Sigma \cup \{\triangleright\}$ is the tape alphabet, and $q_I, q_F \in Q$ are, respectively, the initial and the accepting state of TM. The transition function is defined as $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{-1, 0, 1\}$. The input string always starts with the symbol $\triangleright$ that is never written or changed by TM, i.e., for all the transitions $\delta(q, \gamma) = (q, \gamma', d)$ we have $\gamma = \triangleright$ iff $\gamma' = \triangleright$. Moreover, if $\gamma = \triangleright$, then the tape head moves right, i.e., $d = 1$. A *configuration* of TM is a tuple $C = (q, x, pos)$, where $q \in Q$ is a state, $x \in \Sigma^*$ is the tape content and pos is the head position. The *initial configuration* of TM is defined as $C_{init} = (q_I, \gamma_1 \ldots \gamma_N, 1)$, where the current state of TM is q_I, the tape head is at position 1, and $\gamma_1 \ldots \gamma_N \in \Sigma^*$ is an input string of length N. We also assume TM is polynomially space-bounded [Baier and Katoen, 2008], i.e., there is a polynomial P such that for an input $\gamma_1 \ldots \gamma_N \in \Sigma^*$ the machine visits at most the first $P(N)$ cells of the tape. We assume $P(N) \geq N$.

The aim of this construction is to define CR-$\mathcal{C} = ((S, A), \mathfrak{A})$ that preserves the following property: a configuration C is reachable in TM from C_{init} if and only if the state of CR-$\mathcal{C}$ corresponding to the configuration C is reachable in CR-$\mathcal{C}$.

We begin by introducing the background set $S = \{e, h, w_1, \ldots, w_{P(N)}\}$. The concentration levels of e are used to encode the states of Q: we define a bijection $\mathsf{c} : Q \longrightarrow \{1, \ldots, m\}$ assigning concentration levels to the states, i.e., $\{e \mapsto \mathsf{c}(q)\}$ encodes $q \in Q$. The concentration of h denotes the position of the tape head (the concentration level values for h are taken from the set $\{0, \ldots, P(N) + 1\}$). The entities $w_1, \ldots, w_{P(N)}$ encode the symbols on the tape, i.e., tape contents.
Configurations of TM. Let $n \leq P(N)$. The configuration $C = (q, \gamma_1 \ldots \gamma_n, pos)$ is encoded in CR-$\mathcal{C}$ as follows:

$$conf_{\text{CR-}\mathcal{C}}(C) = \left(\{e \mapsto \mathsf{c}(q), h \mapsto pos\} \cup \bigcup_{j \in \{1,\ldots,n\}} \{w_j \mapsto \gamma_j\} \right). \tag{5.1}$$

Next, we define the reactions of A that aim to emulate the steps of TM in CR-$\mathcal{C}$.
Transition function. For each transition $\delta(q, \gamma) = (r, \gamma', d)$ and each tape head position $pos \in \{1, \ldots, P(N)\}$ we define the following reaction:

$$\begin{aligned}
&(\{e \mapsto \mathsf{c}(q), h \mapsto pos, w_{pos} \mapsto \gamma\},\\
&\{e \mapsto (\mathsf{c}(q) + 1), h \mapsto (pos + 1), w_{pos} \mapsto (1 - \gamma)\},\\
&\{e \mapsto c(r), h \mapsto (pos + d), w_{pos} \mapsto \gamma'\}).
\end{aligned}$$

The reactants encode the concentration levels encoding the state, the head position, and the symbol for the transition to be enabled. The inhibitors are used to enforce

exact concentration levels by not allowing concentrations higher than specified by the reactants. Finally, the products encode the successor state, the new head position, and the symbol written on the tape.
Tape contents. For all $i, pos \in \{1, \ldots, P(N)\}$ such that $i \neq pos$, we define reactions that preserve the i^{th} symbol of the tape if the tape head is at a different position *pos*.

$$(\{w_i \mapsto 1, h \mapsto pos\}, \{h \mapsto (pos + 1)\}, \{w_i \mapsto 1\}).$$

Tape boundaries. If the tape reaches $\triangleright$, then the enforced move right is encoded using the following reaction:

$$(\{e \mapsto 1\}, \{h \mapsto 1\}, \{h \mapsto 1\}).$$

The reaction is enabled in any state $q \in Q$ as there is no upper bound on the concentration level of e. We do not need to handle the remaining boundary of the tape since we assume *TM* visits only the first $P(N)$ cells of the tape; however if the head reaches the position $P(N) + 1$ the computation halts since no transitions are enabled when the tape head at the position $P(N) + 1$.
Context automaton. We define $\mathfrak{A} = (\{\mathfrak{q}_0, \mathfrak{q}_1\}, \mathfrak{q}_0, R)$ with the following transition relation:

$$R = \{\mathfrak{q}_0 \xrightarrow{C_{init}} \mathfrak{q}_1, \mathfrak{q}_1 \xrightarrow{\varnothing_S} \mathfrak{q}_1\}.$$

The role of the context automaton is to provide the encoded initial configuration as the initial context and allow for the subsequent computation steps by providing transitions with empty contexts.

The reduction runs in polynomial time since the encoding of the transition function requires $\mathcal{O}(P(N) \cdot |\delta|)$ reactions[1] and the encoding of the preservation of the tape contents requires $\mathcal{O}(P(N)^2)$ reactions. The construction of CR-$\mathcal{C}$ ensures that a configuration C is reachable in *TM* if and only if $conf_{\text{CR-}\mathcal{C}}(C)$ is reachable in CR-$\mathcal{C}$. Therefore, the reachability problem for CRRSC is PSPACE-hard. □

The reachability of $\mathbf{d} \in \mathcal{B}(S)$ can be expressed in terms of existential rsLTL model checking using the following formula:

$$\mathsf{F}\left(\bigwedge_{e \in carr(\mathbf{d})} (e = \mathbf{d}(e))\right).$$

Therefore, from Lemma 5.2.8 we also get the following result.

Corollary 5.2.9. *The existential rsLTL model checking problem is PSPACE-hard.*

Let ϕ be an rsLTL formula, CR-$\mathcal{C}$ be a CRRSC, and $\mathcal{M}$ be the model for CR-$\mathcal{C}$. The existential decision problem yields *true* if $\mathcal{M} \models_\exists \phi$, and *false* otherwise. Since $\mathcal{M} \models \phi$ if and only if $\mathcal{M} \not\models_\exists \neg\phi$, the universal decision problem yields *true* if and only if the existential variant of the problem for $\neg\phi$ yields *false*.

[1]The size of the transition function δ is denoted by $|\delta|$.

Therefore, from the PSPACE-hardness of the existential rsLTL model checking problem we get PSPACE-hardness of its universal variant. This follows from the fact that *co*PSPACE = PSPACE, i.e., the complement of any PSPACE-hard problem is also PSPACE-hard [Papadimitriou, 1994].

Corollary 5.2.10. *The rsLTL model checking problem is PSPACE-hard.*

Lemma 5.2.11. *The rsLTL model checking problem is in PSPACE.*

Proof. Since there is a polynomial translation of the rsLTL model checking problem to the LTL model checking problem, it is sufficient to show the LTL model checking problem for CRRS is in PSPACE.

The proof follows the same reasoning as the one for Lemma 5.47 in [Baier and Katoen, 2008]. It gives a nondeterministic polynomial space-bounded algorithm solving the existential LTL model checking problem. The algorithm nondeterministically guesses a path in $TS \otimes \mathcal{G}_\phi$, i.e., in the product of, respectively, a finite transition system and a generalised nondeterministic Büchi automaton for the verified LTL formula ϕ.

However, here TS is not given as the input and needs to be obtained from CRRS. In fact, it only must be possible to obtain a successor state in polynomial space and a method for that is demonstrated in the proof of Lemma 4.4.3 for the rsCTLK model checking problem. The proof also uses the same technique as the proof of Lemma 3.4.2 for rsCTL but the proof for rsCTLK additionally shows how to handle the context automaton. □

The following theorem follows directly from Corollary 5.2.10 and Lemma 5.2.11.

Theorem 5.2.12. *The rsLTL model checking problem is PSPACE-complete.*

5.2.3 Bounded semantics

Motivated by various successful applications of bounded model checking to practical problems such as software verification [Kroening and Strichman, 2016], we focus on the bounded model checking approach defined for finite prefixes of paths. This approach requires us to specify when a given formula holds while considering only a finite number of states and actions that belong to the prefix of the considered path.

Definition 5.2.13. A path $\sigma = (\mathtt{w}_0, \alpha_0, \mathtt{w}_1, \alpha_1, \dots)$ is a (k,l)-*loop* (or k-*loop*) if there exist $k \geq l > 0$ such that $\mathtt{w}_{l-1} = \mathtt{w}_k$ and $\sigma = (\mathtt{w}_0, \alpha_0, \dots, \alpha_{l-2}, \mathtt{w}_{l-1})(\alpha_l, \mathtt{w}_{l+1}, \alpha_{l+1}, \dots, \alpha_{k-1}, \mathtt{w}_k)^\omega$.

The bounded semantics for rsLTL is defined for finite path prefixes. We define a satisfiability relation that for a given path considers its first k states and $k-1$ actions only.

Definition 5.2.14. The fact that a formula ϕ holds in a path σ with bound $k \in \mathbb{N}$ is denoted by $\sigma \models^k \phi$. Then, $\sigma \models^k \phi$ if and only if:

– σ is a (k,l)-loop for some $0 < l \leq k$ and $\sigma \models \phi$, or

– $\sigma \models_{nl} \phi$, where:

$$
\begin{array}{lll}
\sigma \models_{nl} \mathfrak{a} & \text{iff} & \sigma_s(0) \models_b \mathfrak{a}, \\
\sigma \models_{nl} \phi_1 \wedge \phi_2 & \text{iff} & \sigma \models_{nl} \phi_1 \text{ and } \sigma \models_{nl} \phi_2, \\
\sigma \models_{nl} \phi_1 \vee \phi_2 & \text{iff} & \sigma \models_{nl} \phi_1 \text{ or } \sigma \models_{nl} \phi_2, \\
\sigma \models_{nl} \mathsf{X}_{\mathfrak{a}}\phi & \text{iff} & k > 0,\ \sigma_a(0) \models_b \mathfrak{a}, \text{ and } \sigma^1 \models_{nl} \phi, \\
\sigma \models_{nl} \phi_1 \mathsf{U}_{\mathfrak{a}} \phi_2 & \text{iff} & (\exists 0 \leq j \leq k)\big(\sigma^j \models_{nl} \phi_2 \\
 & & \text{and } (\forall 0 \leq l < j)(\sigma^l \models_{nl} \phi_1 \text{ and } \sigma_a(l) \models_b \mathfrak{a})\big) \\
\sigma \models_{nl} \phi_1 \mathsf{R}_{\mathfrak{a}} \phi_2 & \text{iff} & (\exists 0 \leq j \leq k)\big(\sigma^j \models_{nl} \phi_1 \text{ and } ((\forall 0 \leq l \leq j)(\sigma^l \models_{nl} \phi_2) \\
 & & \text{and } (\forall 0 \leq l < j)(\sigma_a(l) \models_b \mathfrak{a}))\big)
\end{array}
$$

Lemma 5.2.15. *Let $k \in \mathbb{N}$, ϕ be an rsLTL formula, and σ be a path. Then, $\sigma \models^k \phi$ implies $\sigma \models \phi$.*

Lemma 5.2.16. *Let ϕ be an rsLTL formula and $\mathcal{M}$ be a model. Then, $\mathcal{M} \models \phi$ implies that there exists $k \in \mathbb{N}$ such that $\mathcal{M} \models^k \phi$.*

The proofs for these lemmas are similar to the ones for LTL [Biere *et al.*, 1999b]. The only difference in these proofs is related to the augmented temporal operators that impose additional restrictions on the considered path by using multiset expressions.

For a bound $k \in \mathbb{N}$ we define the relation $\models^k_\exists$ for models as follows: $\mathcal{M} \models^k_\exists \phi$ iff there exists $\sigma \in \Pi_{\mathcal{M}}(\mathfrak{w}^{init})$ s.t. $\sigma \models^k \phi$. The *bounded model checking problem* for rsLTL is defined as the decision problem of checking if $\mathcal{M} \models^k_\exists \phi$ for a given bound $k \in \mathbb{N}$.

Based on Lemma 5.2.15 and 5.2.16 we state the following theorem:

Theorem 5.2.17. *Let ϕ be an rsLTL formula and $\mathcal{M}$ be a model. Then, $\mathcal{M} \models_\exists \phi$ iff there exists $k \in \mathbb{N}$ such that $\mathcal{M} \models^k_\exists \phi$.*

5.3 SMT-based encoding

In this section we provide a translation of the bounded model checking problem for rsLTL into the satisfiability modulo theory (SMT) [Kroening and Strichman, 2016] with the integer arithmetic theory. The SMT problem is a generalisation of the Boolean satisfiability problem, where some functions and predicate symbols have interpretations from the underlying theory.

Let CR-$\mathcal{C} = (\mathcal{R}, \mathfrak{A})$ be a CRRSC where $\mathcal{R} = (S, A)$, $\mathfrak{A} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$, and let $\mathcal{M}$ be the model for CR-$\mathcal{C}$ and ϕ an rsLTL formula. For an integer $k \geq 0$ we construct a formula $[\mathcal{M}, \phi, k]$ such that: $\mathcal{M} \models^k_\exists \phi$ iff $[\mathcal{M}, \phi, k]$ is satisfiable. We encode all the paths of the model $\mathcal{M}$ that are bounded with k. The entities of S are denoted by $e_1, \ldots, e_m$, where $m = |S|$. For each $i \in \{0, \ldots, k\}$ we introduce

the following sets of positive integer variables used in the encoding:

$$\mathbf{P}_i = \{\mathtt{p}_{i,1}, \ldots, \mathtt{p}_{i,m}\},$$
$$\mathbf{P}_i^{\mathcal{E}} = \{\mathtt{p}_{i,1}^{\mathcal{E}}, \ldots, \mathtt{p}_{i,m}^{\mathcal{E}}\}.$$

Let σ be a path of $\mathcal{M}$. Then, $\mathtt{p}_{i,1}, \ldots, \mathtt{p}_{i,m}$ and $\mathtt{p}_{i,1}^{\mathcal{E}}, \ldots, \mathtt{p}_{i,m}^{\mathcal{E}}$ encode $\sigma_b(i)$ and $\sigma_a(i)$, respectively. We also introduce the variables $\mathtt{q}_0, \ldots, \mathtt{q}_k$ which are used to encode the locations of $\mathfrak{A}$. The location $\sigma_{ca}(i)$ of the context automaton is then encoded with $\mathtt{q}_i$. Then, we define $\overline{\mathbf{p}}_i = (\mathtt{p}_{i,1}, \ldots, \mathtt{p}_{i,m})$ and $\overline{\mathbf{p}}_i^{\mathcal{E}} = (\mathtt{p}_{i,1}^{\mathcal{E}}, \ldots, \mathtt{p}_{i,m}^{\mathcal{E}})$. With $\overline{\mathbf{p}}_i[j]$ and $\overline{\mathbf{p}}_i^{\mathcal{E}}[j]$ we denote, respectively, $\mathtt{p}_{i,j}$ and $\mathtt{p}_{i,j}^{\mathcal{E}}$. Then, we also define $\mathbf{P} = \bigcup_{i=0}^{k} \mathbf{P}_i$ and $\mathbf{P}^{\mathcal{E}} = \bigcup_{i=0}^{k} \mathbf{P}_i^{\mathcal{E}}$.

We define the following functions that map background set entities to the corresponding variables of the encoding: for all $0 \leq i \leq k$ we define $\mathtt{t}_i : S \to \mathbf{P}_i$ and $\mathtt{t}_i^{\mathcal{E}} : S \to \mathbf{P}_i^{\mathcal{E}}$ such that $\mathtt{t}_i(e_j) = \mathtt{p}_{i,j}$, $\mathtt{t}_i^{\mathcal{E}}(e_j) = \mathtt{p}_{i,j}^{\mathcal{E}}$ for all $1 \leq j \leq m$. The function $\mathtt{e} : \mathcal{Q} \to \{0, \ldots, |\mathcal{Q}| - 1\}$ maps states of the context automaton to the corresponding natural values used in the encoding. The set of the reactions that are capable of producing $e \in S$ is defined as $\mathit{Prod}(e) = \{a \in A \mid \mathbf{p}_a(e) > 0\}$. Let f_1, f_2, f_3 be expressions over $\mathbf{P} \cup \mathbf{P}^{\mathcal{E}}$, then we define the *if-then-else* operator:

$$f_1 \to f_2 \mid f_3 = (f_1 \wedge f_2) \vee (\neg f_1 \wedge f_3).$$

To define the SMT encoding of the paths we need auxiliary functions that correspond to elements of the encoding.

The encoding of the reactions is defined in two steps: we define a formula encoding the condition for when a reaction is enabled and a formula encoding what that reaction produces when it is enabled.

Enabledness. The enabledness of a reaction $a \in A$ is encoded as follows:

$$\mathsf{En}_a(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}) = \bigwedge_{e \in S} \left(\mathtt{t}_i(e) \geq \mathbf{r}_a(e) \vee \mathtt{t}_i^{\mathcal{E}}(e) \geq \mathbf{r}_a(e)\right) \wedge \bigwedge_{e \in S} \left(\mathtt{t}_i(e) < \mathbf{i}_a(e) \wedge \mathtt{t}_i^{\mathcal{E}}(e) < \mathbf{i}(e)\right).$$

The formula encodes the conditions for $a \in A$ to be enabled, i.e., in the current state and in the context, the concentration levels of the reactants specified in $\mathbf{r}_a$ need to be sufficient and the concentration levels of all its inhibitors need to be below the threshold specified by $\mathbf{i}_a$.

Entity concentration. To encode the produced entity concentration level of an entity $e \in S$ we take all the reactions that have e in their products, i.e., all the reactions of $\mathit{Prod}(e)$ and order them with respect to the produced concentration levels of e. Let $a_1, a_2, \ldots, a_w \in \mathit{Prod}(e)$ and assume $\mathbf{p}_{a_j} \leq \mathbf{p}_{a_{j+1}}$ for all $1 \leq j < w$. First, we encode the produced concentration level of entity e where $j \in \{1, \ldots, w\}$

when there exist reactions producing e, i.e., $w > 0$, and at least one such reaction is enabled (the remaining cases are handled later). The encoding is recursive and is defined as follows:

$$\mathsf{C}_e^j\big(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}\big) = \begin{cases} \mathsf{En}_{a_j}\big(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}\big) \rightarrow (\mathtt{t}_{i+1}(e) = \mathbf{p}_{a_j}) \mid \mathsf{C}_e^{j-1}\big(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}\big) & \text{if } 1 < j \leq w, \\ \mathsf{En}_{a_j}\big(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}\big) \wedge (\mathtt{t}_{i+1}(e) = \mathbf{p}_{a_j}) & \text{if } j = 1. \end{cases}$$

In the definition of $\mathsf{C}_e^j\big(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}\big)$ with the use of the *if-then-else* operator we always encode the highest available concentration of e if the corresponding reaction producing e with that concentration level is enabled. Finally, we define the complete entity concentration encoding for all the reactions:

$$\mathsf{C}_e\big(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}\big) = \begin{cases} \mathtt{t}_{i+1}(e) = 0 & \text{if } w = 0, \\ \mathsf{C}_e^w(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}) \vee \left(\left(\bigwedge_{a \in Prod(e)} \neg \mathsf{En}_a(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}) \right) \wedge (\mathtt{t}_{i+1}(e) = 0) \right) & \text{if } w > 0. \end{cases}$$

In the above, we handle the remaining cases: (1) when $w = 0$, i.e., if there are no reactions producing e, and (2) there are reactions producing e or (3) none of them are enabled. In (1) and (3) the entity e is produced with the concentration level 0. In (2) the recursive encoding of $\mathsf{C}_e^w\big(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}\big)$ ensures production of e and selection of its maximal concentration level by starting the encoding from a_w, i.e., the reaction producing the highest concentration level of e.

