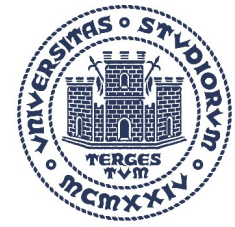


UNIVERSITÀ
DEGLI STUDI
DI TRIESTE

UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXVIII CICLO DEL DOTTORATO DI RICERCA IN

Scienze della terra, fluidodinamica e matematica.
Interazioni e metodiche

Complexity, Computability, and Cryptography with Reaction Systems

Settore scientifico-disciplinare: **INF/01**

DOTTORANDO
ROCCO ASCONE

COORDINATORE
PROF. STEFANO MASET

SUPERVISORE DI TESI
PROF. LUCA MANZONI

SUPERVISORE DI TESI
DOTT.SSA GIULIA BERNARDINI

ANNO ACCADEMICO 2024/2025

Università degli Studi di Trieste
Dipartimento di Matematica, Informatica e Geoscienze
Dottorato di Ricerca in Scienze della Terra, Fluidodinamica e Matematica.
Interazioni e Metodiche – Ciclo XXXVIII

Complexity Computability Criptography *with* Reaction Systems

ROCCO ASCONE

Settore scientifico-disciplinare: INF/01

Supervisors: Prof. Luca Manzoni and Dr. Giulia Bernardini
Coordinator: Prof. Stefano Maset

If people do not believe that mathematics is simple,
it is only because they do not realize how complicated life is.

– John von Neumann (1903 – 1957)

ACKNOWLEDGMENTS

I would first like to express my deepest gratitude to my supervisors, Giulia B. and Luca, for guiding me throughout this three-year journey. Their inspiration, encouragement, and constant support have been invaluable, and their mentorship has played a crucial role in my academic and personal growth.

I am sincerely grateful to all the colleagues with whom I have collaborated during these years: Francesco, Gloria, Enrico, Luca, Roberto, Nadia, Alessio, Massimo, Esteban, Veronica, and Giulia P. I also wish to thank all the people I met at conferences, who made those occasions such inspiring and enjoyable opportunities for collaboration.

My thanks go to my office mate and to everyone in the Economo building (formerly C5) for creating a pleasant and welcoming environment in which to work. In particular, I would like to thank Giulia M. (there have certainly been many Giulia to keep track of!), without whom I would probably never have enrolled in the PhD programme, nor successfully completed even half of my reimbursement forms. I am also deeply thankful to Gloria for her support and kindness during the past year, which have helped make challenging days lighter and brighter.

My gratitude extends to Umberto and Michele for their friendship and company in this distant place from which I am writing these words. I would also like to thank all UAI group for their warm hospitality and for making my stay in Santiago a truly special experience.

Academia means little without the support of one's family and friends. I am profoundly grateful to my parents and my brother for their endless patience and understanding throughout all my life. Finally, my heartfelt thanks go to Erica, for her constant support, patience, and the love she shares with me every day.

LIST OF PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

Journal Publications

1. R. Ascone, G. Bernardini, A. Conte, M. Equi, E. Gabory, R. Grossi, N. Pisanti.
   Pattern matching with Elastic-Degenerate strings and Elastic-Founder graphs.
   In: *Algorithms for Molecular Biology* (accepted).

2. R. Ascone, G. Bernardini, L. Manzoni.
   Cycles and global attractors of Reactantless and Inhibitorless Reaction Systems.
   In: *Theoretical Computer Science*, 2025.
   doi:10.1016/j.tcs.2025.115300.

3. R. Ascone, G. Bernardini, F. Leiter, L. Manzoni.
   Chemical Pure Reaction Automata in Maximally Parallel Manner.
   In: *Journal of Membrane Computing*, 2025.
   doi:10.1007/s41965-024-00176-7

4. R. Ascone, G. Bernardini, E. Formenti, F. Leiter, L. Manzoni.
   Pure Reaction Automata.
   In: *Natural Computing*, 2024.
   doi:10.1007/S11047-024-09980-7

5. R. Ascone, G. Bernardini, L. Manzoni.
   Fixed points and attractors of Additive Reaction Systems.
   In: *Natural Computing*, 2024.
   doi:10.1007/s11047-024-09977-2

6. R. Ascone, G. Bernardini, L. Manzoni.
   Fixed points and attractors of Reactantless and Inhibitorless Reaction Systems.
   In: *Theoretical Computer Science*, 2024.
   doi:10.1016/j.tcs.2023.114322

Conference Publications

1. R. Ascone, L. Mariot, L. Manzoni, G. Pietropolli.
   Evolving Cryptographic Boolean Functions with Reaction Systems.
   In: *The Genetic and Evolutionary Computation Conference* (short paper at **GECCO 2025**).
   doi:10.1145/3712255.3726685

2. R. Ascone, G. Bernardini, A. Conte, M. Equi, E. Gabory, R. Grossi, N. Pisanti.
   A unifying taxonomy of pattern matching in Degenerate Strings and Founder Graphs.
   In: *24th International Workshop on Algorithms in Bioinformatics* (**WABI 2024**).
   doi:10.4230/LIPIcs.WABI.2024.14

3. R. Ascone, G. Bernardini, F. Leiter, L. Manzoni.
   Chemical Pure Reaction Automata in Maximally Parallel Manner.
   In: *25th Conference on Membrane Computing* (**CMC 2024**).

Submitted

1. R. Ascone, G. Bernardini, A. Conte, V. Guerrini, G. Punzi
   Are Depth-2 Regular Expressions Hard to Intersect?
   Submitted to: *Journal of Computer and System Sciences* (2025).
   doi:10.48550/arXiv.2507.03593

2. R. Ascone, G. Bernardini, L. Manzoni, G. Pietropolli.
   A Novel Bio-Inspired Encoding for Evolving Cryptographic Boolean Functions.
   Submitted to: *Swarm and Evolutionary Computation* (2025).

CONTENTS

1

INTRODUCTION

Nature has always constituted a fundamental reference point for human existence, and numerous scientific disciplines have sought to draw inspiration from it. In contrast, computers and modern technology are often perceived as distant from natural phenomena. Nevertheless, the study of natural processes offers valuable insights that can inform the design of computational models.

This thesis adopts the framework of Natural Computing, an area of research concerned with computation inspired by mechanisms observed in nature. In particular, we focus on Reaction Systems, a bio-inspired computational model introduced nearly two decades ago. Our analysis addresses this model from the perspectives of computational Complexity (Part I), Computability (Part II), and Cryptography (Part III).

Parts I to III constitute the main core of the thesis devoted to Reaction Systems, while Part V presents the work done in a second line of research on Fine-Grained Complexity applied to pattern matching on degenerate strings and intersection of regular expressions.

1.1 PART I

*Reaction System*s (RSs) are a growing and now established computational model, introduced by A. Ehrenfeucht and G. Rozenberg [107, 108], that take inspiration from the chemical reactions occurring inside living cells. A set of *entities* or chemical species is transformed by one or more reactions and, like in real life, a reaction for which all *reactants* are present and all *inhibitors* are absent will generate all the expected *products*. RSs have some characteristics that distinguish them from other bio-inspired models. First of all, there is no permanence: an entity that is not used by any reaction will not remain in the system, but it will disappear. Second, RSs operate on a qualitative basis, meaning that the presence of a reactant in a given state implies that it is available in sufficient quantity for all reactions that require it, thus avoiding conflicts over shared resources. Other related models waiving this assumption have been proposed in the literature, see e.g. [12, 18, 22, 64, 170, 194, 210, 211, 213, 261]. Nevertheless, the computational power of the simpler qualitative model has been demonstrated by several studies [8, 27, 38, 39, 84, 160, 158] showing that RSs can be effectively used to simulate various biological processes. Concerning theoretical aspects, their properties have been studied from the point of view of the complexity of the dynamics [19, 20, 23, 35, 111, 123, 124, 125, 148, 253]; from the causality perspective [34, 36, 58]; by introducing in a natural way additional restrictions and

extensions [26, 60, 56, 183, 241]; by classifying and simulating them [181, 251, 252]; and by relating them with other computational models [106, 166, 218].

Although the conventional framework for RSs does not limit the number of reactants and inhibitors involved in each reaction, an alternative branch of research concentrates on systems with constrained resources. Ehrenfeucht et al. [110] first investigated how bounding the number of reactants and inhibitors in the reactions can affect the kinds of functions that a RS can define. Manzoni et al. [181] then classified resource-bounded systems in such a way that the reaction functions enjoy specific properties within each class: in particular, they identified the class of *inhibitorless* RSs, in which all reactions have an empty set of inhibitors; the class of *reactantless* RSs, in which the set of reactants is always empty; and the class of RSs, later named *additive* [19], in which each reaction only uses at most one reactant and no inhibitors. Dennunzio et al. [100] studied the complexity of reachability in several subclasses of inhibitorless and reactantless systems; Azimi et al. [26] studied how to list all steady states of a system whose reactions have a small quantity of both reactants and inhibitors; Teh et al. [253] studied the evolvability problem in reactantless and inhibitorless systems. Chapters 3 to 5 in Part 1 of this thesis are based on [19, 20, 23] where we investigate the computational complexity of various problems related to fixed points, cycles and attractors in reactantless, inhibitorless and additive RSs.

Table 1 summarizes the problems studied in Part 1, and we can notice that, despite these restrictions on the resources, the complexity of the considered problems is not necessarily reduced. For example, deciding whether a state is part of a cycle is **PSPACE**-complete in the general case, and it remains so in both the resource-bounded classes (Corollary 5.4.3 and Theorem 5.2.7). This is not always the case: for instance, deciding whether two RSs have the same result function is **coNP**-complete in the unconstrained case, while it is in **P** for reactantless and inhibitorless RSs (Corollary 3.5.6 and Corollary 3.5.4). Furthermore, there exist problems for which the complexity for reactantless RSs is not always the same as for inhibitorless RSs: e.g., deciding on the existence of a fixed point is in **P** for inhibitorless RSs [139], whereas it is an **NP**-complete problem in reactantless RSs (Theorem 3.3.1). Finally, we show that some particular dynamical behaviours can not be obtained, e.g. no global 3-cycle attractor (like the one in Figure 8 in Chapter 2) can exist in the dynamics of a reactantless RSs or in the dynamics of an inhibitorless RSs (Proposition 5.3.5 and Lemma 5.1.5).

Most of the reductions in this part of the thesis are from the most fundamental problems in the corresponding complexity class (e.g. SAT for **NP**-hardness and TAUTOLOGY for **coNP**-hardness), even though we also show non-trivial reductions between different problems and classes of RSs (see Theorem 5.3.6).

Furthermore, for the class of additive RSs, we prove that all the problems considered in Chapter 3 are in **P**. The proofs rely on a reduction to a graph representation of RS called the *influence graph*. As a future research direction, it would be interesting to use the influence graph to decide the complexity of the problems regarding cycles that are left open in Chapter 5. Regarding fixed points, it remains an open problem to figure out the complexity of deciding whether there exists a fixed point attractor for inhibitorless RSs.

Problem		$\mathcal{RS}(\infty,\infty)$	$\mathcal{RS}(0,\infty)$	$\mathcal{RS}(\infty,0)$	$\mathcal{RS}(1,0)$
A given state is a fixed point attractor		**NP**-c [123]	**NP**-c [20]	**NP**-c [20]	**P** [19]
$\exists$ fixed point		**NP**-c [123]	**NP**-c [20]	**P** [139]	
$\exists$ common fixed point		**NP**-c [123]	**NP**-c [20]	**NP**-c [20]	**P** [19]
sharing all fixed points		**coNP**-c [123]	**coNP**-c [20]	**coNP**-c [20]	**P** [19]
$\exists$ fixed point attractor		**NP**-c [123]	**NP**-c [20]	Open	**P** [19]
$\exists$ common fixed point attractor		**NP**-c [123]	**NP**-c [20]	**NP**-c [20]	**P** [19]
sharing all fixed points attractors		$\Pi_2^{\mathbf{P}}$-c [123]	$\Pi_2^{\mathbf{P}}$-c [20]	$\Pi_2^{\mathbf{P}}$-c [20]	**P** [19]
$\exists$ fixed point not attractor		$\Sigma_2^{\mathbf{P}}$-c [20]	$\Sigma_2^{\mathbf{P}}$-c [20]	$\Sigma_2^{\mathbf{P}}$-c [20]	**P** [19]
$\exists$ common fixed point not attractor		$\Sigma_2^{\mathbf{P}}$-c [20]	$\Sigma_2^{\mathbf{P}}$-c [20]	$\Sigma_2^{\mathbf{P}}$-c [20]	**P** [19]
sharing all fixed points not attractors		**coNP**-c [20]	**coNP**-c [20]	**coNP**-c [20]	**P** [19]
$\mathrm{res}_{\mathcal{A}} = \mathrm{res}_{\mathcal{B}}$		**coNP**-c [20]	**P** [20]	**P** [20]	
res bijective		**coNP**-c [124]	**P** [20]	**P** [20]	
A given state is a global attractor		**PSPACE**-c [124]	**P** [23]	**P** [23]	
$\exists$ global fixed point attractor		**PSPACE**-c [124]	**P** [23]	**P** [23]	
$\exists$ global cycle attractor of length at least k	$k = 2$	**PSPACE**-c [101]	**PSPACE**-c [23]	$\nexists$ [23]	
	$k > 2$	**PSPACE**-c [101]	$\nexists$ [23]	$\nexists$ [23]	
A given state is part of a cycle		**PSPACE**-c [124]	**PSPACE**-c [23]	**PSPACE**-c [23]	Open
$\exists$ common cycle		**PSPACE**-c [124]	**PSPACE**-c [23]	**PSPACE**-c [23]	Open
sharing all cycles		**PSPACE**-c [124]	**PSPACE**-c [23]	**PSPACE**-c [23]	Open
reachability		**PSPACE**-c [100]	**PSPACE**-c [100]	**PSPACE**-c [100]	Open

Table 1. Computational complexity of the problems studied in Part I of this thesis for different classes of RSs. **NP**-c, **coNP**-c, $\Sigma_2^{\mathbf{P}}$-c, $\Pi_2^{\mathbf{P}}$-c and **PSPACE**-c are shorthands for **NP**-complete, **coNP**-complete, $\Sigma_2^{\mathbf{P}}$-complete, $\Pi_2^{\mathbf{P}}$-complete and **PSPACE**-complete, respectively; $\mathcal{RS}(\infty,\infty)$, $\mathcal{RS}(0,\infty)$, $\mathcal{RS}(\infty,0)$ and $\mathcal{RS}(1,0)$ denote unconstrained, reactantless, inhibitorless and additive RSs, respectively (see Definition 2.1.1 for a formal declaration). The highlighted cells contain the original results from the papers presented in Part I.

1.2 PART II

Since the states of a RS are subsets of a finite set of entities, their dynamics can only contain a finite number of distinct configurations, making computational universality unattainable for them. A natural way to preserve the fundamentals of RSs, while making them a universal computational model, is to allow multiplicity for the entities. More than a decade ago, the first significant step in this direction was taken by Okubo et al. [211] with the introduction of Reaction Automata (RA) and with the successive streamlining of studies on the topic, see [207, 208, 211, 212, 261]. In RA, a state is a *multiset* of entities (thus allowing an infinite set of possible states), and reactions are modified to require multisets of reactants and products. A reaction can happen in a state only if the multiset of reactants is contained in the state and none of the inhibitors has nonzero multiplicity: since several reactions can compete for the resources of a state, their relative dominance must be established through predetermined criteria known as *manners*. RA are further specified by a fixed initial

state, receive input words over a fixed alphabet, and can accept such words based on a given set of final states.