Context. Let $\mathbf{c} \in \mathcal{B}(S)$ be a multiset of context entities. The encoding of $\mathbf{c}$ is defined as follows:

$$\mathsf{Ct}_{\mathbf{c}}(\overline{\mathbf{p}}_i^{\mathcal{E}}) = \bigwedge_{e \in S} (\mathtt{t}_i^{\mathcal{E}}(e) = \mathbf{c}(e)).$$

Transitions of context automaton. The encoding of the transition relation of the context automaton is a disjunction of the encodings for each transition:

$$\mathsf{Tr}_{\mathfrak{A}}(\mathsf{q}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \mathsf{q}_{i+1}) = \bigvee_{(\mathfrak{q}, \mathbf{c}, \mathfrak{q}') \in R} \big(\mathsf{q}_i = \mathsf{e}(\mathfrak{q}) \wedge \mathsf{Ct}_{\mathbf{c}}(\overline{\mathbf{p}}_i^{\mathcal{E}}) \wedge \mathsf{q}_{i+1} = \mathsf{e}(\mathfrak{q}')\big)\,.$$

Transition relation. We build a conjunction of the produced concentration levels for all entities and the transition relation for the context automaton to encode the transition relation of the model:

$$\mathsf{Tr}_{\text{CR-}\mathcal{C}}(\overline{\mathbf{p}}_i, \mathsf{q}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}, \mathsf{q}_{i+1}) = \left(\bigwedge_{e \in S} \mathsf{C}_e(\overline{\mathbf{p}}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}) \right) \wedge \mathsf{Tr}_{\mathfrak{A}}(\mathsf{q}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \mathsf{q}_{i+1}).$$

Initial state. To encode the initial state of the model where all the concentration levels are set to zero and the context automaton is in its initial state we define the following formula:

$$\mathsf{Init}(\overline{\mathbf{p}}_i, \mathsf{q}_i) = \bigwedge_{e \in S} (\mathsf{t}_i(e) = 0) \wedge \left(\mathsf{q}_i = \mathsf{e}(\mathfrak{q}^{init})\right).$$

Paths. To encode the paths of $\mathcal{M}$ that are bounded with k we unroll the transition relation up to k and combine it with the encoding of the initial state of the model:

$$\mathsf{Paths}^k = \mathsf{Init}(\overline{\mathbf{p}}_0, \mathsf{q}_0) \wedge \bigwedge_{i=0}^{k-1} \mathsf{Tr}_{\text{CR-}\mathcal{C}}(\overline{\mathbf{p}}_i, \mathsf{q}_i, \overline{\mathbf{p}}_i^{\mathcal{E}}, \overline{\mathbf{p}}_{i+1}, \mathsf{q}_{i+1}).$$

Formulae of rsLTL. Next, we present a translation of the rsLTL formulae into an SMT encoding. The rsLTL encoding is based on the fixed point encoding for LTL presented in [Biere *et al.*, 2006].

In rsLTL, in place of propositional variables appearing in standard LTL formulae, we use multiset expressions. Let $\mathfrak{a}$ be a multiset expression, $enc_i^b(\mathfrak{a})$ and $enc_i^{ct}(\mathfrak{a})$ denote the encoding of $\mathfrak{a}$ using the variables of, respectively, $\overline{\mathbf{p}}_i$ and $\overline{\mathbf{p}}_i^{\mathcal{E}}$. The former refers to states of RSC, while the latter refers to actions (or contexts). Since we are defining a translation into SMT, the encodings of multisets are defined in a straightforward way. To deal with (k, l)-loops we introduce an integer variable $\mathtt{L}$. When $\mathtt{L} = l$ holds for a path then the path is a (k, l)-loop:

$$\mathsf{Loops}^k = \neg(\mathtt{L} = 0) \wedge \bigwedge_{i=1}^{k} \left((\mathtt{L} = i) \Rightarrow \mathsf{E}(\overline{\mathbf{p}}_{i-1}, \mathsf{q}_{i-1}, \overline{\mathbf{p}}_k, \mathsf{q}_k)\right),$$

where E encodes the equivalence of two states of the model:

$$\mathsf{E}(\overline{\mathbf{p}}_i, \mathsf{q}_i, \overline{\mathbf{p}}_j, \mathsf{q}_j) = (\mathsf{q}_i = \mathsf{q}_j) \wedge \bigwedge_{c=1}^{m} (\overline{\mathbf{p}}_i[c] = \overline{\mathbf{p}}_j[c]).$$

The encoding of an rsLTL formula ϕ at the position $i \in \{0, \ldots, k\}$ is defined as $[\![\phi]\!]_i^k$. Firstly, we introduce the encoding for propositional formulae:

$[\![\phi]\!]_i^k$	$0 \leq i \leq k$
$[\![\mathfrak{a}]\!]_i^k$	$enc_i^b(\mathfrak{a})$
$[\![\phi_1 \wedge \phi_2]\!]_i^k$	$[\![\phi_1]\!]_i^k \wedge [\![\phi_2]\!]_i^k$
$[\![\phi_1 \vee \phi_2]\!]_i^k$	$[\![\phi_1]\!]_i^k \vee [\![\phi_2]\!]_i^k$

Next, we define the encoding for temporal formulae. The translations of the until and release operators are based on the fixed point encoding for CTL [Clarke *et al.*, 1999]. The encoding introduces an auxiliary translation $\langle\!\langle \phi \rangle\!\rangle_i^k$ that corresponds to computing fixed point approximations.

$[\![\phi]\!]_i^k$	$0 \leq i < k$
$[\![\mathsf{X}_\mathfrak{a}\phi_1]\!]_i^k$	$[\![\phi_1]\!]_{i+1}^k \wedge enc_i^{ct}(\mathfrak{a})$
$[\![\phi_1 \mathsf{U}_\mathfrak{a} \phi_2]\!]_i^k$	$[\![\phi_2]\!]_i^k \vee \big([\![\phi_1]\!]_i^k \wedge ([\![\phi_1 \mathsf{U}_\mathfrak{a} \phi_2]\!]_{i+1}^k \wedge enc_i^{ct}(\mathfrak{a}))\big)$
$[\![\phi_1 \mathsf{R}_\mathfrak{a} \phi_2]\!]_i^k$	$[\![\phi_2]\!]_i^k \wedge \big([\![\phi_1]\!]_i^k \vee ([\![\phi_1 \mathsf{R}_\mathfrak{a} \phi_2]\!]_{i+1}^k \wedge enc_i^{ct}(\mathfrak{a}))\big)$
	$i = k$
$[\![\mathsf{X}_\mathfrak{a}\phi_1]\!]_i^k$	$\bigvee_{j=1}^k ((\mathtt{L} = j) \wedge [\![\phi_1]\!]_j^k) \wedge enc_i^{ct}(\mathfrak{a})$
$[\![\phi_1 \mathsf{U}_\mathfrak{a} \phi_2]\!]_i^k$	$[\![\phi_2]\!]_i^k \vee \big([\![\phi_1]\!]_i^k \wedge (\bigvee_{j=1}^k ((\mathtt{L} = j) \wedge \langle\!\langle \phi_1 \mathsf{U}_\mathfrak{a} \phi_2 \rangle\!\rangle_{i+1}^k) \wedge enc_i^{ct}(\mathfrak{a}))\big)$
$[\![\phi_1 \mathsf{R}_\mathfrak{a} \phi_2]\!]_i^k$	$[\![\phi_2]\!]_i^k \wedge \big([\![\phi_1]\!]_i^k \vee (\bigvee_{j=1}^k ((\mathtt{L} = j) \wedge \langle\!\langle \phi_1 \mathsf{R}_\mathfrak{a} \phi_2 \rangle\!\rangle_{i+1}^k) \wedge enc_i^{ct}(\mathfrak{a}))\big)$
$\langle\!\langle \phi \rangle\!\rangle_i^k$	$0 \leq i < k$
$\langle\!\langle \phi_1 \mathsf{U}_\mathfrak{a} \phi_2 \rangle\!\rangle_i^k$	$[\![\phi_2]\!]_i^k \vee \big([\![\phi_1]\!]_i^k \wedge (\langle\!\langle \phi_1 \mathsf{U}_\mathfrak{a} \phi_2 \rangle\!\rangle_{i+1}^k \wedge enc_i^{ct}(\mathfrak{a}))\big)$
$\langle\!\langle \phi_1 \mathsf{R}_\mathfrak{a} \phi_2 \rangle\!\rangle_i^k$	$[\![\phi_2]\!]_i^k \wedge \big([\![\phi_1]\!]_i^k \vee (\langle\!\langle \phi_1 \mathsf{R}_\mathfrak{a} \phi_2 \rangle\!\rangle_{i+1}^k \wedge enc_i^{ct}(\mathfrak{a}))\big)$
	$i = k$
$\langle\!\langle \phi_1 \mathsf{U}_\mathfrak{a} \phi_2 \rangle\!\rangle_i^k$	$[\![\phi_2]\!]_i^k$
$\langle\!\langle \phi_1 \mathsf{R}_\mathfrak{a} \phi_2 \rangle\!\rangle_i^k$	$[\![\phi_2]\!]_i^k$

The cases for $[\![\phi]\!]_i^k$ when $i = k$ are considered separately: additional transitions for $j \in \{1, \ldots, k\}$ are encoded when (k, j)-loop exists, i.e., when $\mathtt{L} = j$ holds. In contrast to the LTL encoding of [Biere *et al.*, 2006], we require for all the transitions to be constrained by the parameter $\mathfrak{a}$ encoded with $enc_i^{ct}(\mathfrak{a})$.

Finally, the bounded model checking problem for rsLTL is reduced to satisfiability checking, i.e., to verify if $\mathcal{M} \models_\exists^k \phi$ we check the satisfiability of the following formula:

$$[\mathcal{M}, \phi, k] = \mathsf{Paths}^k \wedge \mathsf{Loops}^k \wedge [\![\phi]\!]_0^k.$$

Theorem 5.3.1. *Let CR-$\mathcal{C} = (\mathcal{C}, \mathfrak{A})$ be a CRRSC, $\mathcal{M}$ be its model and ϕ an rsLTL formula. For any $k \in \mathbb{N}$, the formula $[\mathcal{M}, \phi, k]$ is satisfiable iff $\mathcal{M} \models_\exists^k \phi$.*

Proof. We assume an arbitrary $k \in \mathbb{N}$. The formula $[\mathcal{M}, \phi, k]$ is satisfiable iff there exists a valuation of the variables used in the encoding such that the formula is satisfied. The valuation then represents the path prefix of a path in $\mathcal{M}$ for which the formula ϕ holds. We first show that Paths^k encodes path prefixes of paths in $\mathcal{M}$. There exists a path σ in $\mathcal{M}$ and σ^k is its prefix of length k iff there exists a valuation representing σ^k that satisfies Paths^k. Let $i \in \{0, \ldots, k-1\}$. We observe the formula $\mathsf{Tr}_{\text{CR-}\mathcal{C}}$ is satisfied for the valuation encoding $\sigma_s(i)$, $\sigma_a(i)$ and $\sigma_s(i+1)$ iff $\sigma_s(i) \xrightarrow{\sigma_a(i)} \sigma_s(i+1)$. This follows from the encoding of C_e and $\mathsf{Tr}_\mathfrak{A}$. Let us recall that $\sigma_s(i) = (\sigma_b(i), \sigma_{ca}(i))$. For an entity $e \in S$ it is clear from the construction that the formula C_e is satisfied iff the valuation encodes the concentration level of the entity e in $\sigma_b(i+1)$ that is produced by the reactions enabled in $\sigma_b(i)$ with the context $\sigma_a(i)$. The encoding C_e is applied to all $e \in S$, i.e., the valuation must encode in $\sigma_b(i+1)$ the concentration levels of all the entities of S. The formula $\mathsf{Tr}_\mathfrak{A}$ is satisfied iff the valuation encodes a transition $(\mathfrak{q}, \mathbf{c}, \mathfrak{q}') \in R$ such that $\mathfrak{q} = \sigma_{ca}(i)$, $\mathbf{c} = \sigma_a(i)$ and $\mathfrak{q}' = \sigma_{ca}(i+1)$. Then, the formula Paths^k is satisfied iff the valuation encodes a path prefix σ^k such that the state $\sigma_s(0)$ of $\mathcal{M}$ is the initial state and $\sigma_s(i) \xrightarrow{\sigma_a(i)} \sigma_s(i+1)$ for all $i \in \{0, \ldots, k-1\}$.

It remains to show that $\mathsf{Loops}^k \wedge [\![\phi]\!]_0^k$ restricts the valuation so that the formula $[\mathcal{M}, \phi, k]$ is satisfiable iff it encodes a path prefix of a path in which ϕ holds. To show this we apply the same reasoning as for LTL in [Biere *et al.*, 2006, Theorem 3.1], which follows by induction on the structure of the LTL formula. □

5.4 Experimental evaluation

In this section we present the results of an experimental evaluation of the translation presented in Section 5.3. The verification tool was implemented in Python and uses Z3 [de Moura and Bjørner, 2008] for SMT-solving. We implement an incremental approach, i.e., in a single SMT instance we increase the length of the encoded interactive processes by unrolling their encoding until a witness for the verified property is found, instead of creating separate instances for each length tested.

Additionally, we compare the implementation for CRRSC with an implementation for CRRS by verifying reachability properties of the CRRS obtained by applying the translation defined in Section 5.1 to CRRSC. To provide a fair comparison, both the verification approaches were implemented in Python using similar techniques. The implementation for CRRS is based on the encoding from Section 5.3 which is optimised for CRRS by using Boolean variables instead of integer variables. The translation into SMT for CRRS corresponds to the translation for CRRSC – it is assumed that all concentration levels are equal to 1 when an entity is present, and equal to 0 otherwise.

The reachability problem for CRRSC was defined in Section 5.2.2. However, it can also be defined for pairs of multisets that are interpreted similarly to how the reactants and inhibitors are used in the reaction enabledness condition. Then, the *k-step reachability* can be defined for a pair $\rho = (\mathbf{x}, \mathbf{y})$ where $\mathbf{x}, \mathbf{y} \in \mathcal{B}(S)$. We say that ρ is k-step reachable if there exists an interactive process $\pi = (\zeta, \gamma, \delta)$ in CR-$\mathcal{C}$ such that $\delta = (\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_k)$, and $\mathbf{x} \leq \mathbf{d}_k$, $\mathbf{d}_k(e) < \mathbf{y}(e)$, for every $e \in carr(\mathbf{y})$.

5.4.1 Macro-reactions

Incrementation and decrementation operations. With $\uparrow_e^g$ and $\downarrow_e^g$ we denote the set of reactions encoding the operation of, respectively, incrementation and decrementation of concentration levels of $e \in S$ when $g \in S$ is present with a non-zero concentration. With M_e we denote the maximal allowed value of e. Then:

$$\uparrow_e^g \stackrel{def}{=} \{(\{e \mapsto i, g \mapsto 1\}, \varnothing_S, \{e \mapsto i+1\}) \mid 1 \leq i < M_e\}$$

and

$$\downarrow_e^g \stackrel{def}{=} \{(\{e \mapsto i, g \mapsto 1\}, \varnothing_S, \{e \mapsto i-1\}) \mid 2 < i \leq M_e\}.$$

Permanency. In a similar way we introduce a set of reactions for encoding permanency:

$$\Diamond_e^{\mathbf{i}} \stackrel{def}{=} \{(\{e \mapsto i\}, \mathbf{i}, \{e \mapsto i\}) \mid 1 \leq i \leq M_e\}$$

The above is a set of reactions ensuring permanency of $e \in S$ that can be inhibited by $\mathbf{i} \in \mathcal{B}(S)$.

We exploit the notation to use $\uparrow_e^g$, $\downarrow_e^g$, and $\Diamond_e^{\mathbf{i}}$ in place of regular reactions ignoring that they are in fact sets of reactions. In the implementation for CRRSC we introduce an optimisation where these reactions are encoded as *macro-reactions*, that is, as simple operations on integer variables that increment, decrement, or retain the value of the variable encoding concentration of e.

We assume the macro-reactions are allowed only when no ordinary reaction is enabled.

5.4.2 Eukaryotic heat shock response

We test our implementation using an adaptation (CHSR) of the model of the eukaryotic heat shock response (HSR) described in Chapter 3. The adaptation defined below modifies HSR such that it uses concentrations. The heat shock response model was originally introduced in [Azimi *et al.*, 2014b].

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*
temp	temperature value
cool	decreases the temperature
heat	increases the temperature

Table 5.1: Entities used in the heat shock response model.

The HSR model used *stress* and *nostress* entities to distinguish between the presence and absence of the heat shock and it is assumed that the heat shock occurs at (and above) the temperature of $42\,^{\circ}C$. In CHSR this is modelled using the *temp* entity. All the entities except *temp* remain at the concentration level of one unit. We assume that the maximal value of the temperature modelled using the entity *temp* is 50.

The background set S for the RSC modelling CHSR consists of the entities in Table 5.1. The set A_{ord} comprises the reactions in Table 5.2. We also define the set of reactions dealing with temperature $A_{temp} = \uparrow_{temp}^{heat} \cup \downarrow_{temp}^{cool} \cup \diamond_{temp}^{\mathbf{i}}$, where $\mathbf{i} = \{heat \mapsto 1, cool \mapsto 1\}$. By defining the set $\mathbf{i}$ in this way we ensure that the

reactants	inhibitors	products
$hsf \mapsto 1$	$hsp \mapsto 1$	$hsf_{\,3} \mapsto 1$
$hsf \mapsto 1, hsp \mapsto 1, mfp \mapsto 1$	$\varnothing_S$	$hsf_3 \mapsto 1$
$hsf_{\,3} \mapsto 1$	$hsp \mapsto 1, hse \mapsto 1$	$hsf \mapsto 1$
$hsp \mapsto 1, hsf_3 \mapsto 1, mfp \mapsto 1$	$hse \mapsto 1$	$hsf \mapsto 1$
$hsf_3 \mapsto 1, hse \mapsto 1$	$hsp \mapsto 1$	$hsf_3{:}hse \mapsto 1$
$hsp \mapsto 1, hsf_3 \mapsto 1, mfp \mapsto 1, hse \mapsto 1$	$\varnothing_S$	$hsf_3{:}hse \mapsto 1$
$hse \mapsto 1$	$hsf_3 \mapsto 1$	$hse \mapsto 1$
$hsp \mapsto 1, hsf_3 \mapsto 1, hse \mapsto 1$	$mfp \mapsto 1$	$hse \mapsto 1$
$hsf_3{:}hse \mapsto 1$	$hsp \mapsto 1$	$hsp \mapsto 1, hsf_3{:}hse \mapsto 1$
$hsp \mapsto 1, mfp \mapsto 1, hsf_3{:}hse \mapsto 1$	$\varnothing_S$	$hsp \mapsto 1, hsf_3{:}hse \mapsto 1$
$hsf \mapsto 1, hsp \mapsto 1$	$mfp \mapsto 1$	$hsp{:}hsf \mapsto 1$
$hsp{:}hsf \mapsto 1, temp \mapsto 42$	$\varnothing_S$	$hsf \mapsto 1, hsp \mapsto 1$
$hsp{:}hsf \mapsto 1$	$temp \mapsto 42$	$hsp{:}hsf \mapsto 1$
$hsp \mapsto 1, hsf_3 \mapsto 1$	$mfp \mapsto 1$	$hsp{:}hsf \mapsto 1$
$hsp \mapsto 1, hsf_3{:}hse \mapsto 1$	$mfp \mapsto 1$	$hse \mapsto 1, hsp{:}hsf \mapsto 1$
$temp \mapsto 42, prot \mapsto 1$	$\varnothing_S$	$mfp \mapsto 1, prot \mapsto 1$
$prot \mapsto 1$	$temp \mapsto 42$	$prot \mapsto 1$
$hsp \mapsto 1, mfp \mapsto 1$	$\varnothing_S$	$hsp{:}mfp \mapsto 1$
$mfp \mapsto 1$	$hsp \mapsto 1$	$mfp \mapsto 1$
$hsp{:}mfp \mapsto 1$	$\varnothing_S$	$hsp \mapsto 1, prot \mapsto 1$

Table 5.2: Reactions of CHSR (curly brackets are omitted)

result of changing the temperature will not be overridden due to the permanency. Finally, the RSC for CHSR is defined as:

$$\mathcal{C}_{\text{CHSR}} = (S, A_{ord} \cup A_{temp}).$$

To define a CRRSC for $\mathcal{C}_{\text{HSR}}$ we use the context automaton $\mathfrak{A}_{\text{HSR}} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ where $\mathcal{Q} = \{\mathfrak{q}_0, \mathfrak{q}_1\}$, $\mathfrak{q}^{init} = \mathfrak{q}_0$ and

$$\begin{aligned} R = \{ & \mathfrak{q}_0 \xrightarrow{\{hsf \mapsto 1, prot \mapsto 1, hse \mapsto 1, temp \mapsto 35\}} \mathfrak{q}_1, \\ & \mathfrak{q}_1 \xrightarrow{\{cool \mapsto 1\}} \mathfrak{q}_1, \\ & \mathfrak{q}_1 \xrightarrow{\{heat \mapsto 1\}} \mathfrak{q}_1, \\ & \mathfrak{q}_1 \xrightarrow{\varnothing_S} \mathfrak{q}_1\}. \end{aligned}$$

Then, the CRRSC for $\mathcal{C}_{\text{CHSR}}$ is defined as CR-$\mathcal{C}_{\text{CHSR}} = (\mathcal{C}_{\text{CHSR}}, \mathfrak{A}_{\text{CHSR}})$. The context set specified in $\mathfrak{A}_{\text{CHSR}}$ for the transition from 0 (the initial state) corresponds to the initial context set used in [Azimi *et al.*, 2014b] as the minimal set of entities needed in HSR, together with the *temp* entity indicating a temperature that does not cause the heat shock.

First, we test the efficiency of our implementation by verifying the reachability of the following results of CR-$\mathcal{C}_{\text{CHSR}}$:

- $\rho_1 = (\mathbf{x}_1, \mathbf{y}_1)$ where:
  - $\mathbf{x}_1 = \{hsp{:}hsf \mapsto 1, hse \mapsto 1, prot \mapsto 1\}$,
  - $\mathbf{y}_1 = \{temp \mapsto 42\}$;
- $\rho_2 = (\mathbf{x}_2, \mathbf{y}_2)$ where $\mathbf{x}_2 = \{mfp \mapsto 1\}$, $\mathbf{y}_2 = \varnothing_S$.

Reachability of ρ_1 proves that it is possible to enter the state where CHSR may become stable, while reachability of ρ_2 proves that it is possible for the proteins to eventually misfold. The k-step reachability for ρ_1 is proved for $k = 4$, while ρ_2

	ρ_1		ρ_2	
	time [s]	memory [MB]	time [s]	memory [MB]
CRRS	17.32	25.08	38.78	28.38
CRRSC	**0.35**	**24.87**	**0.93**	**24.99**
improvement	49.48×	1.01×	41.69×	1.13×

Table 5.3: Results for the verification of reachability properties of CHSR

for $k = 9$. There is no noticeable improvement in memory consumption for the verification of CRRSC over CRRS. However, there is a significant difference in the execution times in favour of CRRSC, e.g., for ρ_1 the verification for CRRSC is 49.48 times faster. The verification results[2] for the reachability properties are summarised in Table 5.3.

The verification results for rsLTL formulae are presented in Table 5.4. The verification of the formula ϕ_2 requires more resources than ϕ_1, since the result is found for a larger value of k and the verified property contains more temporal operators, resulting in a larger encoding.

5.4.3 Scalable chain

For the next benchmark we introduce the *scalable chain* (SC) model. The background set for the system is defined as $S = \{e_1, e_2, \ldots, e_m, inc, dec\}$. Intuitively,

[2]The experimental results were obtained using a system equipped with 3.7GHz Intel Xeon E5 processor and 12GB of memory, running Mac OS X 10.12.3.

Formula		k	time [s]	memory [MB]
ϕ_1	$\mathsf{XF}_{heat>0}(temp > 42)$	9	1.01	30.04
ϕ_2	$\mathsf{XG}_{heat>0}((temp > 42) \Rightarrow \mathsf{F}(mfp > 0))$	21	3.18	34.77

Table 5.4: rsLTL formulae for HSR with the verification performance

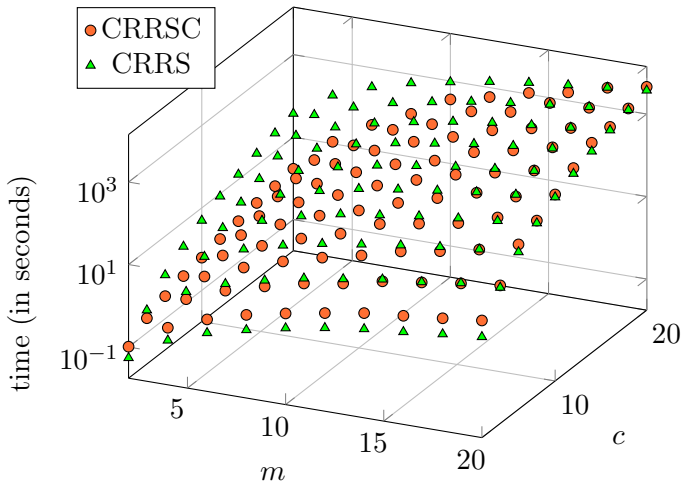

Figure 5.2: Verification results for reachability in SC: execution time

the system executes reactions incrementing concentration levels of m entities, each up to a maximal concentration level c. For $i < m$, when the maximal concentration level of e_i is reached, then the entity e_{i+1} is produced.

The *inc* and *dec* entities cause, respectively, incrementation or decrementation of concentration levels. We define the following sets of reactions:

- $\mathcal{P} = \big\{(\{e_i \mapsto c\}, \varnothing_S, \{e_{i+1} \mapsto 1)\}) \mid 1 \leq i < m\big\}$,
- $\mathcal{O} = \big\{ \uparrow_{e_i}^{inc}, \downarrow_{e_i}^{dec} \mid 1 \leq i \leq m\big\}$,
- $\mathcal{F} = \big\{(\{e_m \mapsto c\}, \{dec \mapsto 1\}, \{e_m \mapsto c\})\big\}$.

The reactions of $\mathcal{P}$ implement the production of the subsequent entities, while their concentration levels are changed by the reactions of $\mathcal{O}$. The reaction of $\mathcal{F}$ ensures persistency of the "final" entity e_m when it reaches the concentration of c, unless *dec* is present. The RSC for the scalable chain system is defined as $\mathcal{C}_{\text{SC}} = (S, \mathcal{P} \cup \mathcal{O} \cup \mathcal{F})$. Next, we define the context automaton $\mathfrak{A}_{\text{SC}} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ where $\mathcal{Q} = \{\mathfrak{q}_0, \mathfrak{q}_1\}$, $\mathfrak{q}^{init} = \mathfrak{q}_0$, and the set R consists of the following transitions:

- $\mathfrak{q}_0 \xrightarrow{\{e_1 \mapsto 1, inc \mapsto 1\}} \mathfrak{q}_1$,
- $\mathfrak{q}_1 \xrightarrow{\{inc \mapsto 1\}} \mathfrak{q}_1$,
- $\mathfrak{q}_1 \xrightarrow{\{dec \mapsto 1\}} \mathfrak{q}_1$.

Finally, we define CR-$\mathcal{C}_{\text{SC}} = (\mathcal{C}_{\text{SC}}, \mathfrak{A}_{\text{SC}})$.

The verified reachability property of the scalable chain system is proved for $k = m \cdot c - 1$. The property expresses the reachability of the maximal concentration level of the entity e_m. The time and memory consumption results are presented in Fig. 5.2–5.3.

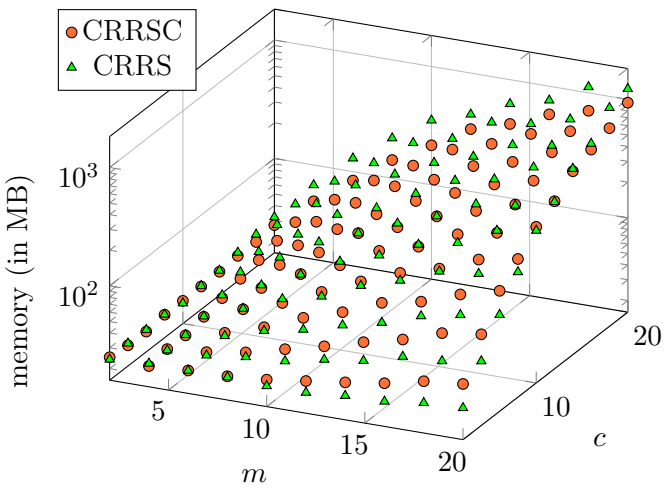

Figure 5.3: Verification results for reachability in SC: memory consumption

In most cases there is an observable advantage of the implementation for CRRSC when the value of c is relatively large compared to m, e.g., for $m = 8$ and $c = 20$ the results for CRRSC are 5.6 times better. For $m = 10$ and $c = 14$ the verification of CRRS proved to be 1.6 times more efficient as it only consumed 1334 seconds, compared to 2155 seconds for CRRSC. However, for $m = 20$ and $c = 16$ CRRS was only 1.2 times better. We attribute this inconsequence to the heuristics of the SMT-solver used. The CRRSC implementation appears to be more memory-efficient when dealing with larger concentration level values. It appears that when the verified system is highly-dependent on a large domain of concentration levels, then the CRRSC will most likely be more suitable.

To test the performance of our rsLTL implementation we use a fixed maximal concentration level $c = 2$ and verify the properties presented in Table 5.5. The time and memory consumption results are presented in Fig. 5.4–5.5. The properties expressed with ϕ_1, ϕ_2, and ϕ_3 are proved for variable values of k that depend on the scaling parameter m. Verifying the formula ϕ_2 requires the most resources since it contains multiple nested operators that also result in multiple levels of recursion when computing the translation. Our implementation proved to be the most efficient for ϕ_4 and ϕ_5. This is mostly due to the very low and constant value of k. This means that only a very small portion of the model needs to be traversed to prove these properties.

5.5 Concluding remarks

We introduced reaction systems with discrete concentrations, which support quantitative modelling. Although the formalism is not more expressive than the standard reaction systems, our experimental results demonstrate that expressing concen-

ϕ_1	$\mathsf{F}_{inc>0}(e_m = c)$
ϕ_2	$\phi_2 = \phi_2^1$, $\phi_2^i = \mathsf{F}_{inc>0}((e_i = c) \wedge \phi_2^i)$ for $i \in \{1, \ldots, m-1\}$, where $\phi_2^m = \mathsf{F}_{inc>0}((e_m = c))$
ϕ_3	$\mathsf{G}((e_1 = 1) \Rightarrow \mathsf{F}_{inc>0}(e_m = c))$
ϕ_4	$\mathsf{F}_{inc>0}(e_1 = c)$
ϕ_5	$\mathsf{X}((e_1 > 0)\mathsf{R}_{inc>0}(e_2 > 0))$

Table 5.5: rsLTL formulae for SC

Formula	k
ϕ_1	$2 \cdot m - 1$
ϕ_2	$2 \cdot m - 1$
ϕ_3	$2 \cdot m$
ϕ_4	1
ϕ_5	2

Table 5.6: Witness lengths for SC

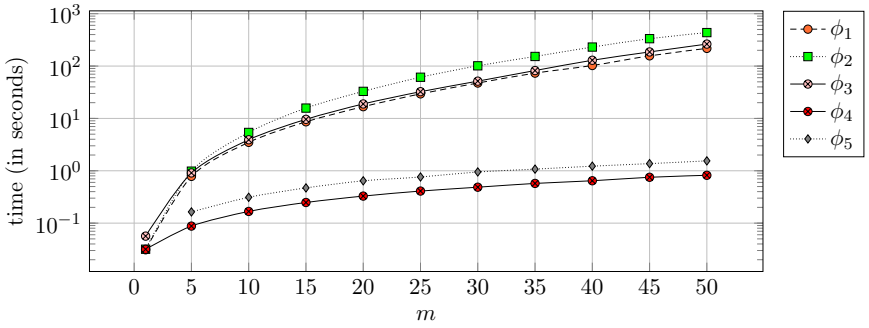

Figure 5.4: Verification results for rsLTL properties of SC: execution time

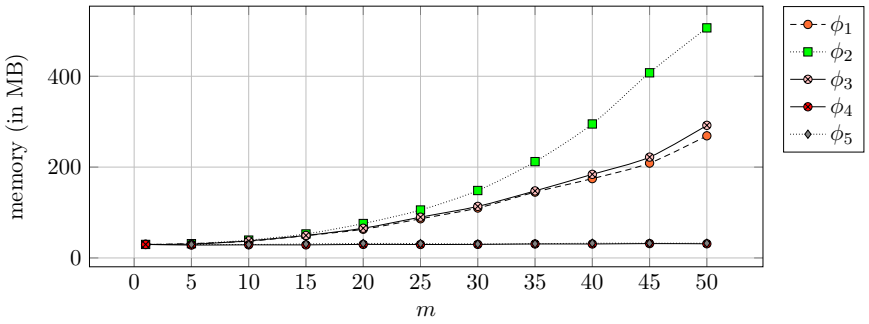

Figure 5.5: Verification results for rsLTL properties of SC: memory consumption

tration levels in an explicit way allows for some improvements in the efficiency of verification, and opens up possibilities for introducing different optimisations. The computational complexity of the model checking problem for reaction systems brings limitations to the practical applicability of the method. However, our experimental results demonstrate that the presented method scales well when verifying properties of large models.

Chapter 6

Parametric model checking for rsLTL

In this chapter we introduce parametric reaction systems and propose a method for reaction synthesis that is based on bounded model checking for rsLTL presented in the previous chapter.

6.1 Parametric reaction systems

In *parametric reaction systems*, reactions can be defined partially, i.e., reactants, inhibitors, and products in *parametric reactions* can be replaced with *parameters*.

Definition 6.1.1. A *parametric reaction system* (PRS) is a triple $\mathcal{P} = (S, P, A)$, where:

- S is a finite *background* set,
- P is a finite set of elements called *parameters*, and
- A is a nonempty finite set of *parametric reactions* over the background set, where each $a \in A$ is a triple $a = (\mathfrak{r}, \mathfrak{i}, \mathfrak{p})$ such that $\mathfrak{r}, \mathfrak{i}, \mathfrak{p} \in \mathcal{B}(S) \cup P$.

The elements $\mathfrak{r}$, $\mathfrak{i}$, and $\mathfrak{p}$ are respectively denoted by $\mathfrak{r}_a$, $\mathfrak{i}_a$, and $\mathfrak{p}_a$ and called the *reactants*, *inhibitors*, and *products* of parametric reaction a.

Definition 6.1.2. Let $\mathcal{P} = (S, P, A)$ be a PRS. A *parameter valuation* of $\mathcal{P}$ is a function $\mathtt{v} : P \cup \mathcal{B}(S) \to \mathcal{B}(S)$ such that $\mathtt{v}(\mathfrak{b}) = \mathfrak{b}$ if $\mathfrak{b} \in \mathcal{B}(S)$.

We also write $\mathfrak{b}^{\leftarrow \mathtt{v}}$ for $\mathtt{v}(\mathfrak{b})$. The set of all the parameter valuations for $\mathcal{P}$ is denoted by $\mathtt{PV}_{\mathcal{P}}$. Let $\mathtt{v} \in \mathtt{PV}_{\mathcal{P}}$. For any subset $X \subseteq A$ of reactions of $\mathcal{P}$ we define:

$$X^{\leftarrow \mathtt{v}} = \{(a_{\mathfrak{r}}^{\leftarrow \mathtt{v}}, a_{\mathfrak{i}}^{\leftarrow \mathtt{v}}, a_{\mathfrak{p}}^{\leftarrow \mathtt{v}}) \mid a \in X\}.$$

By $\mathcal{P}^{\leftarrow \mathtt{v}}$ we denote the structure $(S, A^{\leftarrow \mathtt{v}})$ where all the parameters in A are substituted according to the parameter valuation $\mathtt{v}$. We say that $\mathtt{v} \in \mathtt{PV}_{\mathcal{P}}$ is a *valid parameter valuation* if $\mathcal{P}^{\leftarrow \mathtt{v}}$ yields an RSC.

Definition 6.1.3. A *context-restricted parametric reaction system* (CRPRS) is a pair CR-$\mathcal{P} = (\mathcal{P}, \mathfrak{A})$ such that $\mathcal{P} = (S, P, A)$ is a PRS and $\mathfrak{A} = (\mathcal{Q}, \mathfrak{q}^{init}, R)$ is a context automaton over S.

For $\mathtt{v} \in \mathtt{PV}_{\mathcal{P}}$ we define CR-$\mathcal{P}^{\leftarrow \mathtt{v}} = (\mathcal{P}^{\leftarrow \mathtt{v}}, \mathfrak{A})$.

Example 6.1.4. We consider a simple PRS for a simplified abstract genetic regulatory system based on Example 2.2.2. The system contains two (abstract) genes x and y expressing proteins X and Y, respectively, and a protein complex Q formed by X and Y. The background set is defined as $S = \{x, \widehat{x}, X, y, \widehat{y}, Y, h, Q\}$, where $\widehat{x}$ and $\widehat{y}$ denote RNA polymerase attached to the promoter of genes x and y, respectively. Here h is used as an abstract inhibitor. Finally, the set of parametric reactions consists of the following subsets:

- $A_x = \{(\{x\}, \{h\}, \{x\}), (\{x\}, \{h\}, \{\widehat{x}\}), (\{x, \widehat{x}\}, \{h\}, \{X\})\}$,
- $A_y = \{(\{y\}, \{h\}, \lambda_1), (\lambda_2, \{h\}, \{\widehat{y}\}), (\{y, \widehat{y}\}, \{h\}, \lambda_3)\}$,
- $A_Q = \{(\{X, Y\}, \{h\}, \{Q\})\}$.

Notice that the reactions of A_y use parameters $\lambda_1, \lambda_2, \lambda_3$ to define expression of the protein Y.

Suppose that we investigate the processes starting from the states that already contain x and y. This leads to the following definition of the context automaton: $\mathfrak{A} = (\{\mathfrak{q}_0, \mathfrak{q}_1\}, \mathfrak{q}_0, R)$, where: $R = \{\mathfrak{q}_0 \xrightarrow{\{x,y\}} \mathfrak{q}_1, \mathfrak{q}_1 \xrightarrow{\varnothing} \mathfrak{q}_1, \mathfrak{q}_1 \xrightarrow{\{h\}} \mathfrak{q}_0)\}$. When the context set contains the entity h, $\mathfrak{A}$ reverts back to the initial location, while for the empty context the automaton remains in $\mathfrak{q}_1$.

Finally, the CRPRS is defined as CR-$\mathcal{P} = ((S, P, A), \mathfrak{A})$, where: $P = \{\lambda_1, \lambda_2, \lambda_3\}$ and $A = A_x \cup A_y \cup A_Q$. □

We focus on the synthesis of a parameter valuation, given n observations of the behaviour of the system that are expressed with rsLTL formulae.

Let CR-$\mathcal{P} = (\mathcal{P}, \mathfrak{A})$ be a CRPRS and $F = \{\phi_1, \ldots, \phi_n\}$ be a set of rsLTL formulae. The aim of *parameter synthesis* for CRPRS is to find a valid parameter valuation $\mathtt{v}$ of CR-$\mathcal{P}$ such that:

$$\big(\mathcal{M}(\text{CR-}\mathcal{P}^{\leftarrow \mathtt{v}}) \models_{\exists} \phi_1\big) \wedge \cdots \wedge \big(\mathcal{M}(\text{CR-}\mathcal{P}^{\leftarrow \mathtt{v}}) \models_{\exists} \phi_n\big).$$

Each formula of F corresponds to an interactive process observed in the analysed system via, e.g., experiments or simulations. Therefore, for each such process we expect an individual path in $\mathcal{M}(\text{CR-}\mathcal{P}^{\leftarrow \mathtt{v}})$ and we solve the n model checking problem instances for rsLTL in one instance. However, the parameter valuation v is shared among all instances, which allows us to calculate v for which all the properties of F are satisfied.