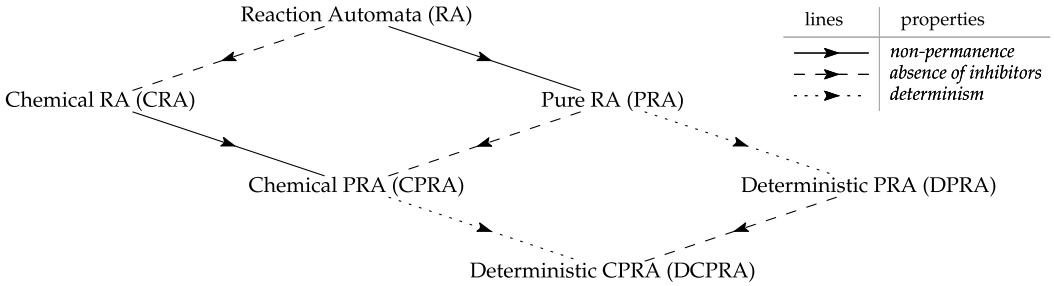


Figure 1. Summary of the properties among the types of automata studied in Part II. The arrow between RA and Pure Reaction Automata (PRA) should be interpreted as "PRA are RA without permanence"; and similarly in the other cases. The three different types of lines distinguish the three different properties: non-permanence (solid line), absence of inhibitors (dashed line) and determinism (dotted line).

Chapter 6 is based on [18], where we continue the study of RA by providing several new results and connections to other models. First of all, we define the *maximally reactive* (mr) manner: a different criterion w.r.t. the standard *maximally parallel* (mp) manner for selecting the competing reactions. We also introduce a new kind of RA, called Pure Reaction Automata (PRA). They differ from those introduced in [261] in how the outcome of a reaction is defined: while classical RA have permanence (i.e. the entities that are not used are preserved), in PRA the entities that are not used by any reaction are lost. This makes PRA more similar, in this aspect, to RSs, where non-permanence is a defining characteristic. The main results of Chapter 6 are: PRA working in mr manner accept any recursively enumerable language; and the subset of Deterministic Pure Reaction Automata (DPRA) working in mr manner can compute any recursive function. Future work includes clarifying how different update manners affect the computational power of RAs, both pure and non-pure; and exploring the role of determinism under time and space constraints.

Whenever a new Turing-complete computational model is introduced, a natural question arises: how much can we restrict the model while keeping universality unchanged? Chapter 7 aims towards an answer: in a similar fashion as in Part I, we study the class of Chemical Pure Reaction Automata (CPRA), which is the *inhibitorless* subclass of PRA (i.e. no inhibitors are present in the reactions). The non-pure counterpart for CPRA are Chemical Reaction Automata (CRA), the class of inhibitorless RA with permanence that have been introduced and studied in [209, 210, 213]. The class of CPRA thus combines the characterising properties of CRA and PRA. See Figure 1 for a visual representation of the relations between the classes of reaction automata considered in Part II. Our main results in Chapter 7 are: CPRA working in mp manner accept any recursively enumerable language; instead, in the deterministic setting, Deterministic Chemical Pure Reaction Automata (DCPRA) are not Turing complete as language acceptors. The proof of the latter result relies on Dickson's Lemma, a standard result in commutative algebra [85] and the fact that the result function of CPRA is monotonic (as for inhibitorless RS [181]). In particular, Dickson's Lemma is used to prove that the space of multisets with a fixed topology is compact. Although the proof is non-standard and interesting *per se*, it remains open which re-

lation exists between the class of languages recognised by DCPRA and the Chomsky hierarchy. Figure 2 summarizes the results obtained in Part II.

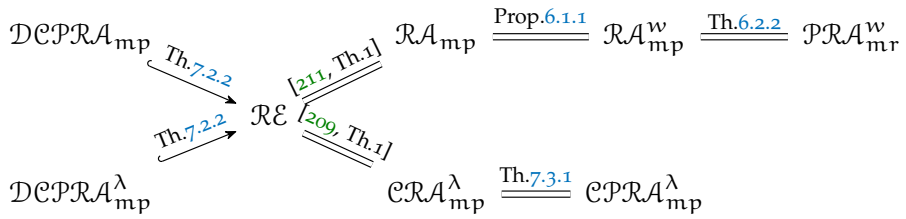


Figure 2. Summary of the results in Part II. The arrow $\hookrightarrow$ indicates strict inclusion between the sets of languages. $\mathcal{RE}$ represents the set of recursively enumerable languages. $\mathcal{DCPRA}_{mp}$, $\mathcal{CPRA}_{mp}$, $\mathcal{CRA}_{mp}$, $\mathcal{RA}_{mp}$ are the sets of languages recognized by DCPRA, CPRA, CRA, and RA, respectively, working in maximally parallel mp manner. $\mathcal{PRA}_{mr}$ is the set of languages recognized by PRA working in maximally reactive mr manner. The apex w means weakly accepted, and the apex λ means accepted in λ-input mode (see Definitions 6.1.5 and 7.1.8).

1.3 PART III

In Part III, we aim to build a connection between RSs and cryptography. By their intrinsic nature, RSs can encode Boolean functions in Disjunctive Normal Form (DNF) [136]: we will call Boolean Reaction Systems (BRSs) the RSs encoding Boolean functions. Even though the DNF encoding of Boolean functions is one of the standard encodings for Boolean logic formulae, together with the Conjunctive Normal Form (CNF) and the Algebraic Normal Form (ANF), in the literature, this encoding has not been used for the representation of Boolean functions satisfying strong cryptographic properties. The aim of Chapter 8 is to discover some properties of the DNF in relation to some cryptographic tools, such as the Walsh transform, that lead to a lower bound on the number of variables a clause must contain in the DNF of functions which are highly nonlinear, even with the balancedness constraint.

The nonlinearity of a Boolean function measures the Hamming distance of the function from the set of all linear functions; furthermore, a function is balanced whenever the function's truth table has an equal number of ones and zeros. These two properties are fundamental for the design of hard-to-decrypt block and stream ciphers, and finding functions with these properties is a well-established line of research [102]. Since the search space grows super-exponentially in the number of inputs, the problem becomes a hard optimization problem. Three main approaches are used in the literature: algebraic constructions, random generation, and (meta-) heuristic methods. Random generation is impractical due to the prohibitive size of the search space. In contrast, algebraic constructions scale to many different function sizes, but they produce the same Boolean functions when using the same starting conditions, and finding new constructions can be far from trivial. Within metaheuristic strategies, Evolutionary Computation (EC) has proven to be highly effective [223], primarily through Genetic Algorithms (GA) [169] and Genetic Programming (GP) [171]. A major limitation of existing EC methods lies in their function encoding: GA methods rely on bitstring encoding of the function's truth table, which

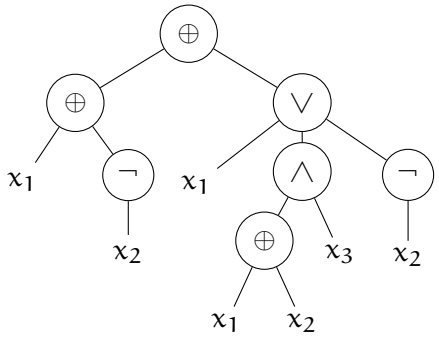
Strategy	Encoding
Genetic Programming	
Genetic Algorithm	$[0,0,0,1,1,1,0,0]$
Evolutionary Boolean Reaction System	$(\{x_1\},\{x_2\},\{\text{True}\})$ $(\{x_2,x_3\},\{x_1\},\{\text{True}\})$

Table 2. Three different encodings for the Boolean function $f : \mathbb{F}_2^3 \to \mathbb{F}_2$ given by $f(x) = (x_1 \oplus \neg x_2) \oplus (x_1 \vee ((x_1 \oplus x_2) \wedge x_3) \vee \neg x_2)$; 00011100 is the bitstring (of length 2^3) representing its truth table. In DNF, f can be written as $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2 \wedge x_3)$; the corresponding BRS is given by the reactions $(\{x_1\},\{x_2\},\{\text{True}\})$, $(\{x_2,x_3\},\{x_1\},\{\text{True}\})$. The DNF of any function f can also be represented via a tree with the root labeled by $\vee$, and its sons always labeled by $\wedge$ (see Figure 27).

grows exponentially with the number of variables, whereas GP methods employ complex operator trees that are difficult to interpret (see Table 2 for an illustration).

In Chapter 9, we overcome these limitations by using an encoding based on DNF and RSs, since the DNF of a Boolean function can be bounded to polynomial size and is easily interpretable. The first work on RS-based evolutionary strategies was presented in [180]. Inspired by this work, we introduce Evolutionary Boolean Reaction Systems (EvoBRS), an original evolutionary framework specifically designed to evolve individuals represented through BRS. The design of its key components is guided by the analysis described in Chapter 8, whose results inform the definition of the EvoBRS operators. Our analysis shows competitive and better results, in terms of success rate, with respect to the state-of-the-art GA-based methods in the literature (see Figure 3). We implemented the algorithm in C++, providing faster execution and accommodating a larger fitness evaluation budget.

1.4 APPENDIX

The appendix of the thesis is not devoted to RSs, but it is included in the manuscript to illustrate the work on a second line of research followed during the PhD studies. We present two works [17, 21] that use an emerging theory of complexity: Fine-Grained Complexity [61]. This theory aims to provide lower bounds for problems that already lie in **P**. The lower bounds are conditioned on famous conjectures like the Strong Exponential Time Hypothesis (SETH) [157]. A fine-grained reduction can be seen as a tool to prove n^k-*hardness* for problems solvable in $\mathcal{O}(n^k)$ time. In Appendix A, we apply this technique to prove quadratic lower bounds for problems of

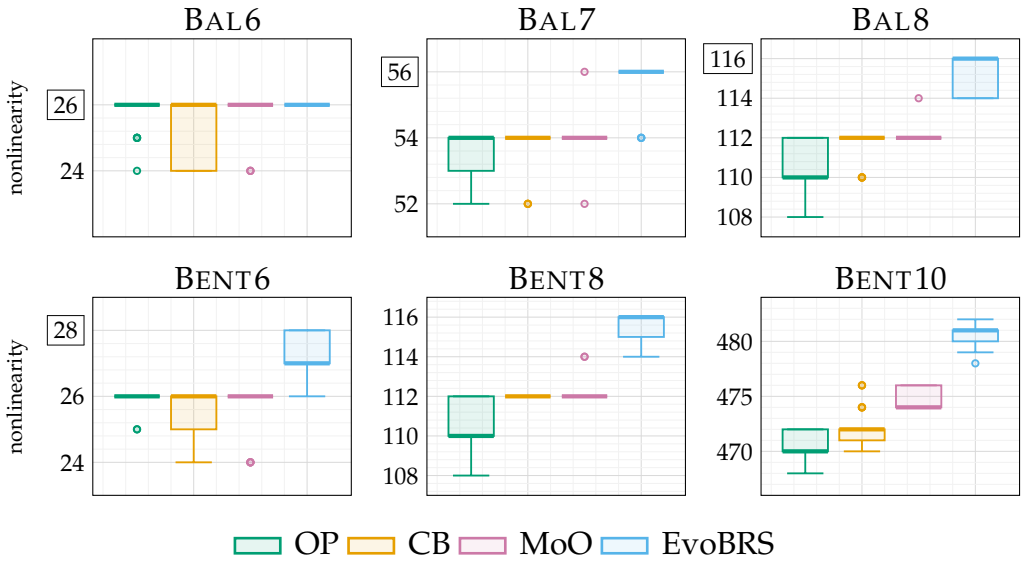


Figure 3. BENT n is the problem of finding Boolean functions on n variables with the highest possible linearity. With the additional constraint of balancedness, the problem is called BAL n. EvoBRS is compared with the best GA-algorithms in [182], and the final distributions, across 30 independent runs, are represented in the boxplots. The maximum possible nonlinearity value for each of the instances is boxed.

pattern matching between degenerate strings. Elastic Degenerate (*ED*) strings and Elastic Founder (*EF*) graphs are examples of degenerate strings widely studied in the literature. See Figure 4 for a schematic representation.

Figure 4. Example of the structure of a *ED* string and *EF* graph. The symbol □ represents any letter of a given alphabet and ϵ is the empty word.

We considered some subclasses of those degenerate strings based on their applications in Bioinformatics. For each possible pair of classes (X, Y), we investigated the problem of matching a string of type X into a string of type Y and established either a non-trivial upper bound or a quadratic lower bound conditioned on SETH.

Elastic Degenerate (*ED*) strings can be thought of as a particular kind of regular expressions called homogeneous. A regular expression is homogeneous if, given its representation as a rooted tree whose inner nodes correspond to operators and leaves correspond to letters of the alphabet, the inner nodes at any fixed level all correspond to the same operator (see Figure 5).

A complete study of homogeneous regular expression pattern matching has been completed in [30, 63]. In Appendix B, we consider all combinations of types of depth-2 homogenous regular expressions and classify the time complexity of nonemptiness intersection. We interestingly show that there exists a quadratic conditional lower bound for testing the intersection of a "concatenation of +s" expression with a "con-

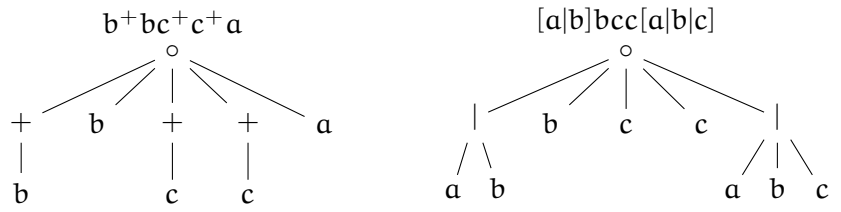


Figure 5. Example of two homogenous regular expressions. The one on the left is of type "concatenation of +s"; the right one is of type "concatenation of ORs". Furthermore, note that the regular expression [a|b]bcc[a|b|c] is a particular kind of degenerate string. Finally, the intersection of the two regular expressions, defined as the intersection of the languages they encode, is the string bbcca.

catenation of ORs". Furthermore, this is the only "hard" (i.e. quadratic) case that does not involve the Kleene star operator.

Part I

COMPLEXITY OF THE DYNAMICS OF RESOURCE-BOUNDED REACTION SYSTEMS

2

INTRODUCTION TO REACTION SYSTEMS DYNAMICS

This chapter introduces all the basic notions regarding Reaction Systems (RSs) that will be used in this part of the thesis. Introduced nearly two decades ago by Ehrenfeucht and Rozenberg [107], RSs are an abstract computational model inspired by the chemical reactions occurring in living cells. The notion at the heart of this model is that the biochemical processes within a cell can be simulated using a limited collection of entities that represent various substances, alongside a set of rules that mimic reactions. A reaction is characterized by its *reactants*, *inhibitors*, and *products*, and it occurs when the set of entities currently present in the cell (i.e. the system's state) includes all reactants and lacks any inhibitors, resulting in the reaction's products. Whenever a set of reactions occurs in a certain state, the system's subsequent state is determined by the union of the products of all those reactions. This process defines a dynamical system whose points are given by all the possible subsets of entities, i.e. all possible states of the RS (Figure 6). Determining the computational complexity of deciding the occurrence of various behaviours of such dynamical systems has been the object of a great deal of research work [28, 34, 37, 99, 101, 125, 204] , as well as several other related questions [65, 66, 90, 163].