Example 6.1.5. Let us assume we performed an experiment on the system from Example 6.1.4 where protein Y was expressed, and we collected the following observations related to the expression of Y:

– when the current state contains y, then y and $\widehat{y}$ are present in the next state:

$$\phi_1^c = \mathsf{G}_{\neg h}(y \Rightarrow \mathsf{X}(y \wedge \widehat{y})),$$

– when y and $\widehat{y}$ are present, then Y is finally produced:

$$\phi_2^c = \mathsf{G}_{\neg h}((y \wedge \widehat{y}) \Rightarrow \mathsf{F}Y),$$

– the entities y, $\widehat{y}$, and Y are eventually produced:

$$\phi^r = (\mathsf{F}_{\neg h} y) \wedge (\mathsf{F}_{\neg h} \widehat{y}) \wedge (\mathsf{F}_{\neg h} Y).$$

These observations are made assuming h is not provided in the context set. Additionally, we observe that the protein Q is not present in the first three steps of the execution and then, after an arbitrary number of steps, it is finally produced:

$$\phi^d = \neg Q \wedge \mathsf{X}(\neg Q \wedge \mathsf{X}(\neg Q \wedge \mathsf{F}Q)).$$

The observations are related to a single interactive process (or an experiment), therefore we constrain the problem using the conjunction of all the observations. Finally, the observations are expressed using the following rsLTL formula:

$$\phi_y = \phi^r \wedge \phi_1^c \wedge \phi_2^c \wedge \phi^d.$$

We perform parameter synthesis for $F = \{\phi_y\}$, that is, we obtain a valid parameter valuation $\mathtt{v}$ such that $\mathcal{M}(\text{CR-}\mathcal{P}^{\leftarrow \mathtt{v}}) \models_\exists \phi$. In fact, it may be possible to obtain more than one such valuation. A parameter valuation $\mathtt{v}_1$ such that

$$\lambda_1^{\leftarrow \mathtt{v}_1} = \{y\},\ \lambda_2^{\leftarrow \mathtt{v}_1} = \{y\},\ \lambda_3^{\leftarrow \mathtt{v}_1} = \{Y\}$$

is valid and satisfies the requirements of our observations. A parameter valuation $\mathtt{v}_2$ such that

$$\lambda_1^{\leftarrow \mathtt{v}_2} = \{X, y\},\ \lambda_2^{\leftarrow \mathtt{v}_2} = \{x, \widehat{x}, y\},\ \lambda_3^{\leftarrow \mathtt{v}_2} = \{X, y, \widehat{y}, Y, Q\}$$

is an another example of a valid valuation which satisfies the requirements. □

Example 6.1.6. We introduce an additional *unknown* into the system declared in Example 6.1.4. That is, we add a parameter λ_4 and re-define the reactions of A_x in such a way that one of them uses the newly introduced parameter:

$$A_x = \{(\{x\}, \{h\}, \{x\}), (\lambda_4, \{h\}, \{\widehat{x}\}), (\{x, \widehat{x}\}, \{h\}, \{X\})\}.$$

Let us assume that in another experiment we observed when the current state contains x, then x and $\widehat{x}$ are found in the next state. This is expressed with the formula $\phi_x = \mathsf{G}_{\neg h}(x \Rightarrow \mathsf{X}(x \wedge \widehat{x}))$. Next, we perform parameter synthesis for $F = \{\phi_x, \phi_y\}$, where ϕ_y is the formula from Example 6.1.5. We use two rsLTL formulae in F since our observations were gathered in two separate experiments and may be related to separate interactive processes. A parameter valuation $\mathtt{v}$ such that

$$\lambda_1^{\leftarrow \mathtt{v}} = \{X, y\},\ \lambda_2^{\leftarrow \mathtt{v}} = \{y\},\ \lambda_3^{\leftarrow \mathtt{v}} = \{\widehat{x}, Y\},\ \lambda_4^{\leftarrow \mathtt{v}} = \{Q\}$$

is valid and satisfies the requirements of our observations. □

The parameter valuation $\mathtt{v}_2$ obtained in Example 6.1.5 and the valuation from Example 6.1.6 do not result in the same reactions as the original ones from Example 2.2.2. This might be undesired, depending on the application of the synthesis and the knowledge of the system under analysis. To address this issue, in the following section we introduce parameter constraints which allow for providing additional restrictions on the parameters used in the synthesis.

6.1.1 Parameter constraints

In some cases restricting parameter valuations using only rsLTL formulae may prove to be less efficient than constraining the valuation using specialised constraints for the parameters of a PRS.

Definition 6.1.7. The grammar of the *parameter constraints* for $\mathcal{P} = (S, P, A)$ is defined as follows:

$$\mathfrak{c} ::= \mathit{true} \mid \lambda[e] \sim c \mid \lambda[e] \sim \lambda[e] \mid \neg\mathfrak{c} \mid \mathfrak{c} \vee \mathfrak{c},$$

where $\lambda \in P$, $e \in S$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$.

The set of all the parameter constraints for $\mathcal{P}$ is denoted by $PC(\mathcal{P})$. Intuitively, $\lambda[e]$ can be used to refer to the concentration of $e \in S$ in the multisets corresponding to the valuations of λ.

Definition 6.1.8. Let $\mathtt{v}$ be a parameter valuation of $\mathcal{P}$. The fact that a parameter constraint $\mathfrak{c}$ holds in $\mathtt{v}$ is denoted by $\mathtt{v} \models_p \mathfrak{c}$ and defined as follows:

$\mathtt{v} \models_p \mathit{true}$	for every $\mathtt{v}$,	
$\mathtt{v} \models_p \lambda[e] \sim c$	iff	$\lambda^{\leftarrow \mathtt{v}}(e) \sim c$,
$\mathtt{v} \models_p \lambda_1[e_1] \sim \lambda_2[e_2]$	iff	$\lambda_1^{\leftarrow \mathtt{v}}(e_1) \sim \lambda_2^{\leftarrow \mathtt{v}}(e_2)$,
$\mathtt{v} \models_p \neg\mathfrak{c}$	iff	$\mathtt{v} \not\models_p \mathfrak{c}$,
$\mathtt{v} \models_p \mathfrak{c}_1 \vee \mathfrak{c}_2$	iff	$\mathtt{v} \models_p \mathfrak{c}_1$ or $\mathtt{v} \models_p \mathfrak{c}_2$.

Definition 6.1.9. A *constrained parametric reaction system* (CPRS) is a tuple $\mathcal{CP} = (S, P, A, \mathfrak{c})$ such that $\mathcal{P} = (S, P, A)$ is a PRS and $\mathfrak{c} \in PC(\mathcal{P})$.

For $\mathtt{v} \in \mathtt{PV}_{\mathcal{P}}$, we then define $\mathcal{CP}^{\leftarrow \mathtt{v}} = \mathcal{P}^{\leftarrow \mathtt{v}}$. A parameter valuation $\mathtt{v} \in \mathtt{PV}_{\mathcal{P}}$ is *valid* in $\mathcal{CP}$ if it is valid in $\mathcal{P}$ and $\mathtt{v} \models_p \mathfrak{c}$.

Definition 6.1.10. A *context-restricted CPRS* (CR-CPRS) is a pair CR-$\mathcal{CP} = (\mathcal{CP}, \mathfrak{A})$ such that $\mathcal{CP} = (S, P, A, \mathfrak{c})$ is a CPRS and $\mathfrak{A}$ is a context automaton over S.

We also denote CR-$\mathcal{CP}^{\leftarrow \mathtt{v}} = (\mathcal{CP}^{\leftarrow \mathtt{v}}, \mathfrak{A})$.

Example 6.1.11. Let us consider the system used in Example 6.1.6. We might want to assume that the parameters used in the reactions of A_y do not use any of the entities used in the reactions of A_x. Let $E = \{x, \widehat{x}, X\}$ be the set of entities we want to exclude and that are used in A_x. The described restriction can be expressed using the following parameter constraint:

$$\bigwedge_{i \in \{1,\ldots,3\}} \bigwedge_{e \in E} (\lambda_i[e] = 0)$$

Similarly, we can express that the parameter λ_4 used in the reaction of A_x is only allowed to use the entities of E:

$$\bigwedge_{e \in (S \setminus E)} (\lambda_4[e] = 0).$$

□

Example 6.1.12. It is possible to constrain multisets corresponding to parameters. Suppose $\lambda_1, \lambda_2, \lambda_3 \in P$. To constrain $\lambda_1^{\leftarrow \mathtt{v}}$ to be a sub-multiset of $\lambda_2^{\leftarrow \mathtt{v}}$ (i.e., $\lambda_1^{\leftarrow \mathtt{v}} \subseteq \lambda_2^{\leftarrow \mathtt{v}}$, for all $\mathtt{v}$), we define:

$$\mathit{submset}(\lambda_1, \lambda_2) = \bigwedge_{e \in S} (\lambda_1[e] \leq \lambda_2[e]).$$

To constrain $\lambda_3^{\leftarrow \mathtt{v}}$ to be the intersection of $\lambda_1^{\leftarrow \mathtt{v}}$ and $\lambda_2^{\leftarrow \mathtt{v}}$ (i.e., $\lambda_1^{\leftarrow \mathtt{v}} \cap \lambda_2^{\leftarrow \mathtt{v}} = \lambda_3^{\leftarrow \mathtt{v}}$, for all $\mathtt{v}$), we define:

$$\mathit{intersect}(\lambda_1, \lambda_2, \lambda_3) = \bigwedge_{e \in S} \Big(\big((\lambda_1[e] > \lambda_2[e]) \wedge (\lambda_3[e] = \lambda_2[e]) \big) \\ \vee \big((\lambda_1[e] \leq \lambda_2[e]) \wedge (\lambda_3[e] = \lambda_1[e]) \big) \Big).$$

□

The parameter synthesis problem for CR-CPRS is defined similarly as for CRPRS. Let CR-$\mathcal{CP} = (\mathcal{CP}, \mathfrak{A})$ and $F = \{\phi_1, \ldots, \phi_n\}$ be a set of rsLTL formulae. The aim of *parameter synthesis* for CR-CPRS is to find a valid parameter valuation $\mathtt{v}$ of CR-$\mathcal{CP}$ such that:

$$(\mathcal{M}(\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}) \models_\exists \phi_1) \wedge \ldots \wedge (\mathcal{M}(\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}) \models_\exists \phi_n). \tag{6.1}$$

Next, we define the *nonemptiness checking* problem, which is a decision problem related to the problem of parameter synthesis. This problem consists in checking if there exists a valuation $\mathtt{v}$ such that the condition (6.1) holds.

6.1.2 Complexity analysis

Theorem 6.1.13. *The nonemptiness checking problem for CR-CPRS and rsLTL is PSPACE-complete.*

Proof. The lower bound follows directly from Corollary 5.2.9, hence the problem is PSPACE-hard.

For the upper bound we need to show the problem is in PSPACE. To show this we define a nondeterministic space-bounded algorithm and use Lemma 5.2.11. Algorithm 17 presents an outline for the nonemptiness checking procedure. First,

Algorithm 17: Nondeterministic procedure for nonemptiness checking

```
1: guess v ∈ PV_P
2: if v ⊨_p c then
3:    R := true
4:    for all φ ∈ F do
5:       R := (M(CR-CP^{←v}) ⊨_∃ φ) ∧ R
6:    end for
7:    if R = true then
8:       return true
9:    end if
10: end if
```

the algorithm nondeterministically generates a valuation $\mathtt{v} \in \mathtt{PV}_{\mathcal{P}}$. If $\mathtt{v}$ is valid in CR-$\mathcal{CP}$, then it proceeds to verifying the rsLTL formulae. For all the formulae $\phi \in F$ the algorithm performs existential rsLTL model checking in $\mathcal{M}(\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}})$. From Lemma 5.2.11 and the fact that PSPACE is closed under complementation, i.e., PSPACE = *co*PSPACE, the existential variant of the rsLTL model checking problem is also in PSPACE. The nonemptiness checking algorithm requires the space needed by the algorithm for rsLTL model checking. Since all the $|F|$ model checking instances are constructed independently and the algorithm only stores the overall result R, the algorithm requires space for at most one instance at any given time. Additionally, the algorithm requires space $\mathcal{O}(|A| \cdot |S|)$ to store the valuation $\mathtt{v}$ and $\mathcal{O}(1)$ for the verification result R. Therefore, the problem remains in PSPACE and given the lower bound we conclude the problem is PSPACE-complete. □

In the following section we show how the synthesis problem can be solved using an incremental approach, which amounts to checking

$$(\mathcal{M}(\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}) \models^{k}_{\exists} \phi_1) \wedge \ldots \wedge (\mathcal{M}(\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}) \models^{k}_{\exists} \phi_n)$$

for $k \geq 0$, by increasing the value of k until a valid parameter valuation is found.

6.2 SMT-based encoding

In this section we provide a translation of the parameter synthesis problem for CR-CPRS and rsLTL into the satisfiability modulo theory (SMT) with the integer arithmetic theory.

Let CR-$\mathcal{CP} = ((S, P, A, \mathfrak{c}), (\mathcal{Q}, \mathfrak{q}^{init}, R))$ and $F = \{\phi_1, \ldots, \phi_n\}$ be a set of rsLTL formulae. We encode the model $\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}$, where $\mathtt{v}$ is a valid parameter valuation of CR-$\mathcal{CP}$. Let $k \geq 0$ be an integer, then for each $f \in \{1, \ldots, n\}$ we encode any possible path prefix of $\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}$. The encoded path prefixes are bounded with k. That is, for each formula ϕ_f we encode a separate path prefix representing its witness. The entities of S are denoted by $e_1, \ldots, e_m$, where $m = |S|$. For each $\phi_f \in F$ and $i \in \{0, \ldots, k\}$ we introduce sets of positive integer variables:

$$\begin{aligned}\mathbf{P}_{f,i} &= \{\mathtt{p}_{f,i,1}, \ldots, \mathtt{p}_{f,i,m}\},\\ \mathbf{P}^{\mathcal{E}}_{f,i} &= \{\mathtt{p}^{\mathcal{E}}_{f,i,1}, \ldots, \mathtt{p}^{\mathcal{E}}_{f,i,m}\},\\ \mathbf{Q}_f &= \{\mathtt{q}_{f,0}, \ldots, \mathtt{q}_{f,k}\}.\end{aligned}$$

Let $\mathtt{ta} : A \to \{1, \ldots, |A|\}$ be a bijection mapping all the reactions to integers. For each $a \in A$ we introduce the set of variables encoding the products:

$$\mathbf{P}^{p}_{f,i,a} = \{\mathtt{p}^{p}_{f,i,\mathtt{ta}(a),1}, \ldots, \mathtt{p}^{p}_{f,i,\mathtt{ta}(a),m}\}.$$

Let $\sigma.f$ be a path of $\mathcal{M}(\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}})$, then

$$\begin{aligned}\overline{\mathbf{p}}_{f,i} &= (\mathtt{p}_{f,i,1}, \ldots, \mathtt{p}_{f,i,m}) \text{ and}\\ \overline{\mathbf{p}}^{\mathcal{E}}_{f,i} &= (\mathtt{p}^{\mathcal{E}}_{f,i,1}, \ldots, \mathtt{p}^{\mathcal{E}}_{f,i,m})\end{aligned}$$

are used to encode $(\sigma.f)_b(i)$ and $(\sigma.f)_a(i)$, respectively. With $\overline{\mathbf{p}}_{f,i}[j]$ and $\overline{\mathbf{p}}^{\mathcal{E}}_{f,i}[j]$ we denote, respectively, $\mathtt{p}_{f,i,j}$ and $\mathtt{p}^{\mathcal{E}}_{f,i,j}$. For $i \geq 1$ we define:

$$\overline{\mathbf{p}}^{p}_{f,i} = (\mathtt{p}^{p}_{f,i,1,1}, \ldots, \mathtt{p}^{p}_{f,i,1,m}, \ldots, \mathtt{p}^{p}_{f,i,|A|,1}, \ldots, \mathtt{p}^{p}_{f,i,|A|,m}).$$

The following functions map the background set entities to the corresponding variables of the encoding: for all $i \in \{0, \ldots, k\}$ we define $\mathtt{t}_{f,i} : S \to \mathbf{P}_{f,i}$ and $\mathtt{t}^{\mathcal{E}}_{f,i} : S \to \mathbf{P}^{\mathcal{E}}_{f,i}$ such that $\mathtt{t}_{f,i}(e_j) = \mathtt{p}_{f,i,j}$ and $\mathtt{t}^{\mathcal{E}}_{f,i}(e_j) = \mathtt{p}^{\mathcal{E}}_{f,i,j}$ for all $j \in \{1, \ldots, m\}$. For all $i \in \{0, \ldots, k\}$ and $a \in A$ we define $\mathtt{t}^{p}_{f,i,a} : S \to \mathbf{P}^{p}_{f,i,a}$ such that: $\mathtt{t}^{p}_{f,i,a}(e_j) = \mathtt{p}^{p}_{f,i,\mathtt{ta}(a),j}$ for all $j \in \{1, \ldots, m\}$.

The bijection $\mathtt{e} : \mathcal{Q} \to \{1, \ldots, |\mathcal{Q}|\}$ maps the states of the context automaton to the integers used in the encoding. Let $\mathtt{tp} : P \to \{1, \ldots, |P|\}$ be a bijection mapping all the parameters to their corresponding integers. We define:

$$\overline{\mathbf{p}}^{par} = (\mathtt{p}^{par}_{1,1}, \ldots, \mathtt{p}^{par}_{1,m}, \ldots, \mathtt{p}^{par}_{|P|,1}, \ldots, \mathtt{p}^{par}_{|P|,m}).$$

For each parameter $\lambda \in P$ we define:

$$\mathbf{P}^{par}_{\lambda} = \{\mathtt{p}^{par}_{\mathtt{tp}(\lambda),1}, \ldots, \mathtt{p}^{par}_{\mathtt{tp}(\lambda),m}\}$$

and $\mathtt{pm}_\lambda : S \to \mathbf{P}^{par}_\lambda$ such that $\mathtt{pm}_\lambda(e_j) = \mathtt{p}^{par}_{\mathtt{tp}(\lambda),j}$. Let $a \in A$ and $\mathfrak{s} \in \{\mathfrak{r}_a, \mathfrak{i}_a, \mathfrak{p}_a\}$. Then, $re^{\mathfrak{s}}(e_j)$ denotes $\mathtt{pm}_{\mathfrak{s}}(e_j)$ if $\mathfrak{s} \in P$, and $\mathfrak{s}(e_j)$ otherwise. To define the SMT encoding of the paths we need auxiliary functions that correspond to elements of the encoding.

Initial state. To encode the initial state of the model for $\phi_f \in F$ we define

$$\mathsf{Init}(\overline{\mathbf{p}}_{f,i}, \mathtt{q}_{f,i}) \stackrel{def}{=} \left(\bigwedge_{e \in S} \mathtt{t}_{f,i}(e) = 0 \right) \wedge \mathtt{q}_{f,i} = \mathtt{e}(\mathfrak{q}^{init}),$$

where all the concentration levels are set to zero, and the context automaton is in its initial state.

Parameter constraints and validity. With $\mathsf{PC}(\overline{\mathbf{p}}^{par})$ we encode the parameter constraints, require that the concentration levels of the reactants are always lower than the concentration levels of the inhibitors, and ensure that all the multisets corresponding to the parameters are non-empty, i.e., for each parameter at least one entity must a have positive concentration level:

$$\mathsf{PC}(\overline{\mathbf{p}}^{par}) \stackrel{def}{=} enc_{\mathfrak{c}}(\overline{\mathbf{p}}^{par}) \wedge \left(\bigwedge_{a \in A} \bigwedge_{e \in S} re^{\mathfrak{i}_a}(e) > 0 \Rightarrow (re^{\mathfrak{r}_a}(e) < re^{\mathfrak{i}_a}(e)) \right) \wedge \left(\bigwedge_{\lambda \in P} \bigvee_{e \in S} \mathtt{pm}_\lambda(e) > 0 \right),$$

where $enc_{\mathfrak{c}}(\overline{\mathbf{p}}^{par})$ is the encoding of $\mathfrak{c}$ and it is defined over the variables of $\overline{\mathbf{p}}^{par}$. The encoding follows directly from the semantics of parameter constraints.

Parametric reactions. The parametric reactions $a \in A$ are encoded with

$$\mathsf{Rct}_a(\overline{\mathbf{p}}_{f,i}, \overline{\mathbf{p}}^{\mathcal{E}}_{f,i}, \overline{\mathbf{p}}^{p}_{f,i+1}, \overline{\mathbf{p}}^{par}) \stackrel{def}{=} \bigwedge_{e \in S} ((\mathtt{t}_{f,i}(e) \geq re^{\mathfrak{r}_a}(e) \vee \mathtt{t}^{\mathcal{E}}_{f,i}(e) \geq re^{\mathfrak{r}_a}(e)) \wedge (\mathtt{t}_{f,i}(e) < re^{\mathfrak{i}_a}(e) \wedge \mathtt{t}^{\mathcal{E}}_{f,i}(e) < re^{\mathfrak{i}_a}(e)) \wedge (\mathtt{t}^{p}_{f,a,i+1}(e) = re^{\mathfrak{p}_a}(e))).$$

Product concentration levels. With the following formula we encode the selection of the maximal concentration levels produced for each entity by all the reactions:

$$\mathsf{Results}(\overline{\mathbf{p}}_{f,i}, \overline{\mathbf{p}}^{p}_{f,i}) \stackrel{def}{=} \left(\bigwedge_{e \in S} \mathtt{t}_{f,i+1}(e) = max(\{0\} \cup \bigcup_{a \in A} \{\mathtt{t}^{p}_{f,a,i+1}(e)\}) \right).$$

Transitions of CPRS. We encode the local state changes of $\mathcal{CP}$ with the following function:

$$\mathsf{Tr}_{\mathcal{CP}}(\overline{\mathbf{p}}_{f,i}, \overline{\mathbf{p}}^{\mathcal{E}}_{f,i}, \overline{\mathbf{p}}^{p}_{f,i+1}, \overline{\mathbf{p}}_{f,i+1}, \overline{\mathbf{p}}^{par}) \stackrel{def}{=} \left(\bigwedge_{a \in A} \mathsf{Rct}_a(\overline{\mathbf{p}}_{f,i}, \overline{\mathbf{p}}^{\mathcal{E}}_{f,i}, \overline{\mathbf{p}}^{p}_{f,i+1}, \overline{\mathbf{p}}^{par}) \right) \wedge \mathsf{Results}(\overline{\mathbf{p}}_{f,i+1}, \overline{\mathbf{p}}^{p}_{f,i+1}).$$

Context. To encode a multiset $\mathbf{c} \in \mathcal{B}(S)$ of context entities we define the following function:

$$\mathsf{Ct}_{\mathbf{c}}(\overline{\mathbf{p}}^{\mathcal{E}}_{f,i}) \stackrel{def}{=} \bigwedge_{e \in S} \mathtt{t}^{\mathcal{E}}_{f,i}(e) = \mathbf{c}(e)$$

Transitions of CA. The encoding of the transition relation of the context automaton is a disjunction of the encoded transitions:

$$\mathsf{Tr}_{\mathfrak{A}}(\mathsf{q}_{f,i}, \overline{\mathbf{p}}^{\mathcal{E}}_{f,i}, \mathsf{q}_{f,i+1}) \stackrel{def}{=} \bigvee_{(\mathfrak{q},\mathbf{c},\mathfrak{q}') \in R} \left(\mathsf{q}_{f,i} = \mathsf{e}(\mathfrak{q}) \wedge \mathsf{Ct}_{\mathbf{c}}(\overline{\mathbf{p}}^{\mathcal{E}}_{f,i}) \wedge \mathsf{q}_{f,i+1} = \mathsf{e}(\mathfrak{q}') \right).$$

Transition relation. The transition relation of the model for CR-$\mathcal{CP}$ is a conjunction of the encoded transition relations for $\mathcal{CP}$ and $\mathfrak{A}$:

$$\mathsf{Tr}_{\text{CR-}\mathcal{CP}}(\overline{\mathbf{p}}_{f,i}, \mathsf{q}_{f,i}, \overline{\mathbf{p}}^{\mathcal{E}}_{f,i}, \overline{\mathbf{p}}^{p}_{f,i+1}, \overline{\mathbf{p}}_{f,i+1}, \overline{\mathbf{p}}^{par}) \stackrel{def}{=} \mathsf{Tr}_{\mathcal{CP}}(\overline{\mathbf{p}}_{f,i}, \overline{\mathbf{p}}^{\mathcal{E}}_{f,i}, \overline{\mathbf{p}}^{p}_{f,i+1}, \overline{\mathbf{p}}_{f,i+1}, \overline{\mathbf{p}}^{par}) \wedge \mathsf{Tr}_{\mathfrak{A}}(\mathsf{q}_{f,i}, \overline{\mathbf{p}}^{\mathcal{E}}_{f,i}, \mathsf{q}_{f,i+1}).$$

Finally, to encode the paths of $\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}$ that are bounded with k, we unroll the transition relation up to k and combine it with the encoding of the initial state of the model:

$$\mathsf{Paths}^k_f \stackrel{def}{=} \mathsf{Init}\left(\overline{\mathbf{p}}_{f,0}, \mathsf{q}_{f,0}\right) \wedge \bigwedge_{i=0}^{k-1} \mathsf{Tr}_{\text{CR-}\mathcal{CP}}(\overline{\mathbf{p}}_{f,i}, \mathsf{q}_{f,i}, , \overline{\mathbf{p}}^{\mathcal{E}}_{f,i}, \overline{\mathbf{p}}^{p}_{f,i+1}, \overline{\mathbf{p}}_{f,i+1}, \overline{\mathbf{p}}^{par}).$$

The encoded rsLTL formula ϕ_f at a position $i \in \{0, \ldots, k\}$ is denoted by $[\![\phi_f]\!]^k_i$. To encode the formula $[\![\phi_f]\!]^k_i$ we use our translation presented in Section 5.3. However, for each formula $\phi_f \in F$, we use independent sets of encoding variables corresponding to its path, i.e., the variables indexed with f. The encoding Loops^k_f for the loop positions is defined for each formula $\phi_f \in F$. Finally, we perform the synthesis of the parameter valuation $\mathtt{v}$ by testing the satisfiability of the following formula:

$$[\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}, F, k] = \bigwedge_{\phi_f \in F} \left(\mathsf{Paths}^k_f \wedge \mathsf{Loops}^k_f \wedge [\![\phi_f]\!]^k_0 \right) \wedge \mathsf{PC}(\overline{\mathbf{p}}^{par}). \tag{6.2}$$

The presented encoding differs from the one for CRRSC and rsLTL (Section 5.3) in how the transition relation is encoded. Here, we use an additional step that

encodes the concentration levels of each entity produced by the individual reactions. These results are then used to select the maximal concentration level produced for a given entity. This is required because some reactions produce parameters for which we do not have concretised values at the encoding stage. Therefore, it is not possible to use the technique demonstrated in Section 5.3, where the reactions are ordered and effectively only the one producing the maximal concentration level is enabled.

We show the correctness of the proposed encoding for a given valid parameter valuation.

Theorem 6.2.1. *Let* CR-$\mathcal{CP} = (\mathcal{CP}, \mathfrak{A})$ *be a CR-CPRS,* $\mathtt{v} \in \mathtt{PV}_{\text{CR-}\mathcal{CP}}$ *be a valid parameter valuation,* $\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}$ *be its model, and* F *be a set of rsLTL formulae. For any* $k \in \mathbb{N}$*, the formula* $[\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}, F, k]$ *is satisfiable iff* $\bigwedge_{\phi \in F} \left(\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}} \models^k_\exists \phi\right)$.

Proof. Since we assume a valid parameter valuation $\mathtt{v}$, to obtain a CRRSC we can perform the substitution of all the parameters that occur in CR-CPRS. Then, the proof is similar to the one of Theorem 5.3.1 for CRRSC and rsLTL. The formula $\mathsf{PC}(\overline{\mathbf{p}}^{par})$ applies only to the encoding of parameters and after performing the substitution we can simply assume that it is *true* as it does not constrain anything. The formula $[\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}, F, k]$ encodes $|F|$ bounded model checking instances (similar to what was described in Section 5.3). Let us consider $\phi_f \in F$. We assume an arbitrary $k \in \mathbb{N}$ and focus on the satisfiability of Paths^k_f, since the encoding of this formula differs from the corresponding one in the encoding for CR-$\mathcal{C}$. There exists a path $\sigma.f$ in $\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}$ and $(\sigma.f)^k$ is its prefix of length k iff there exists a valuation that represents $(\sigma.f)^k$ which satisfies Paths^k_f. Let $i \in \{0, \ldots, k-1\}$. We observe the formula $\mathsf{Tr}_{\text{CR-}\mathcal{CP}}$ is satisfied for the valuation encoding $(\sigma.f)_s(i)$, $(\sigma.f)_a(i)$ and $(\sigma.f)_s(i+1)$ iff $(\sigma.f)_s(i) \xrightarrow{(\sigma.f)_a(i)} (\sigma.f)_s(i+1)$. This follows from the encoding of $\mathsf{Tr}_{\mathcal{CP}}$ and $\mathsf{Tr}_{\mathfrak{A}}$. Let us first recall that $(\sigma.f)_s(i) = ((\sigma.f)_b(i), (\sigma.f)_{ca}(i))$. The formula $\mathsf{Tr}_{\mathcal{CP}}$ is satisfied iff the valuation satisfies $\mathsf{Results}$ and Rct_a for each $a \in A$. The formula Rct_a encodes the produced concentration levels for all the entities $e \in S$ by $a \in A$ using the intermediate variables of $\mathbf{P}^p_{f,i,a}$. Then, $\mathsf{Results}$ ensures the maximal produced concentration levels for each entity and each reaction are encoded using the variables of $\mathbf{P}_{f,i+1}$. It follows from the construction that $\mathsf{Results}$ and Rct_a are satisfied iff the valuation encodes the concentration levels of the entities in the successor state $(\sigma.f)_b(i+1)$ that are produced by the reactions from the state $(\sigma.f)_b(i)$ combined with the context $(\sigma.f)_a(i)$. The formula $\mathsf{Tr}_{\mathfrak{A}}$ is satisfied iff the valuation encodes a transition $(\mathfrak{q}, \mathbf{c}, \mathfrak{q}') \in R$ such that $\mathfrak{q} = (\sigma.f)_{ca}(i)$, $\mathbf{c} = (\sigma.f)_a(i)$ and $\mathfrak{q}' = (\sigma.f)_{ca}(i+1)$. Then, the formula Paths^k_f is satisfied iff the valuation encodes a path prefix $(\sigma.f)^k$ such that the state $(\sigma.f)_s(0)$ of $\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}$ is the initial state and $(\sigma.f)_s(i) \xrightarrow{(\sigma.f)_a(i)} (\sigma.f)_s(i+1)$ for all $i \in \{0, \ldots, k-1\}$. The rest of the proof for Loops^k_f and $[\![\phi_f]\!]^k_0$ follows as for Theorem 5.3.1.

Now it is easy to see that $\mathsf{Paths}^k_f \wedge \mathsf{Loops}^k_f \wedge [\![\phi_f]\!]^k_0$ is satisfiable iff $\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}} \models^k_\exists \phi_f$. Finally, we conclude that $[\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}, F, k]$ is satisfiable iff $\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}} \models^k_\exists \phi_f$ for all $\phi_f \in F$. □

In practice, the encoding $[\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}}, F, k]$ is intended to be satisfiable for any valid parameter valuation $\mathtt{v}$ such that $\bigwedge_{\phi \in F} \left(\mathcal{M}_{\text{CR-}\mathcal{CP}^{\leftarrow \mathtt{v}}} \models^{k}_{\exists} \phi\right)$. The constraints that enforce the valuation to be valid are expressed using the encoding of PC. We extract the valuation of the parameters of P when the formula (6.2) is satisfiable. For the satisfied formula we obtain its model, i.e., the valuations of the variables used in the formula. Let $V(\mathtt{p})$ denote the valuation of a variable $\mathtt{p}$ used in our encoding. The parameter valuations are defined as follows: $\lambda^{\leftarrow \mathtt{v}}(e) = V(\mathtt{pm}_{\lambda}(e))$ for each $e \in S$ and $\lambda \in P$.

6.3 Experimental evaluation

In this section we present the results of an experimental evaluation of the translation presented in Section 6.2. We test our method on a parametric version (PMUTEX) of the reaction system model for the mutual exclusion protocol introduced in Chapter 3. The system consists of $n \geq 2$ processes competing for an exclusive access to the critical section. The background set of the CRRSC modelling the mutual exclusion protocol is defined as $S = \bigcup_{i=1}^{n} S_i$, where the set of background entities corresponding to the i-th process is defined as:

$$S_i = \{out_i, req_i, in_i, act_i, lock, done, s\},$$

where the entities *lock*, *done*, and s are shared amongst all the processes.

We start by defining the context automaton $\mathfrak{A}$. Initially all the processes are outside of their critical sections and are not requesting access, which is indicated by the presence of out_i for each $i \in \{1, \ldots, n\}$. Next, we assume $\mathfrak{A}$ may supply any $C \subseteq \{act_1, ..., act_n\}$ such that $|C| \leq 2$, allowing at most two simultaneously active processes – we assume that if the context contains act_i then the i^{th} process is to perform an action. This leads to the following definition of the context automaton: $\mathfrak{A} = (\{\mathfrak{q}_0, \mathfrak{q}_1\}, \mathfrak{q}_0, R)$, where:

$$R = \{\mathfrak{q}_0 \xrightarrow{\{out_1, ..., out_n\}} \mathfrak{q}_1\} \cup \{\mathfrak{q}_1 \xrightarrow{C} \mathfrak{q}_1 \mid C \subseteq \{act_1, ..., act_n\} \text{ and } |C| \leq 2\}.$$

We allow only at most two active processes at a time to avoid encoding in the context automaton all the 2^n transitions with subsets of $\{act_1, ..., act_n\}$.

The i^{th} process requests 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, i.e., the *lock* entity is produced. 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.

This version of the mutual exclusion protocol implementation differs from the one presented in Chapter 3 by the use of concentration levels. Each process after requesting access to its critical section must wait at least one step before it is allowed to gain access and after entering the critical section, the process performs computations which take two steps.

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

- $(\{out_i \mapsto 1, act_i \mapsto 1\}, \{s \mapsto 1\}, \{req_i \mapsto 1\})$,
- $(\{out_i \mapsto 1\}, \{act_i \mapsto 1\}, \{out_i \mapsto 1\})$,
- $(\{req_i \mapsto 1, act_i \mapsto 1, act_j \mapsto 1\}, \{s \mapsto 1\}, \{req_i \mapsto 1\})$ for each $j \in \{1, \ldots, n\}$ s.t. $i \neq j$,
- $(\{req_i \mapsto 1\}, \{act_i \mapsto 1\}, \{req_i \mapsto 2\})$,
- $(\{req_i \mapsto 2, act_i \mapsto 1\}, \{act_j \mapsto 1 \mid j \in \{1, \ldots, n\} \text{ and } j \neq i\} \cup \{lock \mapsto 1\}, \{in_i \mapsto 3, lock \mapsto 1\})$,
- $(\{in_i \mapsto 3, act_i \mapsto 1\}, \{s \mapsto 1\}, \{in_i \mapsto 2\})$,
- $(\{in_i \mapsto 2, act_i \mapsto 1\}, \{s \mapsto 1\}, \{in_i \mapsto 1\})$,
- $(\{in_i \mapsto 1, act_i \mapsto 1\}, \{s \mapsto 1\}, \{out_i \mapsto 1, done \mapsto 1\})$,
- $(\{in_i \mapsto 1\}, \{s \mapsto 1\}, \{in_i \mapsto 1\})$.

Next, we assume here that the system is open and we allow for introducing new processes that participate in the communication to gain access to the critical section. Let us assume we are allowed to modify the behaviour of an additional process (the n^{th} process) only by introducing an additional reaction. Such an assumption could be justified by a mechanism that accepts new processes to participate in the protocol only if they contain the reactions of A_i for any $i \in \{1, \ldots, n\}$, while the remaining reactions could be performing some computation outside of the critical section.

Our aim is to violate the property of mutual exclusion by making the first and the n^{th} process enter their critical sections simultaneously. The additional (malicious) reaction uses the parameters of $P = \{\lambda_r, \lambda_i, \lambda_p\}$ and is defined as follows:

$$A_p = \{(\lambda_r, \lambda_i, \lambda_p)\}$$

The set of reactions is defined as:

$$A = \left(\bigcup_{i=1}^{n} A_i\right) \cup A_p \cup \big\{(\{lock \mapsto 1\}, \{done \mapsto 1\}, \{lock \mapsto 1\})\big\}.$$

Finally, we define the CRRSC modelling PMUTEX as: CR-$\mathcal{C}_M = ((S, P, A, \mathfrak{c}), \mathfrak{A})$, where:

$$\mathfrak{c} = \left((\lambda_p[in_n] = 0) \wedge \bigwedge_{\lambda \in P, e \in S \setminus S_n} (\lambda[e] = 0)\right).$$

The constraint $\mathfrak{c}$ constrains the additional reaction by requiring that it may produce only entities related to the n^{th} process and it cannot produce in_n. This is to avoid trivial solutions. Then, we need to synthesise a parameter valuation $\mathtt{v}$ of CR-$\mathcal{CP}_M$ which gives the rsLTL property $\phi = \mathsf{F}((in_1 > 0) \wedge (in_n > 0))$, i.e., $\mathcal{M}(\text{CR-}\mathcal{CP}_M^{\leftarrow \mathtt{v}}) \models_\exists \phi$.

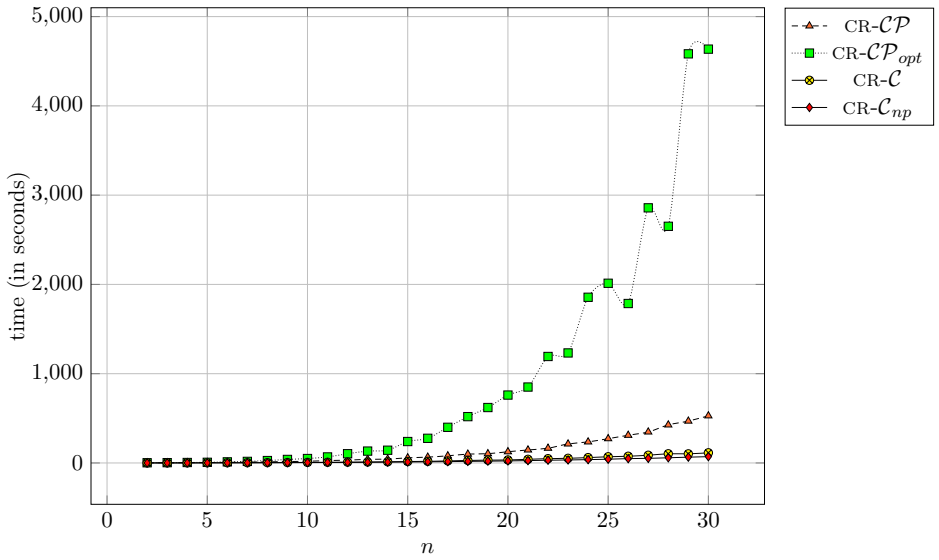

Figure 6.1: Synthesis results for PMUTEX: execution time

The verification tool that we use for this experiment was implemented in Python and it uses Z3 4.5.0 [de Moura and Bjørner, 2008] for SMT-solving. We implement an incremental approach, i.e., in a single SMT instance we increase the length of the encoded interactive processes by unrolling their encoding until witnesses for all the verified formulae are found. Then, the corresponding parameter valuation is extracted. The verification results[1] presented in Fig. 6.1–6.2 compare four approaches: the implementation of the encoding from Section 6.2 (CR-$\mathcal{CP}$) and its extension (CR-$\mathcal{CP}_{opt}$) that optimises the obtained parameter valuations by using OptSMT provided with Z3. We also use the same encoding for verification of the rsLTL property (CR-$\mathcal{C}$), i.e., we replace all the parameters with the obtained parameter valuations and test the formula ϕ in the same way as it is possible with the method defined in Chapter 5. Next, we compare our results with the ones obtained using the non-parametric method (CR-$\mathcal{C}_{np}$) defined in Chapter 5.

Our experimental implementation provides a valuation $\mathtt{v}$ which allows to violate the mutual exclusion property, where $\lambda_r^{\leftarrow \mathtt{v}} = \{out_n \mapsto 1\}$, $\lambda_i^{\leftarrow \mathtt{v}} = \{s \mapsto 1\}$, and $\lambda_p^{\leftarrow \mathtt{v}} = \{req_n \mapsto 2, done \mapsto 1\}$ for all the values $n \geq 2$ tested. This valuation was obtained using CR-$\mathcal{CP}_{opt}$.

[1]The experimental results were obtained using a system equipped with 3.7GHz Intel Xeon E5 processor and 12GB of memory, running Mac OS X 10.13.2.