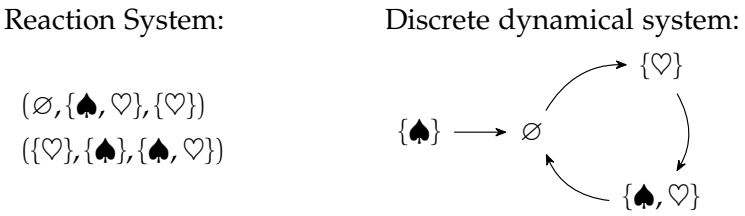


Figure 6. A simple example of a RS with the associated discrete dynamical system. The entities are ♠ and ♡.

Chapter organization. While Section 2.1 is devoted to the standard definitions, Section 2.2 details a description of Reaction Systems and their dynamics in terms of logical formulae which will be useful to prove most of the results in the following chapters.

2.1 BASICS NOTIONS ON REACTION SYSTEMS

Given a finite set S of *entities*, a *reaction* a over S is a triple (R_a, I_a, P_a) of subsets of S; R_a is the set of *reactants*, I_a the set of *inhibitors*, and P_a the nonempty set of *products*. In this thesis, the reactants and inhibitors of a reaction are allowed to be empty sets as in the original definition of reaction systems [107]. The set of all reactions over S is denoted by $\mathrm{rac}(S)$. A RS $\mathcal{A} = (S, A)$ where S consists of the finite set of entities S, called the *background set*, and a set $A \subseteq \mathrm{rac}(S)$ of reactions over S.

Any subset of S is a *state* of the RS; a reaction a is *enabled* in a state T when $R_a \subseteq T$ and $I_a \cap T = \varnothing$, and the set of all the reactions of $\mathcal{A}$ enabled in T is denoted by $\mathrm{en}_{\mathcal{A}}(T)$. The *result function* $\mathrm{res}_a : 2^S \to 2^S$ of a reaction a, where 2^S denotes the power set of S, is defined as

$$\mathrm{res}_a(T) \coloneqq \begin{cases} P_a & \text{if } a \text{ is enabled in } T \\ \varnothing & \text{otherwise.} \end{cases}$$

The definition of res_a naturally extends to sets of reactions: given any $T \subseteq S$ and $A \subseteq \mathrm{rac}(S)$, we define $\mathrm{res}_A(T) \coloneqq \bigcup_{a \in A} \mathrm{res}_a(T)$. Consistently, the result function $\mathrm{res}_{\mathcal{A}}$ of the whole RS $\mathcal{A} = (S, A)$ is defined to be equal to res_A, i.e. the result function of the whole set of reactions of the RS. In this way, any RS $\mathcal{A} = (S, A)$ induces a discrete dynamical system with state set 2^S and next state function $\mathrm{res}_{\mathcal{A}}$. Such a dynamical system can be represented with a directed graph with vertex set 2^S and with edges $(T, \mathrm{res}_{\mathcal{A}}(T))$ for all $T \in 2^S$. See Figure 7 for an example.

Example 2.1.1. Consider the RS $\mathcal{A} = (S, A)$ with background set $S = \{\spadesuit, \heartsuit, \diamondsuit\}$ and set of reactions $A = \{a = (\{\spadesuit\}, \{\heartsuit\}, \{\heartsuit\}), b = (\{\heartsuit\}, \{\spadesuit\}, \{\diamondsuit\}), c = (\{\diamondsuit\}, \{\heartsuit\}, \{\spadesuit\}))\}$. Let $T = \{\spadesuit, \diamondsuit\}$ a state, then T enables reaction a since $\{\spadesuit\} \subseteq T$ and $\{\heartsuit\} \cap T = \varnothing$; furthermore T enables reaction c since $\{\diamondsuit\} \subseteq T$ and $\{\heartsuit\} \cap T = \varnothing$. Collecting all together, the result of $\mathcal{A}$ on T is given by the union of $P_a = \{\heartsuit\}$ and $P_c = \{\spadesuit\}$, thus $\mathrm{res}_{\mathcal{A}}(T) = P_a \cup P_c = \{\spadesuit, \heartsuit\}$. The complete dynamics of $\mathcal{A}$ is shown in Figure 7.

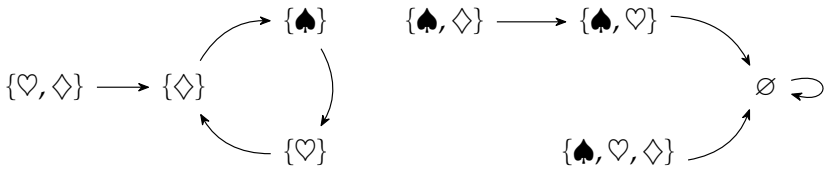


Figure 7. Graph representation of the dynamical system of the RS $\mathcal{A} = (S, A)$ with background set $S = \{\spadesuit, \heartsuit, \diamondsuit\}$ and set of reactions $A = \{(\{\spadesuit\}, \{\heartsuit\}, \{\heartsuit\}), (\{\heartsuit\}, \{\spadesuit\}, \{\diamondsuit\}), (\{\diamondsuit\}, \{\heartsuit\}, \{\spadesuit\}))\}$. Arrows connect each state T with $\mathrm{res}_{\mathcal{A}}(T)$.

In this thesis, we are interested in the *dynamics* of RSs, that is, the study of the successive states of the system under the action of the result function $\mathrm{res}_{\mathcal{A}}$ starting from some initial set of entities. The *orbit* or *state sequence* of a given state T of a RS $\mathcal{A}$ is defined as the sequence of states obtained by subsequent iterations of $\mathrm{res}_{\mathcal{A}}$ starting from T, namely, the sequence $(T, \mathrm{res}_{\mathcal{A}}(T), \mathrm{res}^2_{\mathcal{A}}(T), \dots)$. Note that since S is finite, for

any state T the sequence $\left(\mathrm{res}_{\mathcal{A}}^{n}(T)\right)_{n\in\mathbb{N}}$ is always ultimately periodic. In particular, the orbit of a state T is a *cycle* of length k (or k-cycle) if there exists $k \in \mathbb{N}$ such that $\mathrm{res}_{\mathcal{A}}^{k}(T) = T$, and $\mathrm{res}_{\mathcal{A}}^{h}(T) \neq T$ for every $h < k$. In the special case where $k = 1$, i.e. $\mathrm{res}_{\mathcal{A}}(T) = T$, T is said to be a *fixed point*. Note that fixed points of $\mathcal{A}$ correspond to self-loops in the graph representation of its dynamical system. For example, in the RS of Figure 7 the state $T = \varnothing$ is a fixed point, and the state $T = \{\diamondsuit\}$ belongs to a cycle of length 3.

Any set of cycles forms an *invariant set* for $\mathcal{A}$: a set of states $\mathcal{U} \subseteq 2^S$ such that $\cup_{U\in\mathcal{U}}\{\mathrm{res}_{\mathcal{A}}(U)\} = \mathcal{U}$. Conversely, any invariant set for $\mathcal{A}$ is a set of cycles [124]. A *local attractor* for $\mathcal{A}$ is an invariant set $\mathcal{U}$ such that there exists an external state $T \notin \mathcal{U}$ with $\mathrm{res}_{\mathcal{A}}(T) \in \mathcal{U}$. An invariant set $\mathcal{U}$ is a *global attractor* if for all states $T \in 2^S$ there exists $k \in \mathbb{N}$ such that $\mathrm{res}_{\mathcal{A}}^{k}(T) \in \mathcal{U}$, i.e. $\mathcal{U}$ is eventually reached from every possible state of $\mathcal{A}$. When a global attractor $\mathcal{U}$ consists of only one state T, we say that T is a *global fixed-point attractor*. If a local attractor $\mathcal{U}$ consists of only one state T, we say that T is a *fixed-point attractor*. A *fixed point not attractor* is a fixed point that is not an attractor, i.e. it is not reachable from any state other than T itself. Finally, $\mathcal{U}$ is a *global cycle attractor* if all the states in $\mathcal{U}$ belong to the same cycle (see Figure 8 for an example).

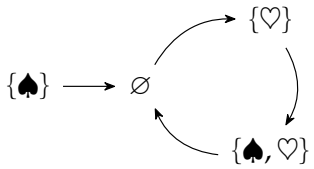


Figure 8. A global 3-cycle attractor in the dynamics of the RS $\mathcal{A} = (S, A)$ with background set $S = \{♠, ♡\}$ and reactions $A = \{(\varnothing, \{♠, ♡\}, \{♡\}), (\{♡\}, \{♠\}, \{♠, ♡\})\}$. Arrows connect each state T with $\mathrm{res}_{\mathcal{A}}(T)$.

Remark 2.1.1. It is important to clarify that the results obtained in this part of the thesis are limited to RSs where there is no interaction with an external environment (i.e. they are *closed*). Formally, we deal only with *context-independent RSs*, where the next state of the system is entirely determined by the current state. This is different from RSs with context, where additional entities could be added at each time step according to a given sequence $C_0, C_1, \ldots \subseteq S$ of *contexts*. More on that in Chapters 6 and 7.

We now recall the classification of RSs in terms of the number of resources employed per reaction [181].

Definition 2.1.1 ([181])**.** Let $i, r \in \mathbb{N}$. The class $\mathcal{RS}(r, i)$ consists of all RSs having at most r reactants and i inhibitors for each reaction. We also define the (partially) unbounded classes $\mathcal{RS}(\infty, i) = \bigcup_{r=0}^{\infty} \mathcal{RS}(r, i)$, $\mathcal{RS}(r, \infty) = \bigcup_{i=0}^{\infty} \mathcal{RS}(r, i)$, and $\mathcal{RS}(\infty, \infty) = \bigcup_{r=0}^{\infty} \bigcup_{i=0}^{\infty} \mathcal{RS}(r, i)$.

We will call $\mathcal{RS}(0, \infty)$ the class of *reactantless* RSs, and $\mathcal{RS}(\infty, 0)$ the class of *inhibitorless* RSs. See Figure 9 for an example of a reactantless RS and an inhibitorless RS.

(a) Dynamics of the reactantless RS given by the reaction $(\varnothing, \{\spadesuit, \heartsuit\}, \{\spadesuit\})$.

(b) Dynamics of the inhibitorless RS given by the reaction $(\{\spadesuit\}, \varnothing, \{\spadesuit, \heartsuit\})$.

Figure 9. A global 2-cycle attractor (9a) and a global attractor consisting of two fixed points (9b). No global 3-cycle attractor (like the one in Figure 8) can exist in the dynamics of the reactantless RS (Lemma 5.1.5) or in the dynamics of the inhibitorless RS (Proposition 5.3.5).

Class of RSs	**Subclass of $2^S \to 2^S$**
$\mathcal{RS}(\infty, \infty)$	all
$\mathcal{RS}(0, \infty)$	antitone
$\mathcal{RS}(\infty, 0)$	monotone
$\mathcal{RS}(1, 0)$	additive
$\mathcal{RS}(0, 0)$	constant

Table 3. Functions computed by restricted classed of RSs.

Definition 2.1.2 ([181]). Let $\mathcal{A} = (S, A)$ and $\mathcal{A}' = (S', A)$, with $S \subseteq S'$, be two RSs, and let $k \in \mathbb{N}$. We say that $\mathcal{A}'$ k-*simulates* $\mathcal{A}$ if and only if, for all $T \subseteq S$ and all $n \in \mathbb{N}$, we have

$$\mathrm{res}^n_{\mathcal{A}}(T) = \mathrm{res}^{kn}_{\mathcal{A}'}(T) \cap S.$$

Definition 2.1.3 ([181]). Let X and Y be classes of RSs, and let $k \in \mathbb{N}$. We define the binary relation $\preceq_k$ as follows: $X \preceq_k Y$ if and only if for all $\mathcal{A} \in X$ there exists a reaction system in Y that l-simulates $\mathcal{A}$ for some $l \leqslant k$. We say that $X \preceq Y$ if and only if $X \preceq_k Y$ for some $k \in \mathbb{N}$. We write $X \approx_k Y$ if $X \preceq_k Y$ and $Y \preceq_k X$, and $X \approx Y$ for $X \preceq Y \wedge Y \preceq X$. Finally, the notation $X \prec Y$ is shorthand for $X \preceq Y \wedge Y \not\preceq X$.

The relation $\preceq$ is a preorder and the relation $\approx$ induces exactly five equivalence classes [181, Theorem 30]:

$$\mathcal{RS}(0,0) \prec \mathcal{RS}(1,0) \prec \mathcal{RS}(\infty,0) \prec \mathcal{RS}(0,\infty) \prec \mathcal{RS}(\infty,\infty). \tag{1}$$

We remark that this classification does not consider the number of products as a parameter because RSs can always be assumed to be in *singleton product normal form* [60]: any reaction $(R, I, \{p_1, \ldots, p_m\})$ can be replaced by the set of reactions $(R, I, \{p_1\}), \ldots, (R, I, \{p_m\})$ that produce the same result.

The five equivalence classes in (1) have a characterisation in terms of functions over the Boolean lattice 2^S [181], listed in Table 3. Recall that a function $f : 2^S \to 2^S$ is *antitone* if $X \subseteq Y$ implies $f(X) \supseteq f(Y)$, *monotone* if $X \subseteq Y$ implies $f(X) \subseteq f(Y)$, *additive* (or an upper-semilattice endomorphism) if $f(X \cup Y) = f(X) \cup f(Y)$ for all $X, Y \in 2^S$. We say that the RS $\mathcal{A} = (S, A)$ computes the function $f : 2^S \to 2^S$ if $\mathrm{res}_{\mathcal{A}} = f$. Since

a function $f : 2^S \to 2^S$ is additive if and only if $f = \mathrm{res}_{\mathcal{A}}$ for some $\mathcal{A} \in \mathcal{RS}(1,0)$, we will refer to RSs in class $\mathcal{RS}(1,0)$ as the *additive* RSs, whose dynamical properties are exploited in Chapter 4.

2.2 LOGICAL DESCRIPTION

In this section, we recall a logical description of RSs and formulae related to their dynamics (see [123] for its first introduction). This description is sufficient for proving membership in many complexity classes. For the background notions of logic and descriptive complexity, we refer the reader to the classical book of Neil Immerman [156].