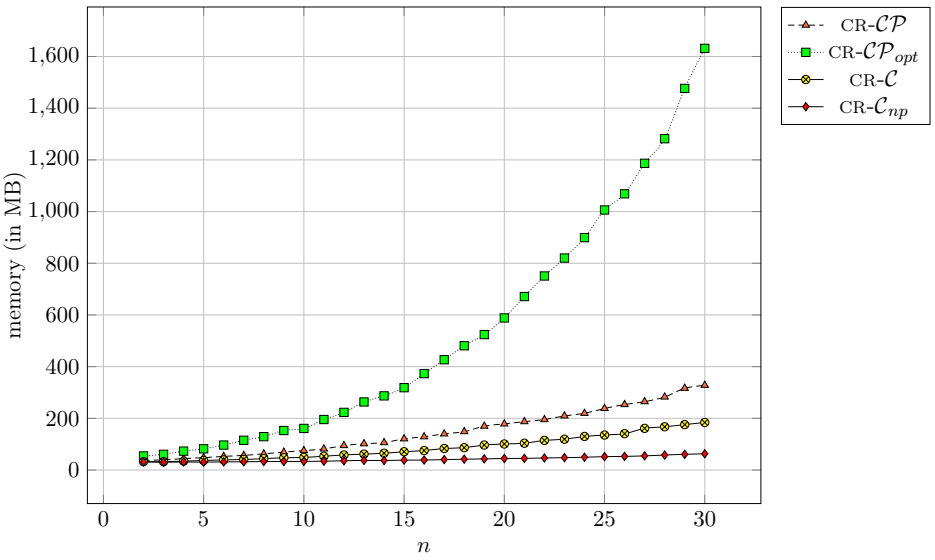

Figure 6.2: Synthesis results for PMUTEX: memory consumption

When using CR-$\mathcal{CP}_{opt}$, the memory consumption increases. However, the method might require less time to calculate the result than CR-$\mathcal{CP}$. The difference in time and memory consumption between the parametric (CR-$\mathcal{CP}$) and the non-parametric (CR-$\mathcal{C}$) approach is minor. However, CR-$\mathcal{C}_{np}$ is the most efficient of all the approaches tested. This suggests that our parameter synthesis method might possibly be improved by optimising the encoding used.

6.4 Concluding remarks

We have presented a method for reaction mining which allows for calculating parameter valuations for partially defined reactions of reaction systems. We also demonstrated how the presented method can be used for synthesis of an attack in which we inject an additional instruction represented by a reaction, where we use rsLTL to express the goal of the attack.

Assuming there is a finite set of allowed concentration levels for the parameters, the presented method also allows for enumerating all the possible parameter valuations for fixed-length processes. This can be achieved by adding an additional constraint blocking the parameter valuation obtained in the previous step.

When dealing with parameter synthesis, the parameters could be associated with the model [Alur *et al.*, 1993b, Hune *et al.*, 2002] or with the formalism used to express its properties [Knapik *et al.*, 2015, Jones *et al.*, 2012].

We focused on the synthesis of the parameters which appear in the reaction system. One could also consider extending this approach to include parameters in the context automaton. This could allow to synthesise the behaviour of the

environment which leads to satisfaction of the verified rsLTL property. However, in the implementation of our approach, when the verified formula is satisfied, we also obtain the witness which contains the entire context sequence generated by the context automaton. This sequence represents the behaviour of the environment which leads to satisfaction of the rsLTL formula.

Parameters could also be introduced in the rsCTL or rsLTL formulae to obtain parametric variants of these logics, e.g., by introducing parameters in place of the families of sets of entities (or the multiset expressions for rsLTL).

Chapter 7

Reaction systems model checking toolkit

This chapter presents an overview of the reaction systems model checking toolkit which has evolved from the experimental implementations presented in the previous chapters. The toolkit consists of two independent modules. This is the result of using two different underlying verification methodologies: ReactICS-BDD uses BDDs for storing an manipulating states of the verified systems, and ReactICS-SMT interacts with an SMT-solver to perform the verification.

7.1 ReactICS-BDD

ReactICS-BDD is implemented in C++. In this thesis the version 2.0 is described. In the previous versions of the tool, ICRRS and rsCTL were used as the input. However, since CRMARS and rsCTLK extend these in a conservative way, the current version no longer supports the input of ICRRS and rsCTL. The architecture of ReactICS-BDD is presented in Figure 7.1. The *parser* module parses a single input file containing a description of a CRMARS and a list of rsCTLK formulae. Each formula is labelled with an identifier that allows for it to be selected from the command line. The *parser* module uses an API providing all the methods required to build the data structures used for storing CRMARS and the rsCTLK formulae. The provided API could also be used to implement a graphical interface for the tool or an alternative parser. The data structure for CRMARS is held by the *RctSys* module, *CtxAut* is responsible for the operations on the context automaton, and *RSform* is used for the rsCTLK formulae. The Boolean functions used in the symbolic encoding in the form of BDDs are handled by *SymRS*, which is responsible for the symbolic representation of CRMARS. The *SymRS* module interacts with the CUDD library. Finally, the *MChecker* module implements all the model checking tasks and state space manipulation methods.

When the program is executed with the `-h` parameter it prints out a list of the available options. The output of the command is presented in Figure 7.4. The

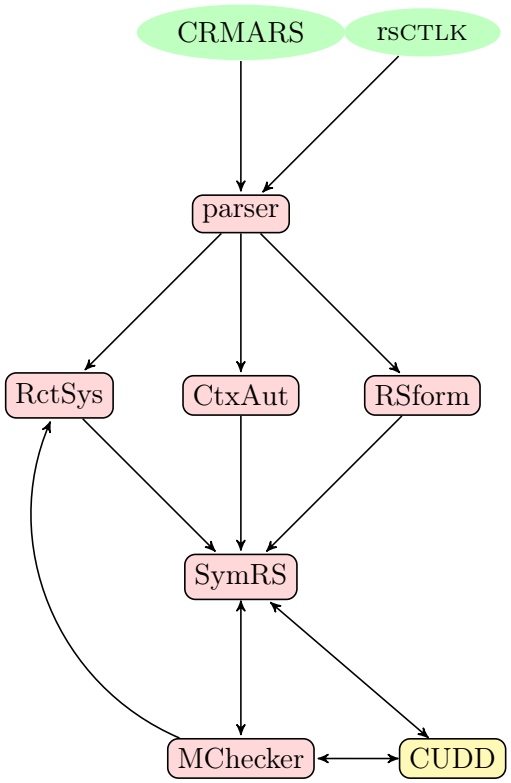

Figure 7.1: Architecture of ReactICS-BDD

options `-b`, `-x`, and `-z` were described in Section 3.7 and 4.7. The main feature of the tool is rsCTLK model checking and it is performed when `-c form` is provided, where `form` is the identifier of the formula to be verified. The reaction system and the formula to be verified need to be declared in the input file, using the specification language introduced in the following section.

7.1.1 Reaction systems specification language

We introduce here *Reaction Systems Specification Language* (RSSL), which is the input language of ReactICS-BDD. We assume *ident* to be any alphanumerical string that starts with a letter. Firstly, we introduce the basic expressions of the language:

$$
\begin{array}{lll}
\langle \textit{entity} \rangle & ::= & \langle \textit{ident} \rangle \\
\langle \textit{entities} \rangle & ::= & \langle \textit{entity} \rangle \text{ `,' } \langle \textit{entities} \rangle \\
 & \mid & \langle \textit{entity} \rangle \\
\langle \textit{entities-with-empty} \rangle & ::= & \epsilon \\
 & \mid & \langle \textit{entities} \rangle \\
\langle \textit{proc-ident} \rangle & ::= & \langle \textit{ident} \rangle
\end{array}
$$

⟨*proc-ent*⟩ ::= ⟨*proc-ident*⟩ '`.`' ⟨*entity*⟩

⟨*formula-ident*⟩ ::= ⟨*ident*⟩

The elements such as *entity*, *proc-ident* and *formula-ident* are defined to improve readability of the rules, but their definitions could be easily omitted and their occurrences replaced with *ident*. The root element of the grammar is *input*, and it is defined together with the main constructs of the language as follows:

⟨*input*⟩ ::= ϵ
| '`options`' '`{`' ⟨*options*⟩ '`}`' '`;`' ⟨*input*⟩
| '`reactions`' '`{`' ⟨*reactions-per-proc*⟩ '`}`' '`;`' ⟨*input*⟩
| '`context-automaton`' '`{`' ⟨*ctxaut*⟩ '`}`' '`;`' ⟨*input*⟩
| '`rsctlk-property`' '`{`' ⟨*formula-ident*⟩ '`:`' ⟨*rsctlk*⟩ '`}`' '`;`' ⟨*input*⟩

⟨*options*⟩ ::= ϵ
| ⟨*option*⟩ '`;`' ⟨*options*⟩

⟨*option*⟩ ::= '`make-progressive`'
| '`use-context-automaton`'

⟨*reactions-per-proc*⟩ ::= ϵ
| ⟨*process*⟩ '`;`' ⟨*reactions-per-proc*⟩

⟨*process*⟩ ::= ⟨*proc-ident*⟩ '`{`' ⟨*reactions*⟩ '`}`' '`;`'

⟨*reactions*⟩ ::= ϵ
| ⟨*reaction*⟩ '`;`' ⟨*reactions*⟩

⟨*reaction*⟩ ::= '`{`' ⟨*entities*⟩ '`,`' ⟨*entities-with-empty*⟩ '`->`' ⟨*entities*⟩ '`}`'

⟨*entities-with-empty*⟩ ::= ϵ
| ⟨*entities*⟩

Processes are synonymous with agents. The option `make-progressive` modifies the specified context automaton according to the construction presented in Chapter 4, to ensure it is progressive. Currently the option `use-context-automaton` is required for all the inputs. In the future versions we may allow for specifying different ways of controlling the contexts by introducing similar options. Context automata are specified using the *ctxaut* rule:

⟨*state*⟩ ::= ⟨*ident*⟩

⟨*states*⟩ ::= ⟨*state*⟩ '`,`' ⟨*states*⟩
| ⟨*state*⟩

⟨*proc-entities*⟩ ::= ⟨*proc-ident*⟩ '`=`' '`{`' ⟨*entities*⟩ '`}`'

⟨*entities-per-proc*⟩ ::= ⟨*proc-entities*⟩ ⟨*entities-per-proc*⟩
| ⟨*proc-entities*⟩

⟨*transition*⟩ ::= ‘`{`’ ⟨*entities-per-proc*⟩ ‘`}`’ ‘`:`’ ⟨*state*⟩ ‘`->`’ ⟨*state*⟩ ⟨*state-cond*⟩ ‘`;`’

⟨*transitions*⟩ ::= ϵ
| ⟨*transition*⟩ ‘`;`’ ⟨*transitions*⟩
| ⟨*transition*⟩

⟨*state-cond*⟩ ::= ϵ
| ‘`:`’ ⟨*state-constr*⟩

⟨*ctxaut*⟩ ::= ϵ
| ‘`states`’ ‘`{`’ ⟨*states*⟩ ‘`}`’ ‘`;`’ ⟨*ctxaut*⟩
| ‘`init-state`’ ‘`{`’ ⟨*state*⟩ ‘`}`’ ‘`;`’ ⟨*ctxaut*⟩
| ‘`transitions`’ ‘`{`’ ⟨*transitions*⟩ ‘`}`’ ‘`;`’ ⟨*ctxaut*⟩

For the state constraints we use the following grammar:

⟨*state-constr*⟩ ::= ⟨*proc-ent*⟩
| ‘`(`’ ⟨*state-constr*⟩ ‘`)`’
| ‘`NOT`’ ⟨*state-constr*⟩
| ⟨*state-constr*⟩ ‘`AND`’ ⟨*state-constr*⟩
| ⟨*state-constr*⟩ ‘`OR`’ ⟨*state-constr*⟩
| ⟨*state-constr*⟩ ‘`XOR`’ ⟨*state-constr*⟩

The formulae of rsCTLK are specified using the following grammar:

⟨*agent*⟩ ::= ⟨*proc-ident*⟩

⟨*agents*⟩ ::= ⟨*agent*⟩ ‘`,`’ ⟨*agents*⟩
| ⟨*agent*⟩

⟨*rsctlk*⟩ ::= ⟨*proc-ent*⟩
| ‘`(`’ ⟨*rsctlk*⟩ ‘`)`’
| ‘`NOT`’ ⟨*rsctlk*⟩
| ⟨*rsctlk*⟩ ‘`AND`’ ⟨*rsctlk*⟩
| ⟨*rsctlk*⟩ ‘`OR`’ ⟨*rsctlk*⟩
| ⟨*rsctlk*⟩ ‘`XOR`’ ⟨*rsctlk*⟩
| ⟨*rsctlk*⟩ ‘`IMPLIES`’ ⟨*rsctlk*⟩
| ‘`EX`’ ⟨*rsctlk*⟩
| ‘`EU`’ ‘`(`’ ⟨*rsctlk*⟩ ‘`,`’ ⟨*rsctlk*⟩ ‘`)`’
| ‘`EF`’ ⟨*rsctlk*⟩
| ‘`EG`’ ⟨*rsctlk*⟩
| ‘`E`’ ‘`<`’ ⟨*state-constr*⟩ ‘`>`’ ‘`X`’ ⟨*rsctlk*⟩
| ‘`E`’ ‘`<`’ ⟨*state-constr*⟩ ‘`>`’ ‘`U`’ ‘`(`’ ⟨*rsctlk*⟩ ‘`,`’ ⟨*rsctlk*⟩ ‘`)`’
| ‘`E`’ ‘`<`’ ⟨*state-constr*⟩ ‘`>`’ ‘`F`’ ⟨*rsctlk*⟩
| ‘`E`’ ‘`<`’ ⟨*state-constr*⟩ ‘`>`’ ‘`G`’ ⟨*rsctlk*⟩
| ‘`AX`’ ⟨*rsctlk*⟩
| ‘`AU`’ ‘`(`’ ⟨*rsctlk*⟩ ‘`,`’ ⟨*rsctlk*⟩ ‘`)`’

| 'AF' ⟨*rsctlk*⟩
| 'AG' ⟨*rsctlk*⟩
| 'A' '<' ⟨*state-constr*⟩ '>' 'X' ⟨*rsctlk*⟩
| 'A' '<' ⟨*state-constr*⟩ '>' 'U' '(' ⟨*rsctlk*⟩ ',' ⟨*rsctlk*⟩ ')'
| 'A' '<' ⟨*state-constr*⟩ '>' 'F' ⟨*rsctlk*⟩
| 'A' '<' ⟨*state-constr*⟩ '>' 'G' ⟨*rsctlk*⟩
| 'UK' '[' ⟨*agent*⟩ ']' '(' ⟨*rsctlk*⟩ ')'
| 'NK' '[' ⟨*agent*⟩ ']' '(' ⟨*rsctlk*⟩ ')'
| 'UE' '[' ⟨*agents*⟩ ']' '(' ⟨*rsctlk*⟩ ')'
| 'NE' '[' ⟨*agents*⟩ ']' '(' ⟨*rsctlk*⟩ ')'
| 'UC' '[' ⟨*agents*⟩ ']' '(' ⟨*rsctlk*⟩ ')'
| 'NC' '[' ⟨*agents*⟩ ']' '(' ⟨*rsctlk*⟩ ')' .

The grammar for rsCTLK is consistent with the one defined in Chapter 4. The semantics is straightforward but we use different tokens for the epistemic operators: the universal operators are prefixed with `U` and the existential operators with `N`, e.g., `UK` is K and `NK` is $\overline{\mathrm{K}}$. An example of an input file specified in RSSL is presented in Figure 7.2.

7.2 ReactICS-SMT

ReactICS-SMT is implemented in Python. The architecture of ReactICS-SMT is presented in Figure 7.5. This module does not have a traditional input file and the system together with its properties needs to be specified in Python. This requires interacting with the *RctSys*, *CtxAut*, and *FormLTL* modules directly to specify a model checking instance. The verification tasks are implemented by *SMTCheckerRS*, *SMTCheckerRSC*, and *SMTCheckerPRS*. The *SMTCheckerRS* implements a basic BMC method for CRRS using mostly Boolean variables in the encoding. The *SMTCheckerRSC* extends *SMTCheckerRS* by allowing for specifying concentrations, which are encoded using integer variables. Finally, the *SMTCheckerPRS* provides an extension allowing for CR-CPRS synthesis. When verifying rsLTL properties, the verification modules use *FormEnc* to obtain the (incremental) encodings for the verified rsLTL formula. All the verification modules and the *FormEnc* module interact with the Z3 SMT-solver. The SMT-solver used is Z3 [de Moura and Bjørner, 2008].

7.2.1 Interacting with ReactICS-SMT

Here we demonstrate how to use ReactICS-SMT with Python. We focus on parameter synthesis since it contains all the elements of the other types of implemented methods. Firstly, all of the required modules need to be imported. The Python code for the imports is presented in Listing 7.1. In our example we use the following system: $\mathcal{P} = (S, \{\lambda\}, A)$ where $S = \{a, b, c, h\}$ and $A = \{(\{a \mapsto 1\}, \{h \mapsto 1\}, \{b \mapsto 2\}), (\lambda, \{h \mapsto 1\}, \{c \mapsto 1\})\}$. The code that creates $\mathcal{P}$ is presented in Listing 7.2. The

```
options { use-context-automaton; make-progressive; };
reactions {

    proc0 {
        {{out}, {} -> {approach}};
        {{approach}, {req} -> {req}};
        {{allowed}, {} -> {in}};
        {{in}, {} -> {out,leave}};
        {{req}, {in} -> {req}};
    };

    proc1 {
        {{out}, {} -> {approach}};
        {{approach}, {req} -> {req}};
        {{allowed}, {} -> {in}};
        {{in}, {} -> {out,leave}};
        {{req}, {in} -> {req}};
    };
};

context-automaton {
    states { init, green, red };
    init-state { init };
    transitions {
        { proc0={out} proc1={out} }: init -> green;
        { proc0={allowed} }: green -> red : proc0.req;
        { proc1={allowed} }: green -> red : proc1.req;
        { proc0={} }: green -> green : ~proc0.req AND ~proc1.req;
        { proc1={} }: green -> green : ~proc0.req AND ~proc1.req;
        { proc0={} }: red -> green : proc0.leave;
        { proc1={} }: red -> green : proc1.leave;
        { proc0={} }: red -> red : ~proc0.leave AND ~proc1.leave;
        { proc1={} }: red -> red : ~proc0.leave AND ~proc1.leave;

    };
};

rsctlk-property { f1 : EF( E<proc0.allowed>X( proc0.in ) )
                       AND EF( E<proc1.allowed>X( proc1.in ) ) };

rsctlk-property { f2 : EF( proc0.approach AND proc1.approach ) };

rsctlk-property { f3 : AG( proc0.in IMPLIES K[proc0](~proc1.in) ) };

rsctlk-property { f4 : AG( proc0.in IMPLIES C[proc0,proc1](~proc1.in) ) };
```

Figure 7.2: Input file for ReactICS-BDD

```
% ./reactics -c f2 -v input.rs
Verbose level: 1
ii VERBOSE(1): reactics.cc (main:146): Parsing input.rs
ii VERBOSE(1): rsin_driver.cc (setupReactionSystem:90): Using RS with CA
ii VERBOSE(1): rsin_driver.cc (setupReactionSystem:91): Using ordinary RS
ii VERBOSE(1): symrs.cc (encode:53): Encoding...
ii VERBOSE(1): symrs.cc (initBDDvars:100): Initialising CUDD
ii VERBOSE(1): symrs.cc (initBDDvars:104): Preparing BDD variables
ii VERBOSE(1): symrs.cc (initBDDvars:295): All BDD variables ready
ii VERBOSE(1): symrs.cc (encodeCtxAutTrans:935): Encoding context automaton's transition relation
ii VERBOSE(1): symrs.cc (encodeTransitions:762): Decomposing reactions
ii VERBOSE(1): symrs.cc (encodeTransitions:770): Encoding reactions
ii VERBOSE(1): symrs.cc (encodeTransitions:785): Using monolithic transition relation encoding
ii VERBOSE(1): symrs.cc (encodeTransitions:820): Reactions ready
ii VERBOSE(1): symrs.cc (encodeTransitions:823): Augmenting transitions with transitions for context automaton
ii VERBOSE(1): symrs.cc (encodeInitStates:840): Encoding initial states (using context automaton)
ii VERBOSE(1): symrs.cc (getEncCtxAutInitState:926): Encoding context automaton's initial state
ii VERBOSE(1): symrs.cc (encodeInitStates:847): Initial states encoded
ii VERBOSE(1): symrs.cc (encode:77): Encoding done
Using BDD-based Bounded Model Checking
ii VERBOSE(1): mc.cc (checkRSCTLKbmc:623): Bounded model checking for RSCTLK formula: EF((proc0.approach AND proc1.approach))
ii VERBOSE(1): mc.cc (checkRSCTLKbmc:625): Processing the formula: encoding entities
ii VERBOSE(1): mc.cc (checkRSCTLKbmc:627): Entities encoded
ii VERBOSE(1): mc.cc (checkRSCTLKbmc:629): Processing the formula: encoding actions/contexts
ii VERBOSE(1): mc.cc (checkRSCTLKbmc:631): Contexts encoded
Formula EF((proc0.approach AND proc1.approach)) holds
```