Each of the problems studied in this thesis can be characterised by a logical formula. A RS $\mathcal{A} = (S, A)$ with background set $S \subseteq \{0, \ldots, n-1\}$ and $|A| \leqslant n$ can be described by the vocabulary $(S, R_{\mathcal{A}}, I_{\mathcal{A}}, P_{\mathcal{A}})$, where S is a unary relation symbol and $R_{\mathcal{A}}, I_{\mathcal{A}}, P_{\mathcal{A}}$ are binary relation symbols. The intended meaning of the symbols is the following: the set of entities is $S = \{i : S(i)\}$ and each reaction $a_j = (R_j, I_j, P_j) \in A$ is described by the sets $R_j = \{i \in S : R_{\mathcal{A}}(i,j)\}$, $I_j = \{i \in S : I_{\mathcal{A}}(i,j)\}$, and $P_j = \{i \in S : P_{\mathcal{A}}(i,j)\}$. We will also need some additional vocabularies: $(S, R_{\mathcal{A}}, I_{\mathcal{A}}, P_{\mathcal{A}}, T)$, where T is a unary relation representing a subset of S, $(S, R_{\mathcal{A}}, I_{\mathcal{A}}, P_{\mathcal{A}}, T_1, T_2)$ with two additional unary relations representing sets, and $(S, R_{\mathcal{A}}, I_{\mathcal{A}}, P_{\mathcal{A}}, R_{\mathcal{B}}, I_{\mathcal{B}}, P_{\mathcal{B}})$ denoting two RSs over the same background set.

The following formulae, introduced in [123], describe the basic properties of $\mathcal{A}$. The first is true if a reaction a_j is enabled in T:

$$\textsc{en}_{\mathcal{A}}(j, T) \equiv \forall i(S(i) \Rightarrow (R_{\mathcal{A}}(i,j) \Rightarrow T(j)) \wedge (I_{\mathcal{A}}(i,j) \Rightarrow \neg T(j)))$$

and the following is verified if $\mathrm{res}_{\mathcal{A}}(T_1) = T_2$ for $T_1, T_2 \subseteq S$:

$$\textsc{res}_{\mathcal{A}}(T_1, T_2) \equiv \forall i(S(i) \Rightarrow (T_2(i) \Leftrightarrow \exists j(\textsc{en}_{\mathcal{A}}(j, T_1) \wedge P_{\mathcal{A}}(i,j)))).$$

Notice that $\textsc{en}_{\mathcal{A}}$ and $\textsc{res}_{\mathcal{A}}$ are both first-order (FO) formulae. We define the bounded second-order quantifiers $(\forall X \subseteq Y)\varphi$ and $(\exists X \subseteq Y)\varphi$ as a short-hand for $\forall X(\forall i(X(i) \Rightarrow Y(i)) \Rightarrow \varphi)$ and $\exists X(\forall i(X(i) \Rightarrow Y(i)) \wedge \varphi)$. We can use the following formulae (given in [123]) to describe the problems in Chapter 3:

$$\begin{aligned}
\textsc{fix}_{\mathcal{A}}(T) &\equiv \textsc{res}_{\mathcal{A}}(T, T)\\
\textsc{reach}_{\mathcal{A}}(T) &\equiv (\exists U \subseteq S)(\textsc{res}_{\mathcal{A}}(U, T) \wedge \neg \textsc{res}_{\mathcal{A}}(T, U))\\
\textsc{att}_{\mathcal{A}}(T) &\equiv \textsc{fix}_{\mathcal{A}}(T) \wedge \textsc{reach}_{\mathcal{A}}(T)\\
\textsc{fixge}_{\mathcal{A}}(T) &\equiv \textsc{fix}_{\mathcal{A}}(T) \wedge \neg \textsc{reach}_{\mathcal{A}}(T)\\
\textsc{res_eq}_{\mathcal{A},\mathcal{B}} &\equiv (\forall T \subseteq S)(\forall V \subseteq S)(\textsc{res}_{\mathcal{A}}(T, V) \Leftrightarrow \textsc{res}_{\mathcal{B}}(T, V))\\
\textsc{bij}_{\mathcal{A}} &\equiv (\forall T \subseteq S)(\forall U \subseteq S)(\forall V \subseteq S)(\textsc{res}_{\mathcal{A}}(T, V) \wedge \textsc{res}_{\mathcal{A}}(U, V) \Rightarrow \textsc{eq}(T, U)),
\end{aligned}$$

where $\textsc{eq}(T, U) \equiv (\forall i(S(i) \Rightarrow (T(i) \Leftrightarrow U(i))))$. We say that a formula is SO$\exists$, SO$\forall$, or SO$\forall\exists$ if it is logically equivalent to a formula in the required prenex normal form.

Following [124], in order to formulate the problems regarding cycles and global attractors, we need a stronger logic: a second-order logic with a transitive closure operator SO(TC). We denote the transitive closure of a formula $\varphi(X, Y)$ with two

Problem	**Formula**	**Logic class**
$\exists$ fixed point	$(\exists T \subseteq S)\textsc{fix}_{\mathcal{A}}(T)$	$SO\exists$
$\exists$ common fixed point	$(\exists T \subseteq S)(\textsc{fix}_{\mathcal{A}}(T) \wedge \textsc{fix}_{\mathcal{B}}(T))$	$SO\exists$
sharing all fixed points	$(\forall T \subseteq S)(\textsc{fix}_{\mathcal{A}}(T) \Leftrightarrow \textsc{fix}_{\mathcal{B}}(T))$	$SO\forall$
A given state is a fixed point attractor	$\textsc{att}_{\mathcal{A}}(T)$	$SO\exists$
$\exists$ fixed point attractor	$(\exists T \subseteq S)\textsc{att}_{\mathcal{A}}(T)$	$SO\exists$
$\exists$ common fixed point attractor	$(\exists T \subseteq S)\textsc{att}_{\mathcal{A}}(T) \wedge \textsc{att}_{\mathcal{B}}(T)$	$SO\exists$
sharing all fixed points attractors	$(\forall T \subseteq S)(\textsc{att}_{\mathcal{A}}(T) \Leftrightarrow \textsc{att}_{\mathcal{B}}(T))$	$SO\forall\exists$
$\exists$ fixed point not attractor	$(\exists T \subseteq S)\textsc{fixge}(T)$	$SO\exists\forall$
$\exists$ common fixed point not attractor	$(\exists T \subseteq S)(\textsc{fixge}_{\mathcal{A}}(T) \wedge \textsc{fixge}_{\mathcal{B}}(T))$	$SO\exists\forall$
sharing all fixed points not attractors	$(\forall T \subseteq S)(\textsc{fixge}_{\mathcal{A}}(T) \Leftrightarrow \textsc{fixge}_{\mathcal{B}}(T))$	$SO\forall$
$\mathrm{res}_{\mathcal{A}} = \mathrm{res}_{\mathcal{B}}$	$\textsc{res\_eq}_{\mathcal{A},\mathcal{B}}$	$SO\forall$
res bijective	$\textsc{bij}_{\mathcal{A}}$	$SO\forall$
A given state is a global attractor	$\textsc{glob}_{\mathcal{A}}(T)$	$SO(TC)$
$\exists$ a global fixed point attractor	$(\exists T \subseteq S)\textsc{glob}_{\mathcal{A}}(T)$	$SO(TC)$
A given state is part of a cycle	$\textsc{cycle}_{\mathcal{A}}(T)$	$SO(TC)$
$\exists$ common cycle	$(\exists T \subseteq S)\textsc{cycle}_{\mathcal{A}}(T)$	$SO(TC)$
sharing all cycles	$\textsc{sharecycle}_{\mathcal{A},\mathcal{B}}$	$SO(TC)$

Table 4. Problems studied in Chapters 3 and 5 with the associated formulae and logic classes.

free second-order variables by $\varphi^\star(X, Y)$. We can use the following formulae (given in [124]) to describe the problems in Chapter 5:

$$
\begin{aligned}
\textsc{path}_{\mathcal{A}}(T, U) &\equiv \textsc{res}^{\star}_{\mathcal{A}}(T, U)\\
\textsc{glob}_{\mathcal{A}}(T) &\equiv \textsc{fix}_{\mathcal{A}}(T) \wedge (\forall U \subseteq S)\textsc{path}_{\mathcal{A}}(U, T)\\
\textsc{cycle}_{\mathcal{A}}(T) &\equiv \exists U(\textsc{res}_{\mathcal{A}}(T, U) \wedge \textsc{path}_{\mathcal{A}}(U, T))\\
\textsc{res}_{\mathcal{A},\mathcal{B}}(T, U) &\equiv \textsc{res}_{\mathcal{A}}(T, U) \wedge \textsc{res}_{\mathcal{B}}(T, U)\\
\textsc{path}_{\mathcal{A},\mathcal{B}}(T, U) &\equiv \textsc{res}^{\star}_{\mathcal{A},\mathcal{B}}(T, U)\\
\textsc{cycle}_{\mathcal{A},\mathcal{B}}(T) &\equiv (\exists U \subseteq S)(\textsc{res}_{\mathcal{A},\mathcal{B}}(T, U) \wedge \textsc{path}_{\mathcal{A},\mathcal{B}}(U, T))\\
\textsc{sharecycle}_{\mathcal{A},\mathcal{B}} &\equiv (\forall T \subseteq S)(\textsc{cycle}_{\mathcal{A}}(T) \vee \textsc{cycle}_{\mathcal{B}}(T) \Rightarrow \textsc{cycle}_{\mathcal{A},\mathcal{B}}(T)).
\end{aligned}
$$

In Table 4, we give the logic formulae associated with the problems considered in Chapter 3 and Chapter 5.

Remark that existential second-order logic $SO\exists$ characterizes **NP** (Fagin's theorem [156]); universally quantified second-order logic $SO\forall$ gives **coNP**; second-order logic with one alternation of existential and universal quantifiers $SO\exists\forall$ gives $\Sigma_2^{\mathbf{P}}$; in a dual way, second-order logic with one alternation of universal and existential quantifiers $SO\forall\exists$ gives $\Pi_2^{\mathbf{P}}$; second-order logic with a transitive closure operator $SO(TC)$ gives **PSPACE** [156].

3

FIXED POINTS AND ATTRACTORS OF REACTANTLESS AND INHIBITORLESS REACTION SYSTEMS

In this chapter, we classify the computational complexity of deciding on the occurrence of behaviours related to fixed points, attractors and result functions in the special classes of *reactantless* and *inhibitorless* RSs. Even though these problems had been extensively studied in the unconstrained case [101], they remained unexplored in these restricted classes. The main results of this chapter are summarized in Table 5.

Problem	$\mathcal{RS}(\infty,\infty)$	$\mathcal{RS}(0,\infty)$	$\mathcal{RS}(\infty,0)$
A given state is a fixed point attractor	**NP**-c [123]	**NP**-c (Thm. 3.2.2)	**NP**-c (Thm. 3.2.1)
$\exists$ fixed point	**NP**-c [123]	**NP**-c (Thm. 3.3.1)	**P** [139]
$\exists$ common fixed point	**NP**-c [123]	**NP**-c (Cor. 3.3.2)	**NP**-c (Thm. 3.4.1)
sharing all fixed points	**coNP**-c [123]	**coNP**-c (Thm. 3.3.10)	**coNP**-c (Thm. 3.4.3)
$\exists$ fixed point attractor	**NP**-c [123]	**NP**-c (Thm. 3.3.3)	Unknown
$\exists$ common fixed point attractor	**NP**-c [123]	**NP**-c (Cor. 3.3.4)	**NP**-c (Cor. 3.4.2)
sharing all fixed points attractors	$\Pi_2^{\mathbf{P}}$-c [123]	$\Pi_2^{\mathbf{P}}$-c (Cor. 3.3.9)	$\Pi_2^{\mathbf{P}}$-c (Thm. 3.4.5)
$\exists$ fixed point not attractor	$\Sigma_2^{\mathbf{P}}$-c (Cor. 3.3.7)	$\Sigma_2^{\mathbf{P}}$-c (Thm. 3.3.5)	$\Sigma_2^{\mathbf{P}}$-c (Cor. 3.4.6)
$\exists$ common fixed point not attractor	$\Sigma_2^{\mathbf{P}}$-c (Cor. 3.3.8)	$\Sigma_2^{\mathbf{P}}$-c (Cor. 3.3.6)	$\Sigma_2^{\mathbf{P}}$-c (Cor. 3.4.7)
sharing all fixed points not attractors	**coNP**-c (Cor. 3.3.12)	**coNP**-c (Cor. 3.3.11)	**coNP**-c (Cor. 3.4.4)
$\mathrm{res}_{\mathcal{A}} = \mathrm{res}_{\mathcal{B}}$	**coNP**-c (Thm. 3.5.1)	**P** (Cor. 3.5.6)	**P** (Cor. 3.5.4)
res bijective	**coNP**-c [124]	**P** (Cor. 3.6.10)	**P** (Cor. 3.6.5)

Table 5. Computational complexity of the problems studied in this chapter for different classes of RSs. **NP**-c, **coNP**-c, $\Sigma_2^{\mathbf{P}}$-c and $\Pi_2^{\mathbf{P}}$-c are shorthands for **NP**-complete, **coNP**-complete, $\Sigma_2^{\mathbf{P}}$-complete and $\Pi_2^{\mathbf{P}}$-complete, respectively; $\mathcal{RS}(\infty,\infty)$, $\mathcal{RS}(0,\infty)$ and $\mathcal{RS}(\infty,0)$ denote unconstrained, reactantless and inhibitorless RSs, respectively (see Definition 2.1.1). Light-blue cells contain the results presented in this chapter.

Chapter organization. In Section 3.2, we study the problem of deciding whether a given state is a fixed point attractor in both constrained models. In Sections 3.3 and 3.4 we study the complexities of fixed point problems in reactantless, resp. inhibitorless, RSs. In Section 3.5 we study the problem of deciding whether two RSs have the same result function. In Section 3.6 we consider the problem of deciding whether the result function is bijective. Finally, in Section 3.7 we discuss our results and suggest future research directions.

This chapter is based on the following publication: R. Ascone, G. Bernardini, L. Manzoni. Fixed points and attractors of Reactantless and Inhibitorless Reaction Systems. In: *Theoretical Computer Science*, 2024 [20].

3.1 TOOLS FOR THE REDUCTIONS

The hardness proofs we give in this chapter are obtained via reductions from SAT, VALIDITY [214] or ∀∃SAT [249]. In Definition 3.1.1 we introduce a few sets of entities corresponding to a Boolean formula in CNF or DNF that will be used in such proofs.

Definition 3.1.1. Given φ a Boolean formula in CNF or DNF with clauses $C = \{\varphi_1, \ldots, \varphi_m\}$ over the variables $V = \{x_1, \ldots, x_n\}$, we define the additional set of entities $\overline{V} \coloneqq \{\overline{x_j} : x_j \in V\}$. The elements of the power set of V and $\overline{V}$ are in a natural bijection: thus for any $X \subseteq V$, $\overline{X}$ denotes the set $\{\overline{x} : x \in X\} \subseteq \overline{V}$. Furthermore, we will denote by $\mathrm{pos}(\varphi_r) \subseteq V$ the variables that occur non-negated in φ_r and $\mathrm{neg}(\varphi_r) \subseteq V$ the variables that occur negated in φ_r. With the identification above we will also have $\overline{\mathrm{pos}}(\varphi_r) \subseteq \overline{V}$ and $\overline{\mathrm{neg}}(\varphi_r) \subseteq \overline{V}$. In addition, we will use card suit symbols $\{\spadesuit, \heartsuit, \diamondsuit, \clubsuit\}$ to denote extra entities that do not belong to any of the sets defined above.

In the reductions provided in several proofs in this chapter, we will represent any assignment for a Boolean formula φ as a subset of entities $X \cup \overline{V \setminus X}$, where $X \subseteq V$ is the subset of variables that are assigned value `true` and $V \setminus X$ is the subset of variables that are assigned value `false`. In general, when interpreting sets of entities as assignments, the elements of V represent variables that are assigned value `true`, and the elements of $\overline{V}$ variables that are assigned value `false`. For instance, the inclusion $\mathrm{pos}(\varphi_r) \subseteq X$ can be interpreted as *all variables non-negated in φ_r are set to true*; $\overline{\mathrm{pos}}(\varphi_r) \cap \overline{V \setminus X} \neq \varnothing$ as *there exists a non-negated variable in φ_r that is set to false*; $\overline{\mathrm{neg}}(\varphi_r) \cap \overline{V \setminus X} = \varnothing$ as *none of the variables that occurred negated in φ_r is set to false*; and similarly for all the other cases. Furthermore, $X \cup \overline{V \setminus X} \vDash \varphi$ (resp. $X \cup \overline{V \setminus X} \nvDash \varphi$) denotes that the assignment satisfies (resp. does not satisfy) φ.

3.2 A GIVEN STATE IS A FIXED POINT ATTRACTOR

In this section, we study the problem of deciding whether a given state is a fixed point attractor. The problem is **NP**-complete for $\mathcal{RS}(\infty, \infty)$ [123, Theorem 4]; we prove that it remains so also for reactantless and inhibitorless reaction systems.

Theorem 3.2.1. *Given $A \in \mathcal{RS}(\infty, 0)$ and a state $T \subseteq S$, it is* **NP***-complete to decide whether T is a fixed point attractor.*

Proof. The problem is in **NP** (see Table 4). In order to show **NP**-hardness, we give a reduction from SAT [214]. Given a Boolean formula $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_m$ in CNF

over the variables $V = \{x_1, \dots, x_n\}$, we define a RS $\mathcal{A}$ with background set $S \coloneqq V \cup \overline{V} \cup C \cup \{\spadesuit\}$, where $C, V, \overline{V}, \spadesuit$ are as in Definition 3.1.1, and reactions

$$
\begin{aligned}
&(\{x\}, \varnothing, \{\varphi_j\}) && \text{for } 1 \leqslant j \leqslant m \text{ and } x \in \mathrm{pos}(\varphi_j) && (2)\\
&(\{\overline{x}\}, \varnothing, \{\varphi_j\}) && \text{for } 1 \leqslant j \leqslant m \text{ and } \overline{x} \in \overline{\mathrm{neg}}(\varphi_j) && (3)\\
&(\{x_i, \overline{x_i}\}, \varnothing, \{\spadesuit\}) && \text{for } 1 \leqslant i \leqslant n && (4)\\
&(\{x_i, \varphi_j\}, \varnothing, \{\spadesuit\}) && \text{for } 1 \leqslant i \leqslant n \text{ and } 1 \leqslant j \leqslant m && (5)\\
&(\{\overline{x_i}, \varphi_j\}, \varnothing, \{\spadesuit\}) && \text{for } 1 \leqslant i \leqslant n \text{ and } 1 \leqslant j \leqslant m && (6)\\
&(\{\varphi_j\}, \varnothing, \{\varphi_j\}) && \text{for } 1 \leqslant j \leqslant m && (7)\\
&(\{\spadesuit\}, \varnothing, \{\spadesuit\}). && && (8)
\end{aligned}
$$

Reactions (7) imply that $T \coloneqq C$ is a fixed point for $\mathcal{A}$. Furthermore, consider a state $T' \neq C$ such that $\mathrm{res}_{\mathcal{A}}(T') = C$. It must hold $T' = X_1 \cup \overline{X_2}$ with $X_1 \subseteq V$, $X_2 \subseteq V$ and $X_1 \cap X_2 = \varnothing$, as otherwise $\spadesuit$ would be generated by one of the reactions of type (4), (5), (6) or (8). Furthermore, consider $\varphi_j \in \mathrm{res}_{\mathcal{A}}(T')$ for any $1 \leqslant j \leqslant m$: since $T' \cap C = \varnothing$, then either $X_1 \cap \mathrm{pos}(\varphi_j) \neq \varnothing$ or $X_2 \cap \mathrm{neg}(\varphi_j) \neq \varnothing$ because at least one of the reactions of type (2) or (3) with φ_j in the product must be enabled. We can thus construct an assignment satisfying φ from $T' = X_1 \cup \overline{X_2}$. We remark that if $X_1 \cup X_2 \subsetneq V$, the variables in $V \setminus (X_1 \cup X_2)$ are irrelevant to the satisfiability of φ, thus they can be assigned any value. Therefore, we proved that if C is an attractor then φ is satisfiable.

The converse follows immediately, since if $X \subseteq V$ is the subset of variables that are given a true value in an assignment that satisfies φ, then $X \cup \overline{V \setminus X}$ enables at least one reaction of type (2) or (3) for each clause, thus $\mathrm{res}_{\mathcal{A}}(X \cup \overline{V \setminus X}) = C$. We obtain that $T = C$ is a fixed point attractor if and only if φ is satisfiable. The mapping $\varphi \mapsto (\mathcal{A}, T)$ is clearly computable in polynomial time; hence deciding if a given fixed point T is an attractor is **NP**-hard. □

We highlight that all the reactions used in the proof of Theorem 3.2.1 involve at most two reactants, implying that the problem is **NP**-complete even for the more restricted class $\mathcal{RS}(2, 0)$. Note, however, that if we are given a fixed point of $\mathcal{A}$, it is in **P** to decide whether it is a special kind of attractor: namely if it can be reached from one of its subsets or from one of its supersets, as we prove in the following remark.

Remark 3.2.1. Let $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ and T be a fixed point for $\mathrm{res}_{\mathcal{A}}$. If there exists $T' \subsetneq T$ such that $\mathrm{res}_{\mathcal{A}}(T') = T$, then for all $x \in T \setminus T'$

$$T = \mathrm{res}_{\mathcal{A}}(T') \subseteq \mathrm{res}_{\mathcal{A}}(T \setminus \{x\}) \subseteq \mathrm{res}_{\mathcal{A}}(T) = T,$$

thus T is reachable from $T \setminus \{x\}$; and if there exists $T'' \supsetneq T$ such that $\mathrm{res}_{\mathcal{A}}(T'') = T$, then for all $x \in T'' \setminus T$

$$T = \mathrm{res}_{\mathcal{A}}(T) \subseteq \mathrm{res}_{\mathcal{A}}(T \cup \{x\}) \subseteq \mathrm{res}_{\mathcal{A}}(T'') = T,$$

thus T is reachable from $T \cup \{x\}$. To decide whether T is an attractor of the form T' and T'' it thus suffices to check, for all $x \in S$, whether T is reachable from $T \setminus \{x\}$ or $T \cup \{x\}$, and it thus requires polynomial time.

Theorem 3.2.2. *Given $\mathcal{A} \in \mathcal{RS}(0, \infty)$ and a state $T \subseteq S$, it is* **NP***-complete to decide whether T is a fixed point attractor.*

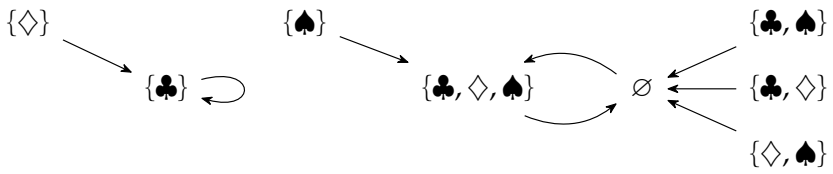


Figure 10. Graph representation of the dynamics given by the reactions (14), (15), (16) over the subset of the background set $\{\clubsuit, \diamondsuit, \spadesuit\}$.

Proof. The problem is in **NP** (see Table 4). In order to show **NP**-hardness, we give a reduction from SAT [214]. Given a Boolean formula $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_m$ in CNF over the variables $V = \{x_1, \ldots, x_n\}$, we define $\mathcal{A}$ a RS with background set $S := V \cup \overline{V} \cup C \cup \{\clubsuit, \diamondsuit, \spadesuit\}$, where each set is as defined in Definition 3.1.1, and the set of reactions is

$$(\varnothing, \{\overline{x}, \clubsuit, \spadesuit\}, \{\varphi_j\}) \quad \text{for } 1 \leqslant j \leqslant m \text{ and } \overline{x} \in \overline{\mathrm{pos}}(\varphi_j) \tag{9}$$
$$(\varnothing, \{x, \clubsuit, \spadesuit\}, \{\varphi_j\}) \quad \text{for } 1 \leqslant j \leqslant m \text{ and } x \in \mathrm{neg}(\varphi_j) \tag{10}$$
$$(\varnothing, \{x_i, \overline{x_i}, \clubsuit, \spadesuit\}, \{\spadesuit\}) \quad \text{for } 1 \leqslant i \leqslant n \tag{11}$$
$$(\varnothing, \{\varphi_j\}, \{\spadesuit\}) \quad \text{for } 1 \leqslant j \leqslant m \tag{12}$$
$$(\varnothing, S \setminus (C \cup \{\clubsuit\}), C) \tag{13}$$
$$(\varnothing, \{\diamondsuit, \spadesuit\}, \{\clubsuit\}) \tag{14}$$
$$(\varnothing, \{\clubsuit, \spadesuit\}, \{\clubsuit\}) \tag{15}$$
$$(\varnothing, \{\clubsuit, \diamondsuit\}, \{\clubsuit, \diamondsuit, \spadesuit\}) \tag{16}$$

By reactions (13) and (14), $T := C \cup \{\clubsuit\}$ is a fixed point for $\mathcal{A}$. Let $T' \neq C \cup \{\clubsuit\}$ such that $\mathrm{res}_{\mathcal{A}}(T') = C \cup \{\clubsuit\}$, then $C \subseteq T'$, otherwise $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T')$ because of the reactions of type (12). In order to have $\mathrm{res}_{\mathcal{A}}(T') \cap \{\clubsuit, \diamondsuit, \spadesuit\} = \{\clubsuit\}$, T' must either satisfy $T' \cap \{\clubsuit, \diamondsuit, \spadesuit\} = \{\clubsuit\}$ or $T' \cap \{\clubsuit, \diamondsuit, \spadesuit\} = \{\diamondsuit\}$ (see Figure 10). We prove by contradiction that the first case cannot occur: indeed, in this case, the only reactions enabled by T' are (13) and (14). Since adding any element of $V \cup \overline{V}$ to T' would disable (13) and we have $C \subseteq \mathrm{res}_{\mathcal{A}}(T')$, we deduce that we would have $T' = C \cup \{\clubsuit\}$, a contradiction. Therefore T' must be of the form $X_1 \cup \overline{X_2} \cup C \cup \{\diamondsuit\}$, where $X_1 \subseteq V$ and $\overline{X_2} \subseteq \overline{V}$. If we had $X_1 \cup X_2 \subsetneq V$ then we would also have $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T')$ because one of the reactions (11) would be enabled; thus it must hold $X_1 \cup X_2 = V$.

If $x \in X_1 \cap X_2$ then neither x nor $\overline{x}$ will be able to generate any φ_j because of reactions (9) and (10), and its value is therefore irrelevant to the satisfiability of φ; however, since $C \subseteq \mathrm{res}_{\mathcal{A}}(T')$, for each $\varphi_j \in C$ it must hold either $(X_1 \setminus X_2) \cap \mathrm{pos}(\varphi_j) \neq \varnothing$ or $(X_2 \setminus X_1) \cap \mathrm{neg}(\varphi_j) \neq \varnothing$. Therefore, we proved that if $C \cup \{\clubsuit\}$ is an attractor then φ is satisfiable with an assignment of the type $X_1 \setminus (X_1 \cap X_2) \cup \overline{X_2 \setminus (X_1 \cap X_2)}$, where the variables in $(X_1 \cap X_2)$ can be assigned any value. The converse follows immediately, since if $X \subseteq V$ are the variables set to true in an assignment satisfying φ, then $X \cup \overline{V \setminus X} \cup C \cup \{\diamondsuit\}$ is a state attracted by $C \cup \{\clubsuit\}$. We obtain that $T = C \cup \{\clubsuit\}$ is a fixed point attractor if and only if φ is satisfiable. The mapping $\varphi \mapsto (\mathcal{A}, T)$ is computable in polynomial time, hence deciding if a given fixed point T is an attractor is **NP**-hard. □

3.3 FIXED POINTS FOR REACTANTLESS RSS

In this section, we prove **NP**-hardness, **coNP**-hardness, $\Sigma_2^{\mathbf{P}}$-hardness and $\Pi_2^{\mathbf{P}}$-hardness for problems of fixed points in the class of reactantless RSs.

The problem of deciding if there exists a fixed point is **NP**-complete for $\mathcal{RS}(\infty,\infty)$ [123, Theorem 2]; the following theorem shows that it remains difficult also in $\mathcal{RS}(0,\infty)$.

Theorem 3.3.1. *Given $\mathcal{A} \in \mathcal{RS}(0,\infty)$, it is* **NP***-complete to decide if $\mathcal{A}$ has a fixed point.*

Proof. The problem is in **NP** (see Table 4). In order to show **NP**-hardness, we reduce SAT [214] to this problem. Given a Boolean formula $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_m$ in CNF over the variables $V = \{x_1, \ldots, x_n\}$, we define a RS $\mathcal{A}$ with background set $S := V \cup \overline{V} \cup \{\spadesuit\} \cup \{\clubsuit\}$ (the sets are as in Definition 3.1.1) and the reactions

$$(\varnothing, \overline{\mathrm{neg}}(\varphi_j) \cup \mathrm{pos}(\varphi_j), \{\spadesuit\}) \qquad \text{for } 1 \leqslant j \leqslant m \tag{17}$$

$$\overline{a_i} := (\varnothing, \{x_i\}, \{\overline{x_i}\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{18}$$

$$a_i := (\varnothing, \{\overline{x_i}\}, \{x_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{19}$$

$$(\varnothing, \{\clubsuit\}, \{\clubsuit, \spadesuit\}) \tag{20}$$

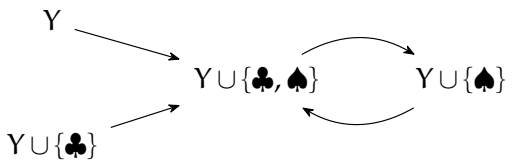
$$(\varnothing, \{\spadesuit\}, \{\clubsuit\}). \tag{21}$$

Given a state $T \subseteq S$, let $T_V = T \cap V$ and $T_{\overline{V}} = T \cap \overline{V}$. When $x_j \in T_V \Leftrightarrow \overline{x_j} \notin T_{\overline{V}}$ for every j, then $T_V \cup T_{\overline{V}}$ encodes an assignment of φ in which the variables having true value are those in T_V and the variables having false value are those in $T_{\overline{V}}$. In this case we say that T is a *well-formed* state of $\mathcal{A}$ and the reactions of type (18),(19) preserve $T_V \cup T_{\overline{V}}$, i.e. $T_V \cup T_{\overline{V}} \subseteq \mathrm{res}_{\mathcal{A}}(T)$. Instead, if T is not a well-formed state then we distinguish two cases:

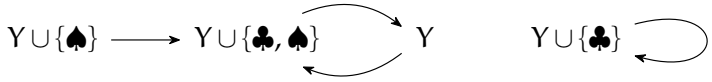
- if $\exists x_i, \overline{x_i} \in T_V \cup T_{\overline{V}}$ then $x_i, \overline{x_i} \notin \mathrm{res}_{\mathcal{A}}(T)$, since neither a_i nor $\overline{a_i}$ is enabled;
- if $\exists x_i, \overline{x_i} \notin T_V \cup T_{\overline{V}}$ then $x_i, \overline{x_i} \in \mathrm{res}_{\mathcal{A}}(T)$, since both a_i and $\overline{a_i}$ are enabled.