Figure 7.3: ReactICS-BDD verification output

```
-----------------------------------------------
-- ReactICS -- Reaction Systems Model Checker --
-----------------------------------------------

 Version:   2.0
 Contact:   Artur Meski <meski@ipipan.waw.pl>

Usage: ./reactics [options] <input file>

TASKS:
-c form  -- perform RSCTLK model checking (form: formula identifier)
-P       -- print parsed system
-r       -- print reactions
-s       -- print all the reachable states
-t       -- print all the reachable states with their successors

OTHER:
-b       -- disable bounded model checking (BMC) heuristic
-x       -- use partitioned transition relation
-z       -- use reordering of the BDD variables
-v       -- verbose (use more than once to increase verbosity)
-p       -- show progress (where possible)

Benchmarking options:
-m       -- measure and display time and memory usage
-B       -- display an easy to parse summary (enables -m)
```

Figure 7.4: ReactICS-BDD output with usage information

`add_bg_set_entity` method is used to add the entities of S with their maximal concentration level. To create a parameter λ and get an object that can be used later on to define reactions using λ, we use the `get_param` method and save the returned object. The reactions are defined in a straightforward way using the `add_reaction` method. In Listing 7.3 we show how to represent the context automaton $\mathfrak{A} = (\{\mathfrak{q}_0, \mathfrak{q}_1\}, \mathfrak{q}_0, R)$ where $R = \{(\mathfrak{q}_0, \{a \mapsto 3\}, \mathfrak{q}_1), (\mathfrak{q}_1, \varnothing_S, \mathfrak{q}_1), (\mathfrak{q}_1, \{h \mapsto 1\}, \mathfrak{q}_1)\}$. In Listing 7.4, $\mathcal{P}$ and $\mathfrak{A}$ are used to create an instance of CR-$\mathcal{P} = (\mathcal{P}, \mathfrak{A})$. The method `show()` called on an instance of `ReactionSystemWithAutomaton` prints out the defined system to the standard output. The result of calling `show()` on the defined

```
from rs import *
from smt import *
from logics import *
from rsltl_shortcuts import *
```

Listing 7.1: ReactICS-SMT imports

```
prs = ReactionSystemWithConcentrationsParam()

ent_with_conc = [("a", 3), ("b",2), ("c",1), ("h",1)]

for ec in ent_with_conc:
    prs.add_bg_set_entity(ec)

lda = prs.get_param("lda")

prs.add_reaction([("a",1)],[("h",1)],[("b", 2)])
prs.add_reaction(lda,[("h",1)],[("c",1)])
```

Listing 7.2: Simple PRS in ReactICS-SMT

```
ca = ContextAutomatonWithConcentrations(prs)
ca.add_init_state("0")
ca.add_state("1")

ca.add_transition("0", [("a", 3)], "1")
ca.add_transition("1", [], "1")
ca.add_transition("1", [("h", 1)], "1")
```

Listing 7.3: Simple CA in ReactICS-SMT

```
crprs = ReactionSystemWithAutomaton(prs, ca)
crprs.show()
```

Listing 7.4: Introduction of CRPRS in ReactICS-SMT

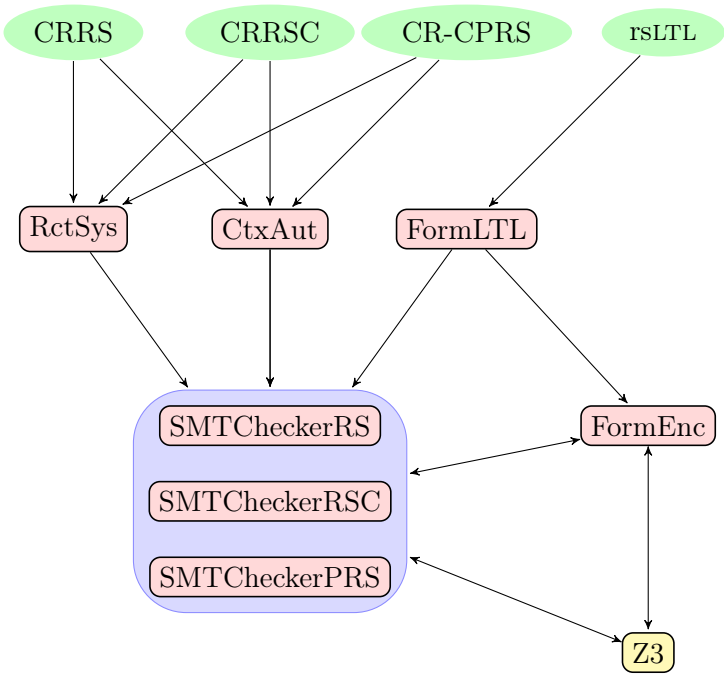

Figure 7.5: Architecture of ReactICS-SMT

```
f = ltl_F(bag_entity("h") == 0, "c")
pc = param_entity(lda, "a") == 0

checker = SmtCheckerRSCParam(crprs, optimise=True)
checker.check_rsltl(formulae_list=[f], param_constr=pc)
```

Listing 7.5: Verification instance in ReactICS-SMT (parameter synthesis)

system is presented in Figure 7.6. Finally, in Listing 7.5 we demonstrate how to verify $f = \mathsf{F}_{h=0}(c > 0)$ with a parameter constraint $\lambda[a] = 0$. To perform parameter synthesis we first create an instance of `SmtCheckerRSCParam`. The instance is initialised with the defined CRPRS and with `optimise` set to `True`, which enables OptSMT and results in minimisation of the synthesised parameter valuations. We initialise verification by calling `check_rsltl()`. The defined CR-$\mathcal{P}$ becomes a CR-CPRS when parameter constraints are provided to the `check_rsltl()` call. Therefore, `check_rsltl()` is invoked with two arguments: a list of rsLTL properties of the system and the parameter constraint. Here, the list of formulae to be verified contains only the formula f. The rsLTL formulae are constructed using the functions defined in the `rsltl_shortcuts.py` file included with the source code of ReactICS-SMT. The source code also includes examples of rsLTL formulae. The verification output for the specified system is presented in Figure 7.7.

```
[*] Background set: {a, b, c, h}
[*] Reactions:
                 reactants                      inhibitors           products
 -               { a=1 }                        { h=1 }             { b=2 }
 -              { @lda }                        { h=1 }             { c=1 }
[*] Permanent entities:
[*] Meta reactions:
[*] Maximal allowed concentration levels:
 -          a           = 3
 -          b           = 2
 -          c           = 1
 -          h           = 1
[*] Context automaton states:
 - 0 [init]
 - 1
[*] Context automaton transitions:
 - 0 --( { ('a', 3) } )--> 1
 - 1 --( 0 )--> 1
 - 1 --( { ('h', 1) } )--> 1
```

Figure 7.6: ReactICS-SMT: the output for the specified CRPRS

7.3 Final remarks

This chapter provided an overview of the reaction systems model checking toolkit. Two separate modules based on BDDs and SMT translations were described. More examples and details on the implementations can be obtained with the software package available at `http://arturmeski.github.io/reactics`.

```
[i] Running rsLTL bounded model checking
[i] Tested formulae:
[i]     F[h == 0](c > 0)
[i] INITIALISING...
[i] Preparing variables for path=0 (level=0)
[i] Concentration level assertions for path=0 (level=0)
[i] STARTING TO ITERATE...

------------------------[ Working at level=0 ]------------------------
[i] Generating the encoding for F[h == 0](c > 0) (1 of 1)

  [...]

------------------------[ Working at level=2 ]------------------------
[i] Generating the encoding for F[h == 0](c > 0) (1 of 1)
[i] Cache hits: 3, encode calls: 8 (approx: 3)
[i] Adding the formulae to the solver...
[i] Adding the encoding for the loops...
[i] Testing satisfiability...
[+] SAT at level=2

============================[ WITNESS ]=============================

  Witness for: F[h == 0](c > 0)

                                                            [ level=0 ]
  State: { }
  Context set: { a=3 }

                                                            [ level=1 ]
  State: { b=2 }
  Context set: { }

                                                            [ level=2 ]
  State: { c=1 }


  Parameters:

   lda: { b=1 }


[i]                                Time: 0.050708999999999976 s
[i]                                      Memory: 28.60546875 MB
```

Figure 7.7: Verification output for parameter synthesis in ReactICS-SMT

Chapter 8

Conclusions

In this chapter we summarise the results of this thesis and provide problems we intend to address in our future work.

8.1 Overview and summary

In this thesis, we have shown model checking is applicable to reaction systems.

We introduced context-restricted reaction systems that allow for specifying the environment in which the reaction system is functioning, rather than considering all possible context sequences generated by the background set of reaction system. The initial approach presented in Chapter 3 considers only a subset of the entities of the background set that are assumed to be relevant for the interactions of the system with its environment. Only these entities are used to generate the processes of the analysed reaction system. Since the properties of reaction systems depend heavily on their interactions with the environment, we introduced rsCTL. It is a temporal logic extending CTL and supporting the specification of context sets. We gave a Boolean encoding for the context-restricted reaction systems and used it to define a symbolic model checking method for checking rsCTL properties of reaction systems. The method uses binary decision diagrams that allow for efficient storage and manipulation of the state-space of the verified system. We also provided complexity analysis for rsCTL model checking and proved the problem is PSPACE-complete.

In [Azimi *et al.*, 2016] the authors defined several biologically-inspired properties of reaction systems together with the corresponding verification problems. We developed a logic that also allows to verify properties such as conserved sets (mass conservation), invariant sets, and steady states. A reaction systems model for the eukaryotic heat shock response mechanism was presented in [Azimi *et al.*, 2014a]. We expressed the properties specified in the paper using rsCTL, and automatically verified them using our implementation. There also exist other approaches to automated analysis of reaction systems such as [Azimi *et al.*, 2015a] or [Nobile *et*

al., 2017], but they are not exhaustive since they only allow for simulation of reaction systems. The convergence and occurrence properties [Salomaa, 2013a, Salomaa, 2013b, Formenti *et al.*, 2015] can also be specified in rsCTL using, respectively, the AX and EX operators.

We generalised reaction systems by introducing multi-agent reaction systems. This new formalism allowed for modelling of distributed and multi-agent systems while also being a conservative extension of ordinary reaction systems. We also introduced extended context automata which, in contrast to the previously defined context automata, make it possible to specify the behaviour of environment depending on the state of the multi-agent reaction system. This is achieved by specifying transition guards in the extended context automaton that restrict the set of states of the multi-agent reaction system in which the context associated with the transition is allowed to be provided. For dealing with a combination of temporal and epistemic properties of multi-agent reaction systems, we combined rsCTL with CTLK to define rsCTLK, which effectively extends rsCTL with epistemic operators. For the introduced formalism we provided a Boolean encoding and given a symbolic model checking method based on binary decision diagrams. We also demonstrated the model checking problem for rsCTLK to be PSPACE-complete. We applied the proposed method to verification of two scalable multi-agent systems with temporal-epistemic properties.

Bounded model checking [Biere *et al.*, 1999a] is one of the most successful verification methods applied to verification of real-world systems [Copty *et al.*, 2001]. To complement symbolic model checking based on binary decision diagrams, we provided bounded model checking for reaction systems based on a reduction to a variant of the Boolean satisfiability problem. We introduced rsLTL, which is a logic for expressing linear-time temporal properties of reaction systems. It is based on LTL, and similarly to rsCTL, it also allows for specifying restrictions of the permitted contexts. Additionally, we introduced reaction systems with discrete concentrations that allow for easier modelling of quantitative aspects of reaction systems. The proposed extension does not provide greater expressivity than the original formalism; however, it facilitates more efficient verification when quantitative aspects are relevant to the functioning of the system. Although there exist other approaches that support modelling of complex dependencies of concentration levels and their changes, e.g., chemical reaction networks theory based on [Horn and Jackson, 1972], reaction systems provide much simpler framework and the processes of reaction systems take into account interactions with the external environment. Reaction systems with durations studied in [Brijder *et al.*, 2011c] share some similarities with the formalism of reaction systems with discrete concentrations. However, in contrast to reaction systems with durations, the execution of reaction systems with discrete concentrations does not explicitly depend on a counter that can be implemented using reactions (see the translation into reaction systems presented in [Brijder *et al.*, 2011c]). The bounded model checking method was implemented by translating the problem into an SMT formula

Formalism	rsCTL/rsCTLK	rsLTL
RS	BDD-MC, BDD-BMC (Ch. 3), rsCTL only	SMT-BMC (Ch. 5)
MARS	BDD-MC, BDD-BMC (Ch. 4), rsCTLK	–
RSC	–	SMT-BMC (Ch. 5)
PRS	–	SMT-BMC (Ch. 6)

Table 8.1: Summary of the results

and then verifying its satisfiability using an SMT-solver. For this approach, we also showed how to generalise the notion of context-restricted reaction systems by introducing context automata for modelling the environment and more precise specification of the context sequences that the environment generates. We have also shown the rsLTL model checking problem to be PSPACE-complete.

In some practical applications, we may be presented with a physical system for which we might be unable to model all of its reactions fully. To address this, we extended our approach for bounded model checking of rsLTL to produce a method for reaction mining. We introduced a formalism for parametric reaction systems, where reactions can be defined partially by using parameters in place of the unknown reaction elements. The valuations of the parameters are synthesised given some, potentially empirically obtained, observations expressed using rsLTL. To this aim we allow for multiple formulae, where each formula specifies a separate observation related to a possible process or execution in the original system. This approach is implemented by translating the problem into an SMT formula and checking its satisfiability. The parameter valuations are extracted for a satisfiable SMT encoding instance. In the encoding, for each rsLTL formula we use a separate set of variables to represent a separate path in which the corresponding rsLTL formula must hold. However, the encoding of the parameters are shared amongst the encodings of the paths. This allows us to extract parameters that guarantee all of the specified rsLTL formulae hold. An application of the proposed method was used to synthesise an attack on a mutual exclusion protocol by calculating parameters of a malicious reaction to be injected into the system. The decision problem corresponding to the introduced parameter synthesis problem is proved to be PSPACE-complete.

For all the introduced verification methods, we provided implementations and evaluated them experimentally. This resulted in an implementation of a complete toolkit for verification of reaction systems. The provided implementations avoid representing state-spaces explicitly by using symbolic model checking techniques. This is particularly important given the complexity of the problems tackled in this thesis. The toolkit has been released online with its complete source code and is available for everyone to use and modify.

The verification methods proposed in this thesis are summarised in Table 8.1. Our BDD-based model checking methods do not support direct modelling of discrete concentrations. Moreover, in all of our BDD-based verification methods we also implement the BDD-based BMC heuristic [Copty *et al.*, 2001, Cabodi *et al.*, 2002].

8.2 Future work and other research problems

We now provide an overview of the problems that we intend to address in our future work.

Verification of universal properties

To verify rsLTL properties of reaction systems (Chapter 5), we proposed a method based on bounded model checking. Such an approach can provide a significant advantage when verifying existential properties because, it is sufficient to find a path prefix for which the verified property holds. This means that the verification can be performed on a fraction of the original model. However, the method is unable to disprove existential properties (or alternatively, to prove universal properties). Therefore, in our future work we plan to provide a complete method for the verification of rsLTL. The method proposed in this thesis can be used as a complete method provided it is known when the entire model has been explored. This can be achieved by establishing a method for computing the diameter (completeness threshold) [Clarke *et al.*, 2004] of the model and using it as the condition to stop model exploration. We also wish to propose a BDD-based model checking method for rsLTL, similar to the one proposed for rsCTL, which would be a complete method capable of verifying rsLTL formulae interpreted existentially and universally.

The parameter synthesis method (Chapter 6) focuses on existential observations that are very natural when obtained from simulations or experiments performed on the system, since it is problematic to derive conclusions about all the possible executions when investigating only a subset of them. However, when we consider some underlying physical properties of the system under investigation, these can be formulated as universal observations. Therefore, in our future work we are going to focus on the synthesis problem with universal observations. Since we use bounded model checking, if no valid parameter valuation exists and no bound on k is assumed, then our method does not terminate. As previously mentioned, this can be solved by computing the diameter of the verified model.

Generalised properties specification formalism: rsCTL*K

To allow for verification of a wider class of properties, we are also going to define rsLTLK: a logic that will extend rsLTL with epistemic operators, similarly to rsCTLK. To implement model checking for rsLTLK, an automata-based approach could be used, just like the one for LTLK presented in [Męski *et al.*, 2014b]. Combining symbolic model checking method for rsLTLK together with the one introduced

for rsCTLK (Chapter 4) would result in a verification method for rsCTL*K that would combine rsLTLK and rsCTLK similarly to how CTL* combines CTL with LTL.

The formalism of multi-agent 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 that allow for assigning integers to valuations should be used.

Timed reaction systems

Reaction systems with time were considered in [Ehrenfeucht and Rozenberg, 2009]. We would like to introduce a formalism for reaction systems with time constraints that would facilitate model checking of real-time systems in the reaction systems setting. Model checking for a discrete time modelling formalism could adopt methods of [Półrola *et al.*, 2014] and [Męski *et al.*, 2011b] for continuous time.

Specification patterns

For the model of eukaryotic heat shock response [Azimi *et al.*, 2014a] we provided its specification using rsCTL (Chapter 3). Different biologically-inspired properties of reaction systems studied in [Azimi *et al.*, 2016] can be specified using the temporal logic formalisms introduced in this thesis. Providing generalised mappings of properties of reaction systems into temporal formulae expressing them [Dwyer *et al.*, 1998] could help simplify the process of formalising behaviours of the analysed systems. These specification patterns could also be used to describe the observations in reaction synthesis (Chapter 6).

Efficiency improvements

Improvements of the encodings defined and used in Chapter 5 and 6 are potentially possible by using an SMT theory involving bit vectors or by basing the rsLTL encoding on one of the more involved and more efficient encodings for LTL presented in [Biere *et al.*, 2006]. In the approach for parameter synthesis we unroll all the path prefixes simultaneously and assume they are of the same length. This happens even if a witness for any of the verified formulae is found for a relatively short path prefix. The unwinding could be optimised to prevent this from happening.

8.3 Final remarks

The benefits of engineering new vaccines and drugs is undisputed, and is a clear demonstration of the applicability of synthetic biology. In such applications, development of methods for ensuring correctness of the results at the design stage is an important area of research. A paper published in *Nature* [Purnick and Weiss, 2009] presented an overview of the research results in synthetic biology. It highlighted the fact that the biological systems are being engineered to become

more and more complex, and it is impossible to use intuition alone to analyse combinations of even small systems. Consequently, developing efficient methods that support such analyses is crucial. While reaction systems are not mentioned in the paper it has been demonstrated by various researchers (see [Corolli *et al.*, 2012, Azimi *et al.*, 2014a, Azimi *et al.*, 2015b, Barbuti *et al.*, 2018]) that they can be used for biological modelling. Furthermore, the paper points out the applicability of model checking in identifying various parameters of these systems. For example, model checking could support identifying the mutations to perform, and could help to predict the behaviour after these perturbations have been made.

This thesis has introduced automated verification methods for reaction systems. These methods are based on model checking and allow for verification of temporal and epistemic properties of reaction systems. Additionally, extensions of reaction systems facilitating more efficient verification as well as modelling and verification of wider classes of systems were introduced. We also demonstrated how state-of-the-art methods and data structures such as SMT-solving and binary decision diagrams can be used for representing large state-spaces to support model checking for reaction systems. All of the proposed methods were implemented and resulted in introduction of a toolkit for verification of reaction systems, which is now available to the research community.

Bibliography

[Alhazov *et al.*, 2016] Artiom Alhazov, Bogdan Aman, Rudolf Freund, and Sergiu Ivanov. Simulating R systems by P systems. In *Membrane Computing, 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016*, pages 51–66, 2016.

[Alhazov, 2006] Artiom Alhazov. P systems without multiplicities of symbol-objects. *Information Processing Letters*, 100(3):124 – 129, 2006.

[Alur *et al.*, 1993a] R. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[Alur *et al.*, 1993b] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 592–601, 1993.

[Armando *et al.*, 2006] Alessandro Armando, Jacopo Mantovani, and Lorenzo Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. In *Model Checking Software, 13th International SPIN Workshop, Vienna, Austria, Proceedings*, pages 146–162. Springer Berlin Heidelberg, 2006.

[Azimi *et al.*, 2014a] Sepinoud Azimi, Bogdan Iancu, and Ion Petre. Reaction system models for the heat shock response. *Fundamenta Informaticae*, 131(3-4):299–312, 2014.

[Azimi *et al.*, 2014b] Sepinoud Azimi, Bogdan Iancu, and Ion Petre. Reaction system models for the heat shock response. *Fundam. Inf.*, 131(3-4):299–312, July 2014.