In both cases, $T \neq \mathrm{res}_{\mathcal{A}}(T)$; so if T is a fixed point, T is a well-formed state. For well-formed states, we can also give an interpretation of the reactions of type (17): they evaluate each disjunctive clause (which is not satisfied if and only if no positive variables are set to true and no negative ones to false) and generate $\spadesuit$ when φ itself is not satisfied by $T_V \cup T_{\overline{V}}$.

Consider the dynamic of the reaction system restricted to a well-formed state $Y \subseteq V \cup \overline{V}$. If Y does not satisfy φ then there are no fixed points among the well-formed states containing Y because of the following (the arrows represent function $\mathrm{res}_{\mathcal{A}}$):



Instead, if Y satisfies φ, then $Y \cup \{\clubsuit\}$ is a fixed point since:



In particular, $\mathcal{A}$ has a fixed point if and only φ is satisfiable. The mapping $\varphi \mapsto \mathcal{A}$ is computable in polynomial time, hence deciding on the existence of fixed points for reactantless RSs is **NP**-hard. □

As an immediate consequence of Theorem 3.3.1 we obtain that deciding if there exists a state that is a common fixed point of two reactantless reaction systems remains **NP**-complete.

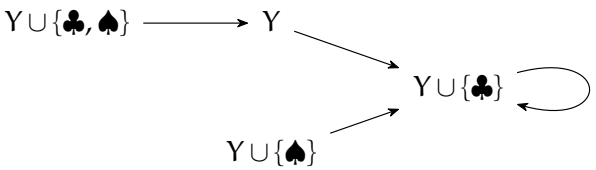
Corollary 3.3.2. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(0, \infty)$ with a common background set S, it is* **NP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point.*

Proof. The problem is in **NP** (see Table 4). By Theorem 3.3.1, when $\mathcal{A} = \mathcal{B}$ the problem is **NP**-complete. □

With a small adaptation of the proof of Theorem 3.3.1, deciding if a fixed point attractor exists is still an **NP**-complete problem. The **NP**-completeness for $\mathcal{RS}(\infty, \infty)$ is proved in [123, Corollary 3]; the following theorem proves it for the case of reactantless RSs.

Theorem 3.3.3. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$, it is* **NP***-complete to decide if $\mathcal{A}$ has a fixed point attractor.*

Proof. The problem is in **NP**, as highlighted in Table 4. In order to show **NP**-hardness, we reduce SAT [214] to this problem. Given a Boolean formula $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_m$ in CNF, we construct the same reaction system $\mathcal{A}$ of Theorem 3.3.1 and we substitute reaction (20) with $(\varnothing, \{\clubsuit\}, \{\clubsuit\})$. In this way, if $Y \subseteq V \cup \overline{V}$ is a well-formed state satisfying φ we have:



which means that $Y \cup \{\clubsuit\}$ is a fixed point reachable from Y or $Y \cup \{\spadesuit\}$. In the other cases (either a well-formed state not satisfying φ or a not well-formed state), $T \subseteq S$ is never a fixed point, as in the proof of Theorem 3.3.1. Since the mapping $\varphi \mapsto \mathcal{A}$ is computable in polynomial time, deciding on the existence of a fixed points attractor for reactantless RSs is **NP**-hard. □

Corollary 3.3.4. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(0, \infty)$ with a common background set S, it is* **NP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point attractor.*

Proof. The problem is in **NP** (see Table 4). By Theorem 3.3.3, when $\mathcal{A} = \mathcal{B}$ the problem is **NP**-hard. □

In contrast, if we consider the existence of a fixed point not attractor the problem is in $\Sigma_2^{\mathbf{P}}$; the following theorem proves the $\Sigma_2^{\mathbf{P}}$-hardness.

Theorem 3.3.5. *Given $\mathcal{A} \in \mathcal{RS}(0, \infty)$, it is $\Sigma_2^{\mathbf{P}}$-complete to decide if $\mathcal{A}$ has a fixed point which is not an attractor.*

Proof. The problem is in $\Sigma_2^{\mathbf{P}}$ (see Table 4). Consider the converse problem, i.e. decide if all fixed points are attractors. In order to show $\Pi_2^{\mathbf{P}}$-hardness of the latter, we construct a reduction from $\forall\exists$SAT [249]. Given a quantified Boolean formula $(\forall V_1)(\exists V_2)\varphi$ over $V = \{x_1, \dots, x_n\}$, with $V_1 \subseteq V$, $V_2 = V \setminus V_1$ and $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_m$ quantifier-free and in CNF, let $\heartsuit_S := \{\heartsuit_i : 1 \leqslant i \leqslant n\}$ be a set of extra entities that are not contained in $V \cup \overline{V} \cup C$ and $V, \overline{V}, C$ are as in Definition 3.1.1. In other words, we want to prove that for any assignment for the variables in V_1, there exists an assignment for the variables in V_2 that satisfies φ if and only if all fixed points of a suitably defined RS are attractors. We thus define a RS $\mathcal{A}$ with background set $S := V \cup \overline{V} \cup C \cup \heartsuit_S \cup \{\clubsuit, \diamondsuit, \spadesuit\}$ and the reactions

$$(\varnothing, \{\overline{x}, \clubsuit, \spadesuit\}, \{\varphi_j\}) \quad \text{for } 1 \leqslant j \leqslant m \quad \text{and } \overline{x} \in \overline{\text{pos}}(\varphi_j) \tag{22}$$
$$(\varnothing, \{x, \clubsuit, \spadesuit\}, \{\varphi_j\}) \quad \text{for } 1 \leqslant j \leqslant m \quad \text{and } x \in \text{neg}(\varphi_j) \tag{23}$$
$$(\varnothing, \overline{\text{neg}}(\varphi_j) \cup \text{pos}(\varphi_j) \cup \{\clubsuit, \spadesuit\}, \{\spadesuit\}) \quad \text{for } 1 \leqslant j \leqslant m \tag{24}$$
$$(\varnothing, \{x_i, \clubsuit, \spadesuit\}, \{\heartsuit_i\}) \quad \text{for } 1 \leqslant i \leqslant n \tag{25}$$
$$(\varnothing, \{\overline{x_i}, \clubsuit, \spadesuit\}, \{\heartsuit_i\}) \quad \text{for } 1 \leqslant i \leqslant n \tag{26}$$
$$(\varnothing, \{x_i, \overline{x_i}, \clubsuit, \spadesuit\}, \{\spadesuit\}) \quad \text{for } 1 \leqslant i \leqslant n \tag{27}$$
$$(\varnothing, \{\varphi_j\}, \{\spadesuit\}) \quad \text{for } 1 \leqslant j \leqslant m \tag{28}$$
$$(\varnothing, \{\heartsuit_i\}, \{\spadesuit\}) \quad \text{for } 1 \leqslant i \leqslant n \tag{29}$$
$$(\varnothing, \{\overline{x}\}, \{x\}) \quad \text{for } x \in V_1 \tag{30}$$
$$(\varnothing, \{x\}, \{\overline{x}\}) \quad \text{for } x \in V_1 \tag{31}$$
$$(\varnothing, S \setminus (C \cup \heartsuit_S \cup \{\clubsuit\} \cup V_1 \cup \overline{V_1}), C \cup \heartsuit_S) \tag{32}$$
$$(\varnothing, \{\diamondsuit, \spadesuit\}, \{\clubsuit\}) \tag{33}$$
$$(\varnothing, \{\clubsuit, \spadesuit\}, \{\clubsuit\}) \tag{34}$$
$$(\varnothing, \{\clubsuit, \diamondsuit\}, \{\clubsuit, \diamondsuit, \spadesuit\}). \tag{35}$$

To study the dynamics of $\mathcal{A}$ we first analyze the form of its fixed points.

Claim 1. If T is a fixed point of $\mathcal{A}$, then $\clubsuit \in T$, $\diamondsuit \notin T$ and $\spadesuit \notin T$.

Proof. If we had $\diamondsuit \in T$, then the only reaction that can produce $\diamondsuit$, i.e. reaction (35), is not enabled, thus $\diamondsuit \notin \text{res}_{\mathcal{A}}(T) = T$, a contradiction. If we had $\spadesuit \in T$ and $\clubsuit \notin T$, then $\{\clubsuit, \diamondsuit\} \subseteq \text{res}_{\mathcal{A}}(T) = T$ by reaction (35), a contradiction. If we had $\spadesuit \in T$ and $\clubsuit \in T$, no reaction that can produce $\clubsuit$ would be enabled (reactions (33), (34), (35)), a contradiction. Therefore, it must hold $\diamondsuit \notin T$ and $\spadesuit \notin T$; if in addition we had $\clubsuit \notin T$, then reaction (35) would be enabled and thus we would have $\{\clubsuit, \diamondsuit, \spadesuit\} \subseteq \text{res}_{\mathcal{A}}(T) = T$, a contradiction. For a visual representation of the dynamics of reactions (33), (34), (35) see Figure 10. The statement follows. ■

Claim 2. The fixed points of $\mathcal{A}$ are the states of type

$$T_U := C \cup \heartsuit_S \cup U \cup \overline{V_1 \setminus U} \cup \{\clubsuit\}, \tag{36}$$

for any $U \subseteq V_1$.

Proof. Let T be a fixed point for $\mathcal{A}$; by Claim 1 and looking at the products of the reactions it must be $T \subseteq C \cup \heartsuit_S \cup V_1 \cup \overline{V_1} \cup \{\clubsuit\}$ with $\clubsuit \in T$. Moreover, if we

had $C \cap T \subsetneq C$ then at least one of the reactions of group (28) would be enabled, thus it would be $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T) = T$, a contradiction. If we had $\heartsuit_S \cap T \subsetneq \heartsuit_S$ then at least one of the reactions of group (29) would be enabled, thus we would have $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T) = T$, a contradiction. By reactions (30), (31) and (27), arguing as in the proof of Theorem 3.3.1, we get that T must be a well-formed state for $V_1 \cup \overline{V_1}$, i.e. $x \in T \cap V_1$ if and only if $\overline{x} \notin T \cap \overline{V_1}$. Therefore, T is of type (36) with $U = T \cap V_1$. Finally, we can immediately check that if a state is of type (36), then it is a fixed point because of reactions (30), (31), (32) and (33). ■

We are now interested in studying when a fixed point T_U, for a given $U \subseteq V_1$, is an attractor. Let $T' \neq T_U$ such that $\mathrm{res}_{\mathcal{A}}(T') = T_U$. If it was $C \cap T' \subsetneq C$ then at least one of the reactions of group (28) would be enabled, thus it would be $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T') = T$, a contradiction; and if it was $\heartsuit_S \cap T' \subsetneq \heartsuit_S$ then at least one of the reactions of group (29) would be enabled, thus it would be $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T') = T$, a contradiction. Therefore it must be $C \cup \heartsuit_S \subseteq T'$.

Claim 3. If $T' \neq T_U$ and $\mathrm{res}_{\mathcal{A}}(T') = T_U$ then $\clubsuit \notin T'$, $\diamondsuit \in T'$ and $\spadesuit \notin T'$.

Proof. If we had $\spadesuit \in T'$ then reactions (22), (23) and (32) would not be enabled, thus we would have $C \cap \mathrm{res}_{\mathcal{A}}(T') = \varnothing$, a contradiction since $C \subseteq T_U = \mathrm{res}_{\mathcal{A}}(T')$. Therefore we obtain $\spadesuit \notin T'$. If we had $\clubsuit \in T'$ we divide two cases:

- if $\diamondsuit \in T'$ then none of the reactions (33), (34), (35) would be enabled, therefore we would have $\clubsuit \notin \mathrm{res}_{\mathcal{A}}(T') = T_U$, a contradiction.
- if $\diamondsuit \notin T'$, the only way to get $C \subseteq \mathrm{res}_{\mathcal{A}}(T')$ would be with reaction (32), since (22), (23) are not enabled if $\clubsuit \in T'$. Therefore to enable (32), we must have $T' \subseteq C \cup \heartsuit_S \cup \{\clubsuit\} \cup V_1 \cup \overline{V_1}$, and since $C \cup \heartsuit_S \subseteq T'$, as previously remarked, we obtain $T' = C \cup \heartsuit_S \cup \{\clubsuit\} \cup U_1 \cup \overline{U_2}$ for some $U_1 \subseteq V_1$ and $U_2 \subseteq V_1$. Since $\mathrm{res}_{\mathcal{A}}(T') = T_U$ contains a well-formed state for $V_1 \cup \overline{V_1}$, then $U_1 \cup \overline{U_2}$ must be a well-formed state for $V_1 \cup \overline{V_1}$. In this case we would obtain $\mathrm{res}_{\mathcal{A}}(T') = C \cup \heartsuit_S \cup \{\clubsuit\} \cup U_1 \cup \overline{U_2}$, hence we would have $U_1 = U$, $U_2 = V \setminus U$ and thus $T' = T_U$, a contradiction.

Therefore $\clubsuit \notin T'$. Finally, if $\diamondsuit \notin T'$ then $\{\clubsuit, \diamondsuit, \spadesuit\} \subseteq \mathrm{res}_{\mathcal{A}}(T') = T_U$ (35), a contradiction. The statement follows. ■

We obtain that if $T' \neq T_U$ is an attractor for T_U then it must be of the form $T' = C \cup \heartsuit_S \cup \{\diamondsuit\} \cup X_1 \cup \overline{X_2}$ for some $X_1 \subseteq V$ and $X_2 \subseteq V$: this follows from $C \cup \heartsuit_S \subseteq T'$ and from Claim 3. In this case, reaction (32) is not enabled, thus the fact that $C \cup \heartsuit_S \subseteq \mathrm{res}_{\mathcal{A}}(T') = T_U$ will only depend on how $X_1 \cup \overline{X_2}$ is constructed. We consider two cases:

- if $\exists x_i, \overline{x_i} \in X_1 \cup \overline{X_2}$ then $\heartsuit_i \notin \mathrm{res}_{\mathcal{A}}(T')$, since neither the i-th reaction of (25) nor the i-th reaction of (26) are enabled;
- if $\exists x_i, \overline{x_i} \notin X_1 \cup \overline{X_2}$ then $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T')$, since the i-th reaction in (27) is enabled.

Therefore $X_1 \cup \overline{X_2}$ must be a well-formed state for $V \cup \overline{V}$, and thus $X_2 = V \setminus X_1$. We now prove that T' being an attractor for T_U implies satisfiability of φ, obtained via an assignment that extends the assignment $U \cup \overline{V_1 \setminus U}$ of the variables in V_1. We can

interpret $X_1 \cup \overline{V \setminus X_1}$ as an assignment for φ, as in the proof of Theorem 3.3.1. If we had $X_1 \cup \overline{V \setminus X_1} \not\models \varphi$, then one of the reactions (24) would be enabled, thus we would have $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T')$, a contradiction. Therefore, if T' is an attractor for T_U it must hold $X_1 \cup \overline{V \setminus X_1} \models \varphi$, thus all clauses are satisfied, i.e. each $\varphi_j \in C$ is generated by at least one reaction of type (22) or (23). We obtain, using also reactions (30) and (31),

$$\mathrm{res}_{\mathcal{A}}(T') = C \cup \heartsuit_S \cup \{\clubsuit\} \cup \{x : x \in X_1 \cap V_1\} \cup \{\overline{x} : \overline{x} \in \overline{X_2} \cap \overline{V_1}\} \tag{37}$$

Since $\mathrm{res}_{\mathcal{A}}(T') = T_U$, comparing (37) and the definition of T_U (36) we find that $U = X_1 \cap V_1$ and $\overline{V_1 \setminus U} = \overline{V \setminus X_1} \cap \overline{V_1}$. This means that if T_U is an attractor there exists an assignment $X_1 \cup \overline{V \setminus X_1}$ satisfying φ where all variables in U are set to true and all variables in $V_1 \setminus U$ are set to false. The converse is also true, i.e. if there exists an assignment $X \cup \overline{V \setminus X}$ such that $X_1 \cap V_1 = U$ and φ is satisfied then T_U is reached by $C \cup \heartsuit_S \cup \{\diamondsuit\} \cup X \cup \overline{V \setminus X}$. Therefore, we proved that any assignment for the variables in V_1 can be extended to a complete assignment that satisfies φ if and only if all fixed points of $\mathcal{A}$ are attractors. Or in other words, all the fixed points of $\mathcal{A}$ are attractors if and only if $(\forall V_1)(\exists V_2)\varphi$ is valid.

The mapping $\varphi \mapsto \mathcal{A}$ is computable in polynomial time, hence deciding if all the fixed points of $\mathcal{A}$ are attractors is $\Pi_2^{\mathbf{P}}$-hard. Therefore, deciding if $\mathcal{A}$ has a fixed point which is not attractor is $\Sigma_2^{\mathbf{P}}$-hard. □

Corollary 3.3.6. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(0, \infty)$ with a common background set S, it is $\Sigma_2^{\mathbf{P}}$-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point which is not an attractor.*

Proof. The problem is in $\Sigma_2^{\mathbf{P}}$ (see Table 4); by Theorem 3.3.5, when $\mathcal{A} = \mathcal{B}$ the problem is $\Sigma_2^{\mathbf{P}}$-complete. □

Since the problems are $\Sigma_2^{\mathbf{P}}$-complete for $\mathcal{RS}(0, \infty)$, we have that they are also $\Sigma_2^{\mathbf{P}}$-complete for $\mathcal{RS}(\infty, \infty)$, as stated by the following corollaries.

Corollary 3.3.7. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, \infty)$, it is $\Sigma_2^{\mathbf{P}}$-complete to decide if $\mathcal{A}$ has a fixed point which is not an attractor.*

Corollary 3.3.8. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, \infty)$ with a common background set S, it is $\Sigma_2^{\mathbf{P}}$-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point which is not an attractor.*

We now study the problem of deciding if two reaction systems share all their fixed point attractors.

Corollary 3.3.9. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(0, \infty)$ with a common background set S, it is $\Pi_2^{\mathbf{P}}$-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all their fixed point attractors.*

Proof. The problem is in $\Pi_2^{\mathbf{P}}$ (see Table 4). In order to show $\Pi_2^{\mathbf{P}}$-hardness, we reduce $\forall\exists$SAT [249] to this problem. Consider the RS $\mathcal{A}$ in the proof of Theorem 3.3.5; we just need to construct a RS $\mathcal{B}$ over the same background set as $\mathcal{A}$ such that all the fixed points attractors of $\mathcal{B}$ are of the form T_U for any $U \subseteq V_1$. Therefore we define the reactions of $\mathcal{B}$ to be the reactions (28), (29), (30), (31), (32), (33), (34) and (35) from the proof of Theorem 3.3.5. We remark that Claims 1 and 2 can be applied to $\mathcal{B}$, therefore all the fixed points of $\mathcal{B}$ are the states T_U for any $U \subseteq V_1$. Furthermore, all the fixed points of $\mathcal{B}$ are attractors and coincide with the ones of $\mathcal{A}$. The mapping $\varphi \mapsto (\mathcal{A}, \mathcal{B})$ is computable in polynomial time, hence deciding if two RSs share all their fixed point attractors is $\Pi_2^{\mathbf{P}}$-hard. □

We now study the problem of deciding whether two RSs share all their fixed points. The problem is **coNP**-complete for $\mathcal{RS}(\infty,\infty)$ [123, Theorem 3], and this is also true for reactantless RSs, as proved in the following theorem.

Theorem 3.3.10. *Given $\mathcal{A},\mathcal{B} \in \mathcal{RS}(0,\infty)$ with a common background set S, it is* **coNP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all their fixed points.*

Proof. The problem lies in **coNP** (see Table 4). In order to show **coNP**-completeness, we reduce VALIDITY [214] to this problem. Given a Boolean formula $\varphi = \varphi_1 \vee \cdots \vee \varphi_m$ in DNF over the variables $V = \{x_1,\dots,x_n\}$, we define a RS $\mathcal{A}$ with background set $S \coloneqq V \cup \overline{V} \cup \{\heartsuit\} \cup \{\clubsuit\}$ (where the sets are as in Definition 3.1.1) and the reactions:

$$(\varnothing, \mathrm{neg}(\varphi_j) \cup \overline{\mathrm{pos}}(\varphi_j) \cup \{\clubsuit\}, \{\heartsuit\}) \qquad \text{for } 1 \leqslant j \leqslant m \tag{38}$$
$$(\varnothing, \{x_i\}, \{\overline{x_i}\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{39}$$
$$(\varnothing, \{\overline{x_i}\}, \{x_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{40}$$
$$(\varnothing, \{\heartsuit\}, \{\heartsuit, \clubsuit\}). \tag{41}$$

As in the proof of Theorem 3.3.1, if T is not a well-formed state then T is not a fixed point. Reactions of type (38) evaluate each conjunctive clause (which is satisfied if and only if no positive variables are set to false and no negative ones to true) and generate $\heartsuit$ when φ itself is satisfied by $T \cap (V \cup \overline{V})$. Consider the dynamic of the reaction system restricted to a well-formed state $Y \subseteq V \cup \overline{V}$. If Y does not satisfy φ there are no fixed points since:

$$Y \cup \{\heartsuit\} \longrightarrow Y \qquad Y \cup \{\heartsuit, \clubsuit\} \longleftarrow Y \cup \{\clubsuit\}$$

where the arrow represent the function $\mathrm{res}_{\mathcal{A}}$. Instead, if Y satisfies φ, $Y \cup \{\heartsuit\}$ is a fixed point since:

$$Y \cup \{\heartsuit\} \qquad Y \qquad Y \cup \{\heartsuit, \clubsuit\} \longleftarrow Y \cup \{\clubsuit\}.$$

Finally, the fixed points of $\mathcal{A}$ are the well-formed states $Y \cup \{\heartsuit\}$ such that $Y \vDash \varphi$. Now, let $\mathcal{B}$ be defined by the following reactions:

$$(\varnothing, \{x_i\}, \{\overline{x_i}\}) \qquad \text{for } 1 \leqslant i \leqslant n$$
$$(\varnothing, \{\overline{x_i}\}, \{x_i\}) \qquad \text{for } 1 \leqslant i \leqslant n$$
$$(\varnothing, \{\clubsuit\}, \{\heartsuit\})$$
$$(\varnothing, \{\heartsuit\}, \{\heartsuit, \clubsuit\}).$$

In a similar way as above, the fixed points of $\mathcal{B}$ are the states $Y \cup \{\heartsuit\}$ where $Y \subseteq V \cup \overline{V}$ is well-formed. We can conclude that $\mathcal{A}$ and $\mathcal{B}$ share all fixed points exactly when all assignments satisfy φ, i.e. φ is a tautology. Since the mapping $\varphi \mapsto (\mathcal{A},\mathcal{B})$ is computable in polynomial time, the problem is **coNP**-hard. □

Note that in the proof of Theorem 3.3.10, the fixed points of $\mathcal{A}$ and $\mathcal{B}$ are not attractors: this implies the following result.

Corollary 3.3.11. *Given $\mathcal{A},\mathcal{B} \in \mathcal{RS}(0,\infty)$ with a common background set S, it is* **coNP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all their fixed points which are not attractors.*

Since the problem is **coNP**-complete for $\mathcal{RS}(0,\infty)$, we have that it is also **coNP**-complete for $\mathcal{RS}(\infty,\infty)$, as stated by the following corollary.

Corollary 3.3.12. *Given $\mathcal{A},\mathcal{B} \in \mathcal{RS}(\infty,\infty)$ with a common background set S, it is* **coNP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all their fixed points which are not attractors.*

3.4 FIXED POINTS FOR INHIBITORLESS RSS

In this section, we prove **NP**-hardness and **coNP**-hardness for problems of fixed points in the class of inhibitorless RSs.

The problem of deciding the existence of a fixed point is entirely trivial for $\mathcal{RS}(\infty,0)$ thanks to the Knaster-Tarski theorem, as first remarked in [181]. On the contrary, the following theorem shows that it is **NP**-complete to decide whether two inhibitorless RSs have a common fixed point.

Theorem 3.4.1. *Given $\mathcal{A},\mathcal{B} \in \mathcal{RS}(\infty,0)$ with a common background set S, it is* **NP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point.*

Proof. The problem is in **NP** (see Table 4). In order to show **NP**-hardness, we reduce SAT [214] to this problem. Given a Boolean formula $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_m$ in CNF over the variables $V = \{x_1,\ldots,x_n\}$, let $\heartsuit_S \coloneqq \{\heartsuit_i : 1 \leqslant i \leqslant n\}$ be a set of extra entities that are not contained in $V \cup \overline{V}$. We define a RS $\mathcal{A}$ with background set $S \coloneqq V \cup \overline{V} \cup \heartsuit_S \cup \{\spadesuit\}$ (with $\spadesuit \notin V \cup \overline{V} \cup \heartsuit_S$ and $V, \overline{V}$ as in Definition 3.1.1) and the reactions

$$(\mathrm{neg}(\varphi_j) \cup \overline{\mathrm{pos}}(\varphi_j) \cup \heartsuit_S, \varnothing, \{\spadesuit\}) \qquad \text{for } 1 \leqslant j \leqslant m \tag{42}$$
$$(\{x_i\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, x_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{43}$$
$$(\{\overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, \overline{x_i}\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{44}$$
$$(\{x_i, \overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\spadesuit\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{45}$$
$$(\{\spadesuit\} \cup \heartsuit_S, \varnothing, \{\spadesuit\}). \tag{46}$$

Note that for all $Y \subseteq V \cup \overline{V}$, and for every $Z_\heartsuit \subsetneq \heartsuit_S$ it holds

$$\mathrm{res}_{\mathcal{A}}(Y \cup Z_\heartsuit) = \mathrm{res}_{\mathcal{A}}(Y \cup Z_\heartsuit \cup \{\spadesuit\}) = \varnothing = \mathrm{res}_{\mathcal{A}}(\varnothing) \tag{47}$$

because no reaction is enabled. We thus consider states $T \subseteq S$ such that $\heartsuit_S \subseteq T$. For every $Y \subseteq V \cup \overline{V}$, we define $\heartsuit_Y \coloneqq \{\heartsuit_i : x_i \in Y \vee \overline{x_i} \in Y\} \subseteq \heartsuit_S$. Note that $\heartsuit_Y = \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S) \cap \heartsuit_S = \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S \cup \{\spadesuit\}) \cap \heartsuit_S$, so when $\heartsuit_Y \subsetneq \heartsuit_S$, the states $Y \cup \heartsuit_S$ and $Y \cup \heartsuit_S \cup \{\spadesuit\}$ reach $\varnothing$ in two steps. In particular, if $T \neq \varnothing$ is a fixed point then it must be of the form $T = Y \cup \heartsuit_Y$ or $T = Y \cup \heartsuit_Y \cup \{\spadesuit\}$ with $Y \subseteq V \cup \overline{V}$ and $\heartsuit_Y = \heartsuit_S$. We remark that $\heartsuit_Y = \heartsuit_S$ means that $x_i \in Y$ or $\overline{x_i} \in Y$ for all $1 \leqslant i \leqslant n$. We divide two cases:

(i) Y is not a well-formed state. Since $\heartsuit_Y = \heartsuit_S$, there exist $x_i, \overline{x_i} \in Y$, so $\spadesuit$ is generated by one of the reactions of type (45). We obtain that $Y \cup \heartsuit_S \cup \{\spadesuit\}$ is a fixed point reachable from $Y \cup \heartsuit_S$.

(ii) Y is a well-formed state. If $Y \vDash \varphi$ then no reaction of type (42) is enabled, so $Y \cup \heartsuit_S$ is a fixed point (not reachable from any other state). Also in this case, $Y \cup \heartsuit_S \cup \{\spadesuit\}$ is a fixed point, thanks to reaction (46). On the other hand, if $Y \nvDash \varphi$ then $Y \cup \heartsuit_S \cup \{\spadesuit\}$ is a fixed point reachable from $Y \cup \heartsuit_S$.

Now, consider the RS $\mathcal{B}$ given by the following reactions:

$$(\varnothing, \varnothing, \heartsuit_S) \tag{48}$$
$$(\{x_i\}, \varnothing, \{x_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{49}$$
$$(\{\overline{x_i}\}, \varnothing, \{\overline{x_i}\}) \qquad \text{for } 1 \leqslant i \leqslant n. \tag{50}$$

The fixed points of $\mathcal{B}$ are the states $Y \cup \heartsuit_S$ for all $Y \subseteq V \cup \overline{V}$. We can conclude that $\mathcal{A}$ and $\mathcal{B}$ share a fixed point exactly when there exists an assignment satisfying φ, i.e. φ is satisfiable. Since the mapping $\varphi \mapsto (\mathcal{A}, \mathcal{B})$ is computable in polynomial time, the problem is **NP**-hard. □

We next show that determining whether two inhibitorless RSs have a common fixed point attractor is **NP**-complete. The proof is an adaptation of the proof of Theorem 3.4.1.

Corollary 3.4.2. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, 0)$ with a common background set S, it is* **NP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point attractor.*

Proof. The problem is in **NP** (see Table 4). Following the proof of Theorem 3.4.1, we just need to ensure that the fixed points of $\mathcal{A}$ are attractors, so we delete reaction (46) from the reactions of $\mathcal{A}$. In this way, when $Y \subsetneq V \cup \overline{V}$ and $Y \vDash \varphi$, we have that

$$\mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S \cup \{\spadesuit\}) = Y \cup \heartsuit_S = \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S),$$

thus $Y \cup \heartsuit_S$ is a fixed point reachable from $Y \cup \heartsuit_S \cup \{\spadesuit\}$. We can conclude that $\mathcal{A}$ and $\mathcal{B}$ share a fixed point attractor exactly when there exists an assignment satisfying φ, i.e. φ is satisfiable. Since the mapping $\varphi \mapsto (\mathcal{A}, \mathcal{B})$ is computable in polynomial time, the problem is **NP**-hard. □

We now consider the problem of deciding if two RSs share all their fixed points and prove that, like in the case of reactantless RSs, this problem is **coNP**-complete.