[Azimi *et al.*, 2015a] Sepinoud Azimi, Cristian Gratie, Sergiu Ivanov, and Ion Petre. Dependency graphs and mass conservation in reaction systems. *Theoretical Computer Science*, 598:23–39, 2015.

[Azimi *et al.*, 2015b] 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.

[Azimi *et al.*, 2016] Sepinoud Azimi, Cristian Gratie, Sergiu Ivanov, Luca Manzoni, Ion Petre, and Antonio E Porreca. Complexity of model checking for reaction systems. *Theoretical Computer Science*, 623:103–113, 2016.

[Azimi *et al.*, 2017] Sepinoud Azimi, Charmi Panchal, Andrzej Mizera, and Ion Petre. Multi-stability, limit cycles, and period-doubling bifurcation with reaction systems. *International Journal of Foundations of Computer Science*, 28(08):1007–1020, 2017.

[Azimi, 2017] Sepinoud Azimi. Steady states of constrained reaction systems. *Theoretical Computer Science*, 701:20–26, 2017.

[Baier and Katoen, 2008] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[Barbuti *et al.*, 2016a] Roberto Barbuti, Roberta Gori, Francesca Levi, and Paolo Milazzo. Investigating dynamic causalities in reaction systems. *Theoretical Computer Science*, 623:114–145, 2016.

[Barbuti *et al.*, 2016b] Roberto Barbuti, Roberta Gori, Francesca Levi, and Paolo Milazzo. Specialized predictor for reaction systems with context properties. *Fundamenta Informaticae*, 147(2-3):173–191, 2016.

[Barbuti *et al.*, 2017] Roberto Barbuti, Roberta Gori, Francesca Levi, and Paolo Milazzo. Generalized contexts for reaction systems: definition and study of dynamic causalities. *Acta Informatica*, pages 1–41, 2017.

[Barbuti *et al.*, 2018] Roberto Barbuti, Pasquale Bove, Roberta Gori, Francesca Levi, and Paolo Milazzo. Simulating gene regulatory networks using reaction systems. In *Proceedings of the 27th International Workshop on Concurrency, Specification and Programming, Berlin, Germany, September 24-26, 2018.*, 2018.

[Behrmann *et al.*, 2004] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[Bengtsson *et al.*, 1995] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.

[Biere *et al.*, 1999a] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using SAT

procedures instead of bdds. In *Proceedings of the 36th Conference on Design Automation, New Orleans, LA, USA, June 21-25, 1999.*, pages 317–320, 1999.

[Biere *et al.*, 1999b] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, pages 193–207. Springer-Verlag, 1999.

[Biere *et al.*, 2006] Armin Biere, Keijo Heljanko, Tommi A. Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5), 2006.

[Bottoni *et al.*, 2019] Paolo Bottoni, Anna Labella, and Grzegorz Rozenberg. Reaction systems with influence on environment. *Journal of Membrane Computing*, pages 1–17, 2019.

[Brijder *et al.*, 2010] Robert Brijder, Andrzej Ehrenfeucht, and Grzegorz Rozenberg. A note on causalities in reaction systems. *Electronic Communications of the EASST*, 30, 2010.

[Brijder *et al.*, 2011a] Robert Brijder, Andrzej Ehrenfeucht, Michael Main, and Grzegorz Rozenberg. A tour of reaction systems. *International Journal of Foundations of Computer Science*, 22(07):1499–1517, 2011.

[Brijder *et al.*, 2011b] Robert Brijder, Andrzej Ehrenfeucht, Michael Main, and Grzegorz Rozenberg. A tour of reaction systems. *International Journal of Foundations of Computer Science*, 22(07):1499–1517, 2011.

[Brijder *et al.*, 2011c] Robert Brijder, Andrzej Ehrenfeucht, and Grzegorz Rozenberg. Reaction systems with duration. In *Computation, Cooperation, and Life - Essays Dedicated to Gheorghe Paun on the Occasion of His 60th Birthday*, volume 6610 of *LNCS*, pages 191–202. Springer, 2011.

[Brijder *et al.*, 2012] Robert Brijder, Andrzej Ehrenfeucht, and Grzegorz Rozenberg. Representing reaction systems by trees. In *Computation, Physics and Beyond*, volume 7160 of *Lecture Notes in Computer Science*, pages 330–342, 2012.

[Brodo *et al.*, 2019] Linda Brodo, Roberto Bruni, and Moreno Falaschi. Enhancing reaction systems: a process algebraic approach, 2019.

[Bryant, 1986] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[Burch *et al.*, 1991] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. pages 49–58. North-Holland, 1991.

[Cabodi *et al.*, 2002] 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.

[Clarke and Emerson, 1981] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, pages 52–71, 1981.

[Clarke *et al.*, 1986] E. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[Clarke *et al.*, 1999] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Clarke *et al.*, 2004] Edmund M. Clarke, Daniel Kroening, Joël Ouaknine, and Ofer Strichman. Completeness and complexity of bounded model checking. In *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Proceedings*, pages 85–96, 2004.

[Copty *et al.*, 2001] Fady Copty, Limor Fix, Ranan Fraer, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Y. Vardi. Benefits of bounded model checking at an industrial setting. In *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, pages 436–453, 2001.

[Corolli *et al.*, 2012] 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.

[de Moura and Bjørner, 2008] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS 2008, pages 337–340. Springer-Verlang, 2008.

[Dennunzio *et al.*, 2014] 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.

[Dennunzio *et al.*, 2015a] Alberto Dennunzio, Enrico Formenti, and Luca Manzoni. Reaction systems and extremal combinatorics properties. *Theoretical Computer Science*, 598:138–149, 2015.

[Dennunzio *et al.*, 2015b] Alberto Dennunzio, Enrico Formenti, Luca Manzoni, and Antonio E Porreca. Ancestors, descendants, and gardens of eden in reaction systems. *Theoretical Computer Science*, 608:16–26, 2015.

[Dennunzio *et al.*, 2015c] Alberto Dennunzio, Enrico Formenti, Luca Manzoni, and Antonio E Porreca. Preimage problems for reaction systems. In *International Conference on Language and Automata Theory and Applications*, pages 537–548. Springer, 2015.

[Dennunzio *et al.*, 2016] Alberto Dennunzio, Enrico Formenti, Luca Manzoni, and Antonio E Porreca. Reachability in resource-bounded reaction systems. In *International Conference on Language and Automata Theory and Applications*, pages 592–602. Springer, 2016.

[Dwyer *et al.*, 1998] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Property specification patterns for finite-state verification. In *Proceedings of the Second Workshop on Formal Methods in Software Practice, March 4-5, 1998, Clearwater Beach, Florida, USA*, pages 7–15, 1998.

[Ehrenfeucht and Rozenberg, 2007a] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Events and modules in reaction systems. *Theoretical Computer Science*, 376(1-2):3–16, 2007.

[Ehrenfeucht and Rozenberg, 2007b] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Reaction systems. *Fundamenta Informaticae*, 75(1-4):263–280, 2007.

[Ehrenfeucht and Rozenberg, 2009] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Introducing time in reaction systems. *Theoretical Computer Science*, 410(4-5):310–322, 2009.

[Ehrenfeucht and Rozenberg, 2014] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Zoom structures and reaction systems yield exploration systems. *International Journal of Foundations of Computer Science*, 25(03):275–305, 2014.

[Ehrenfeucht and Rozenberg, 2015] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Standard and ordered zoom structures. *Theoretical Computer Science*, 608:4–15, 2015.

[Ehrenfeucht *et al.*, 2012a] A. Ehrenfeucht, J. Kleijn, M. Koutny, and G. Rozenberg. Reaction systems: A natural computing approach to the functioning of living cells. *A Computable Universe, Understanding and Exploring Nature as Computation (H. Zenil, ed.)*, pages 189–208, 2012.

[Ehrenfeucht *et al.*, 2012b] Andrzej Ehrenfeucht, Jetty Kleijn, Maciej Koutny, and Grzegorz Rozenberg. Minimal reaction systems. In *Transactions on Computational Systems Biology XIV*, pages 102–122. Springer, 2012.

[Ehrenfeucht *et al.*, 2017a] Andrzej Ehrenfeucht, Jetty Kleijn, Maciej Koutny, and Grzegorz Rozenberg. Evolving reaction systems. *Theoretical Computer Science*, 682:79–99, 2017.

[Ehrenfeucht *et al.*, 2017b] Andrzej Ehrenfeucht, Ion Petre, and Grzegorz Rozenberg. Reaction systems: A model of computation inspired by the functioning of the living cell. In *The Role of Theory in Computer Science: Essays Dedicated to Janusz Brzozowski*, pages 1–32. World Scientific, 2017.

[Emerson and Halpern, 1986] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

[Fagin *et al.*, 2003] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, Cambridge, MA, USA, 2003.

[Formenti *et al.*, 2014a] Enrico Formenti, Luca Manzoni, and Antonio E Porreca. Cycles and global attractors of reaction systems. In *International Workshop on Descriptional Complexity of Formal Systems*, pages 114–125. Springer, 2014.

[Formenti *et al.*, 2014b] Enrico Formenti, Luca Manzoni, and Antonio E Porreca. Fixed points and attractors of reaction systems. In *Conference on Computability in Europe*, pages 194–203. Springer, 2014.

[Formenti *et al.*, 2015] Enrico Formenti, Luca Manzoni, and Antonio E Porreca. On the complexity of occurrence and convergence problems in reaction systems. *Natural Computing*, 14(1):185–191, 2015.

[Grumberg and Veith, 2008] Orna Grumberg and Helmut Veith, editors. *25 Years of Model Checking - History, Achievements, Perspectives*, volume 5000 of *Lecture Notes in Computer Science*. Springer, 2008.

[Hirvensalo, 2012] Mika Hirvensalo. On probabilistic and quantum reaction systems. *Theoretical Computer Science*, 429:134–143, 2012.

[Horn and Jackson, 1972] F. Horn and R. Jackson. General mass action kinetics. *Archive for Rational Mechanics and Analysis*, 47(2):81–116, 1972.

[Hune *et al.*, 2002] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. *The Journal of Logic and Algebraic Programming*, 52-53:183 – 220, 2002.

[Huth and Ryan, 2004] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[Ivanov *et al.*, 2018] Sergiu Ivanov, Vladimir Rogojin, Sepinoud Azimi, and Ion Petre. WEBRSIM: A web-based reaction systems simulator. In *Enjoying Natural Computing - Essays Dedicated to Mario de Jesús Pérez-Jiménez on the Occasion of His 70th Birthday*, pages 170–181, 2018.

[Jamroga and Ågotnes, 2007] Wojciech Jamroga and Thomas Ågotnes. Modular interpreted systems. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 131:1–131:8, New York, NY, USA, 2007. ACM.

[Jamroga *et al.*, 2013] Wojciech Jamroga, Artur Męski, and Maciej Szreter. Modularity and openness in modeling multi-agent systems. In *Proceedings Fourth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2013, Borca di Cadore, Dolomites, Italy, 29-31th August 2013.*, pages 224–239, 2013.

[Jones and Lomuscio, 2010] Andrew V. Jones and Alessio Lomuscio. Distributed bdd-based BMC for the verification of multi-agent systems. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 675–682, 2010.

[Jones *et al.*, 2012] Andrew V. Jones, Michal Knapik, Wojciech Penczek, and Alessio Lomuscio. Group synthesis for parametric temporal-epistemic logic. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 1107–1114, 2012.

[Kauffman, 1969] S.A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437 – 467, 1969.

[Kleijn and Koutny, 2011] Jetty Kleijn and Maciej Koutny. Membrane systems with qualitative evolution rules. *Fundam. Inf.*, 110(1-4):217–230, January 2011.

[Kleijn *et al.*, 2018] Jetty Kleijn, Maciej Koutny, Lukasz Mikulski, and Grzegorz Rozenberg. Reaction systems, transition systems, and equivalences. In *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, pages 63–84, 2018.

[Knapik *et al.*, 2015] Michal Knapik, Artur Męski, and Wojciech Penczek. Action synthesis for branching time logic: Theory and applications. *ACM Trans. Embedded Comput. Syst.*, 14(4):64:1–64:23, 2015.

[Kroening and Strichman, 2016] Daniel Kroening and Ofer Strichman. *Decision Procedures - An Algorithmic Point of View, Second Edition.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[Laroussinie *et al.*, 2004] François Laroussinie, Nicolas Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In *CONCUR*, pages 387–401, 2004.

[Lichtenstein and Pnueli, 1985] Orna Lichtenstein and Amir Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings*

of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL '85, pages 97–107, New York, NY, USA, 1985. ACM.

[Lomuscio *et al.*, 2009] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. Mcmas: A model checker for the verification of multi-agent systems. In *Computer Aided Verification*, pages 682–688. Springer Berlin Heidelberg, 2009.

[McMillan, 1993] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[Męski *et al.*, 2014a] Artur Męski, Wojciech Penczek, and Grzegorz Rozenberg. Model checking temporal properties of reaction systems. Technical Report 1028, ICS PAS, April 2014.

[Męski *et al.*, 2014b] 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.

[Męski *et al.*, 2015] Artur Męski, Wojciech Penczek, and Grzegorz Rozenberg. Model checking temporal properties of reaction systems. *Information Sciences*, 313:22–42, 2015.

[Męski *et al.*, 2016] 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.

[Męski *et al.*, 2017] 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.

[Męski *et al.*, 2019] Artur Męski, Maciej Koutny, and Wojciech Penczek. Model checking for temporal-epistemic properties of distributed reaction systems. Technical Report CS-TR-1526, School of Computing, Newcastle University, Newcastle upon Tyne, UK, April 2019.

[Męski *et al.*, 2011a] Artur Męski, Wojciech Penczek, and Agata Półrola. BDD-based bounded model checking for temporal properties of 1-safe petri nets. *Fundamenta Informaticae*, 109(3):305–321, 2011.

[Męski *et al.*, 2011b] Artur Męski, Agata Półrola, Wojciech Penczek, Bozena Wozna-Szczesniak, and Andrzej Zbrzezny. Bounded model checking approaches for verification of distributed time petri nets. In *Proceedings of the International Workshop on Petri Nets and Software Engineering, Newcastle upon Tyne, UK, June 20-21, 2011*, pages 72–91, 2011.

[Męski *et al.*, 2018] 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.

[Nobile *et al.*, 2017] Marco S Nobile, Antonio E Porreca, Simone Spolaor, Luca Manzoni, Paolo Cazzaniga, Giancarlo Mauri, and Daniela Besozzi. Efficient simulation of reaction systems on graphics processing units. *Fundamenta Informaticae*, 154(1-4):307–321, 2017.

[Okubo and Yokomori, 2015] Fumiya Okubo and Takashi Yokomori. Recent developments on reaction automata theory: A survey. In *Recent Advances in Natural Computing*, pages 1–22. Springer, 2015.

[Okubo and Yokomori, 2016] Fumiya Okubo and Takashi Yokomori. The computational capability of chemical reaction automata. *Natural Computing*, 15(2):215–224, 2016.

[Okubo *et al.*, 2012a] Fumiya Okubo, Satoshi Kobayashi, and Takashi Yokomori. On the properties of language classes defined by bounded reaction automata. *Theoretical Computer Science*, 454:206–221, 2012.

[Okubo *et al.*, 2012b] Fumiya Okubo, Satoshi Kobayashi, and Takashi Yokomori. Reaction automata. *Theoretical Computer Science*, 429:247–257, 2012.

[Okubo, 2014] Fumiya Okubo. Reaction automata working in sequential manner. *RAIRO-Theoretical Informatics and Applications*, 48(1):23–38, 2014.

[Panda and Somenzi, 1995] Shipra Panda and Fabio Somenzi. Who are the variables in your neighborhood. In *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, 1995.

[Papadimitriou, 1994] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[Paun *et al.*, 2010] Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing*. Oxford University Press, Inc., New York, NY, USA, 2010.

[Paun, 2002] Gheorghe Paun. *Membrane computing: an introduction*. Springer-Verlag, Berlin Heidelberg New York, 2002.

[Pecheur and Raimondi, 2006a] C. Pecheur and F. Raimondi. Symbolic model checking of logics with actions. In *MoChArt*, pages 113–128, 2006.

[Pecheur and Raimondi, 2006b] Charles Pecheur and Franco Raimondi. Symbolic model checking of logics with actions. In *Model Checking and Artificial Intelligence, 4th Workshop, MoChArt IV, Riva del Garda, Italy, August 29, 2006, Revised Selected and Invited Papers*, pages 113–128, 2006.

[Peled, 1993] 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)*, volume 697 of *LNCS*, pages 409–423. Springer-Verlag, 1993.

[Penczek and Lomuscio, 2003] 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.

[Penczek and Półrola, 2006] Wojciech Penczek and Agata Półrola. *Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach*, volume 20 of *Studies in Computational Intelligence*. Springer, 2006.

[Półrola *et al.*, 2014] Agata Półrola, Piotr Cybula, and Artur Męski. SMT-based reachability checking for bounded time petri nets. *Fundam. Inform.*, 135(4):467–482, 2014.

[Păun and Rozenberg, 2002] Gheorghe Păun and Grzegorz Rozenberg. A guide to membrane computing. *Theor. Comput. Sci.*, 287(1):73–100, 2002.

[Purnick and Weiss, 2009] Priscilla E. M. Purnick and Ron Weiss. The second wave of synthetic biology: from modules to systems. *Nature reviews. Molecular cell biology*, 10(6), 2009.

[Queille and Sifakis, 1982] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, pages 337–351, 1982.

[Raimondi and Lomuscio, 2005] Franco Raimondi and Alessio Lomuscio. Symbolic model checking of multi-agent systems using OBDDs. In *In Proc. of the 3rd NASA Workshop on Formal Approaches to Agent-Based Systems (FAABS III), volume 3228 of LNCS*, pages 213–221. Springer-Verlag, 2005.

[Rudell, 1993] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '93, pages 42–47. IEEE Computer Society Press, 1993.

[Salomaa, 2012a] Arto Salomaa. Functions and sequences generated by reaction systems. *Theoretical Computer Science*, 466:87–96, 2012.

[Salomaa, 2012b] Arto Salomaa. On state sequences defined by reaction systems. In *Logic and Program Semantics*, pages 271–282. Springer, 2012.

[Salomaa, 2013a] Arto Salomaa. Functional constructions between reaction systems and propositional logic. *International Journal of Foundations of Computer Science*, 24(01):147–159, 2013.

[Salomaa, 2013b] Arto Salomaa. Minimal and almost minimal reaction systems. *Natural Computing*, 12(3):369–376, 2013.

[Salomaa, 2014] Arto Salomaa. Minimal reaction systems defining subset functions. In *Computing with New Resources*, pages 436–446. Springer, 2014.

[Salomaa, 2015] Arto Salomaa. Two-step simulations of reaction systems by minimal ones. *Acta Cybern.*, 22(2):247–257, 2015.

[Salomaa, 2017] Arto Salomaa. Minimal reaction systems: Duration and blips. *Theoretical Computer Science*, 682:208–216, 2017.

[Shang *et al.*, 2019] Zeyi Shang, Sergey Verlan, Ion Petre, and Gexiang Zhang. Reaction systems and synchronous digital circuits. *Molecules*, 24:1961, 05 2019.

[Shmulevich and Dougherty, 2010] Ilya Shmulevich and Edward R. Dougherty. *Probabilistic Boolean Networks - The Modeling and Control of Gene Regulatory Networks*. SIAM, 2010.

[Sneppen, 2014] Kim Sneppen. *Models of Life: Dynamics and Regulation in Biological Systems*. Cambridge University Press, 2014.

[Somenzi, 1994] Fabio Somenzi. Cudd: Colorado University decision diagram package. http://vlsi.colorado.edu/∼fabio/CUDD, 1994.

[Stockmeyer and Meyer, 1973] Larry J. Stockmeyer and Albert 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, USA*, pages 1–9. ACM, 1973.

[Tarski, 1955] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5(2):285–309, 1955.

[Teh and Atanasiu, 2017] Wen Chean Teh and Adrian Atanasiu. Minimal reaction systems revisited and reaction system rank. *International Journal of Foundations of Computer Science*, 28(03):247–261, 2017.

[van der Meyden and Wong, 2003] Ron van der Meyden and Ka-shu Wong. Complete axiomatizations for reasoning about knowledge and branching time. *Studia Logica*, 75(1):93–123, 2003.

[Voellmy and Boellmann, 2007] Richard Voellmy and Frank Boellmann. *Chaperone Regulation of the Heat Shock Protein Response*, pages 89–99. Springer New York, New York, NY, 2007.