Theorem 3.4.3. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, 0)$ with a common background set S, it is* **coNP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all their fixed points.*

Proof. The problem lies in **coNP** (see Table 4). In order to show **coNP**-completeness, we reduce VALIDITY [214] to this problem. Given a Boolean formula $\varphi = \varphi_1 \vee \cdots \vee \varphi_m$ in DNF over the variables $V = \{x_1, \ldots, x_n\}$, let $\heartsuit_S \coloneqq \{\heartsuit_i : 1 \leqslant i \leqslant n\}$ be a set of extra entities. We define a RS $\mathcal{A}$ with background set $S \coloneqq V \cup \overline{V} \cup \heartsuit_S \cup \{\heartsuit\}$ (with $\heartsuit \notin V \cup \overline{V} \cup \heartsuit_S$ and $V, \overline{V}$ as in Definition 3.1.1) and the reactions

$$(\overline{\mathrm{neg}}(\varphi_j) \cup \mathrm{pos}(\varphi_j) \cup \heartsuit_S \cup \{\heartsuit\}, \varnothing, \{\heartsuit\}) \qquad \text{for } 1 \leqslant j \leqslant m \tag{51}$$
$$(\{x_i\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, x_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{52}$$
$$(\{\overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, \overline{x_i}\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{53}$$
$$(\{x_i, \overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\heartsuit\}) \qquad \text{for } 1 \leqslant i \leqslant n. \tag{54}$$

For every $Y \subseteq V \cup \overline{V}$, we define $\heartsuit_Y \coloneqq \{\heartsuit_i : x_i \in Y \vee \overline{x_i} \in Y\} \subseteq \heartsuit_S$. As in the proof of Theorem 3.4.1, if $T \neq \varnothing$ is a fixed point then it must be of the form $T = Y \cup \heartsuit_Y$ or $T = Y \cup \heartsuit_Y \cup \{\heartsuit\}$ with $\heartsuit_Y = \heartsuit_S$. We divide two cases:

(i) Y is not a well-formed state. Then, since $\heartsuit_Y = \heartsuit_S$, there exist $x_i, \overline{x_i} \in Y$, so $\heartsuit$ is generated by one of the reactions of type (54). We get that $Y \cup \heartsuit_Y \cup \{\heartsuit\}$ is a fixed point reachable from $Y \cup \heartsuit_Y$.

(ii) Y is a well-formed state. Then, if $Y \vDash \varphi$, a reaction of type (51) is enabled by $Y \cup \heartsuit_Y \cup \{\heartsuit\}$, so $Y \cup \heartsuit_Y \cup \{\heartsuit\}$ is a fixed point (not reachable from any other state). In this case, also $Y \cup \heartsuit_S$ is a fixed point since reactions of type (51) are not enabled. On the other hand, if $Y \nvDash \varphi$ then $Y \cup \heartsuit_S$ is a fixed point reachable from $Y \cup \heartsuit_S \cup \{\heartsuit\}$.

Now, consider the RS $\mathcal{B}$ given by the following reactions:

$$(\{\heartsuit\} \cup \heartsuit_S, \varnothing, \{\heartsuit\}) \tag{55}$$
$$(\{x_i\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, x_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{56}$$
$$(\{\overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, \overline{x_i}\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{57}$$
$$(\{x_i, \overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\heartsuit\}) \qquad \text{for } 1 \leqslant i \leqslant n. \tag{58}$$

With a similar analysis as above, for every well-formed state $Y \subseteq V \cup \overline{V}$ the states $Y \cup \heartsuit_S$, $Y \cup \heartsuit_Y \cup \{\heartsuit\}$ are fixed points (not attractors), and for every not-well-formed state $Y \subseteq V \cup \overline{V}$ such that $\heartsuit_Y = \heartsuit_S$ the state $Y \cup \heartsuit_Y \cup \{\heartsuit\}$ is a fixed point reachable from $Y \cup \heartsuit_S$. We can conclude that $\mathcal{A}$ and $\mathcal{B}$ share all fixed points exactly when all assignments satisfy φ, i.e. φ is a tautology. Since the mapping $\varphi \mapsto (\mathcal{A}, \mathcal{B})$ is computable in polynomial time, the problem is **coNP**-hard. □

Corollary 3.4.4. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, 0)$ with a common background set S, it is* **coNP***-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all their fixed points that are not attractors.*

Proof. The problem is in **coNP** (see Table 4). The **coNP**-hardness follows from the same construction of Theorem 3.4.3. □

In contrast, deciding whether two inhibitorless RSs share all their fixed points that are attractors is $\Pi_2^{\mathbf{P}}$-complete.

Theorem 3.4.5. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, 0)$ with a common background set S, it is $\Pi_2^{\mathbf{P}}$-complete to decide if $\mathcal{A}$ and $\mathcal{B}$ share all their fixed point attractors.*

Proof. The problem is in $\Pi_2^{\mathbf{P}}$ (see Table 4). In order to show $\Pi_2^{\mathbf{P}}$-hardness, we reduce $\forall\exists$SAT [249] to this problem. Given a quantified Boolean formula $(\forall V_1)(\exists V_2)\varphi$ over $V = \{x_1, \ldots, x_n\}$, with $V_1 \subseteq V$, $V_2 = V \setminus V_1$ and $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_m$ quantifier-free and in CNF, let $V_1' = \{x' : x \in V_1\}$ and $\heartsuit_S \coloneqq \{\heartsuit_i : 1 \leqslant i \leqslant n\}$ be extra sets of entities.

We define $\mathcal{A}$ a RS with background set $S \coloneqq V \cup \overline{V} \cup V_1' \cup C \cup \heartsuit_S \cup \{\clubsuit, \spadesuit\}$ (see also Definition 3.1.1) and the reactions

$$(\{x\}, \varnothing, \{\varphi_j\}) \qquad \text{for } 1 \leqslant j \leqslant m \text{ and } x \in \mathrm{pos}(\varphi_j) \tag{59}$$
$$(\{\overline{x}\}, \varnothing, \{\varphi_j\}) \qquad \text{for } 1 \leqslant j \leqslant m \text{ and } \overline{x} \in \overline{\mathrm{neg}}(\varphi_j) \tag{60}$$
$$(\mathrm{neg}(\varphi_j) \cup \overline{\mathrm{pos}}(\varphi_j), \varnothing, \{\spadesuit\}) \qquad \text{for } 1 \leqslant j \leqslant m \tag{61}$$
$$(\{x_i\}, \varnothing, \{\heartsuit_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{62}$$
$$(\{\overline{x_i}\}, \varnothing, \{\heartsuit_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{63}$$
$$(\{x_i, \overline{x_i}\}, \varnothing, \{\clubsuit\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{64}$$
$$(\{x\}, \varnothing, \{x'\}) \qquad \text{for } x \in V_1 \tag{65}$$
$$(C \cup \heartsuit_S, \varnothing, C \cup \heartsuit_S) \tag{66}$$
$$(C \cup \heartsuit_S \cup \{x'\}, \varnothing, \{x'\}) \qquad \text{for } x' \in V_1' \tag{67}$$
$$(C \cup \heartsuit_S \cup \{x_i\}, \varnothing, \{\clubsuit\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{68}$$
$$(C \cup \heartsuit_S \cup \{\overline{x_i}\}, \varnothing, \{\clubsuit\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{69}$$
$$(\{\clubsuit\}, \varnothing, \{\clubsuit\}) \tag{70}$$
$$(\{\spadesuit\}, \varnothing, \{\clubsuit\}). \tag{71}$$

We first note that for any $T \subseteq S \setminus \{\clubsuit\}$ we have $\mathrm{res}_{\mathcal{A}}(T \cup \{\clubsuit\}) = \mathrm{res}_{\mathcal{A}}(T) \cup \{\clubsuit\}$. We start by determining the fixed points of $\mathcal{A}$. We notice that if T is a fixed point then it must be $T = \mathrm{res}_{\mathcal{A}}(T) \subseteq C \cup \heartsuit_S \cup V_1' \cup \{\clubsuit, \spadesuit\}$, because this is the union of the products of all reactions. Furthermore, the only way for $\spadesuit$ to be part of the product is through reactions (61) which use reactants in $V \cup \overline{V}$ that are never part of a fixed point, as we already noticed.

For the same reason, the only reactions that can give rise to a fixed point are (66), (67), (70) and thus we deduce that all the fixed points of $\mathcal{A}$ are $\varnothing$, $\{\clubsuit\}$, and those of the form $C \cup \heartsuit_S \cup U \cup \{\clubsuit\}$ or $C \cup \heartsuit_S \cup U$, with $U \subseteq V_1'$. The states $\varnothing$, $\{\clubsuit\}$ and $C \cup \heartsuit_S \cup U \cup \{\clubsuit\}$ are all attractors, so we now focus on understanding when a state of the form $C \cup \heartsuit_S \cup U$ is a fixed point attractor for a given $U \subseteq V_1'$.

Let $T \neq C \cup \heartsuit_S \cup U$ be a state such that $\mathrm{res}_{\mathcal{A}}(T) = C \cup \heartsuit_S \cup U$. Since $\clubsuit \notin \mathrm{res}_{\mathcal{A}}(T)$, neither (70) or (71) can be enabled, thus $\clubsuit \notin T$ and $\spadesuit \notin T$. We obtain that T is of the form

$$T = T_V \cup T_{\overline{V}} \cup T_{V_1'} \cup T_C \cup T_{\heartsuit_S}$$

where $T_V \coloneqq T \cap V$ and analogously for $T_{\overline{V}}$, $T_{V_1'}$, T_C and $T_{\heartsuit_S}$. If we had $T_C = C$ and $T_{\heartsuit_S} = \heartsuit_S$, then we would also have $T_V = T_{\overline{V}} = \varnothing$ as otherwise $\clubsuit \in \mathrm{res}_{\mathcal{A}}(T)$ would be generated by at least one of reactions of type (68) or (69). Thus, in this case, we would have $C \cup \heartsuit_S \cup U = \mathrm{res}_{\mathcal{A}}(T) = C \cup \heartsuit_S \cup T_{V_1'} = T$, which is a contradiction. Therefore it must hold $T_C \subsetneq C$ and $T_{\heartsuit_S} \subsetneq \heartsuit_S$, and, collecting all together, the reactions of type (66), (67), (68), (69), (70) and (71) are not enabled.

We further remark that as in the previous proofs, $T_V \cup T_{\overline{V}}$ must be a well-formed assignment for φ. Furthermore if $T_V \cup T_{\overline{V}} \vDash \varphi$ then $\mathrm{res}_{\mathcal{A}}(T) = C \cup \heartsuit_S \cup \{x' : x \in T_V \cap V_1\}$. We can also remark that $\spadesuit \in \mathrm{res}_{\mathcal{A}}(T)$ if and only if there exists a $\varphi_j \in C$ that is not satisfied by the assignment corresponding to $T_V \cup T_{\overline{V}}$. We deduce that $C \cup \heartsuit_S \cup U$ is a fixed point attractor if and only if there exists an assignment $X \cup \overline{V \setminus X}$ such that $X \cap V_1 = \{x : x' \in U\}$ and $X \cup \overline{V \setminus X} \vDash \varphi$, i.e. an assignment satisfying φ

that extends the assignment for the variables in V_1 in which the only variables set to true are $\{x : x' \in U\}$. Therefore, we proved that any assignment for the variables in V_1 can be extended to a complete assignment that satisfies φ if and only if all fixed points of $\mathcal{A}$ are attractors. We conclude that all fixed points of $\mathcal{A}$ are attractors if and only $(\forall V_1)(\exists V_2)\varphi$ is valid.

We now finally consider a RS $\mathcal{B}$ with background set S and reactions (66), (67), (70). All the fixed points of $\mathcal{B}$ are attractors and coincide with the ones of $\mathcal{A}$. The mapping $\varphi \mapsto (\mathcal{A}, \mathcal{B})$ is computable in polynomial time, hence deciding if two RSs share all their fixed point attractors is $\Pi_2^{\mathbf{P}}$-hard. □

Corollary 3.4.6. *Given $\mathcal{A} \in \mathcal{RS}(\infty, 0)$ it is $\Sigma_2^{\mathbf{P}}$-complete to decide whether $\mathcal{A}$ has a fixed point which is not an attractor.*

Proof. The problem is in $\Sigma_2^{\mathbf{P}}$ (see Table 4). Consider the converse problem, i.e. deciding if all fixed points are attractors. The $\Pi_2^{\mathbf{P}}$-hardness of the latter follows from the construction of the RS $\mathcal{A}$ in the proof of Theorem 3.4.5. Therefore our problem is $\Sigma_2^{\mathbf{P}}$-complete. □

Corollary 3.4.7. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, 0)$ with a common background set S, it is $\Sigma_2^{\mathbf{P}}$-complete to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point which is not an attractor.*

Proof. The problem is in $\Sigma_2^{\mathbf{P}}$ (see Table 4); by Corollary 3.4.6, when $\mathcal{A} = \mathcal{B}$ the problem is $\Sigma_2^{\mathbf{P}}$-complete. □

3.5 EQUAL RESULT FUNCTION

In this section, we study the problem of deciding if two RSs have the same result function. This problem lies in **coNP** and is complete in general RS.

Theorem 3.5.1. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, \infty)$ with the same background set S, it is* **coNP***-complete to decide whether $\mathrm{res}_{\mathcal{A}} = \mathrm{res}_{\mathcal{B}}$.*

Proof. The problem lies in **coNP** (see Table 4). In order to show **coNP**-completeness, we reduce VALIDITY [214] to this problem. Given a Boolean formula $\varphi = \varphi_1 \vee \cdots \vee \varphi_m$ in DNF over the variables $V = \{x_1, \ldots, x_n\}$, build the RS $\mathcal{A}$ consisting of the background set $S := V \cup \{\heartsuit\}$ (with $\heartsuit \notin V$, and $\mathrm{pos}(\varphi_r) \subseteq V$ and $\mathrm{neg}(\varphi_r) \subseteq V$ the set of variables that occur non-negated and negated in φ_r, respectively) and the following reactions:

$$(\mathrm{pos}(\varphi_j), \mathrm{neg}(\varphi_j), \{\heartsuit\}) \qquad \text{for } 1 \leqslant j \leqslant m. \tag{72}$$

For any state $T \subseteq S$, $T \cap V$ encodes a truth assignment of φ. In this way, the reactions of type (72) evaluate each conjunctive clause and produce the element $\heartsuit$ if the clause (and thus the whole formula φ) is satisfied. Then the RS behaves as follows:

$$\mathrm{res}_{\mathcal{A}}(T) = \begin{cases} \heartsuit & \text{if } T \cap V \vDash \varphi \\ \varnothing & \text{if } T \cap V \nvDash \varphi. \end{cases}$$

Now let $\mathcal{B}$ be the constant RS defined by the reaction $(\varnothing, \varnothing, \{\heartsuit\})$ alone.

By construction, $\mathrm{res}_{\mathcal{A}} = \mathrm{res}_{\mathcal{B}}$ when all assignments satisfy φ. Since the map $\varphi \mapsto (\mathcal{A}, \mathcal{B})$ is computable in polynomial time, deciding if $\mathrm{res}_{\mathcal{A}} = \mathrm{res}_{\mathcal{B}}$ is **coNP**-hard. □

Since in the previous proof the RS $\mathcal{B}$ is constant, the problem of deciding if the result function of a RS is a non-empty constant is **coNP**-complete in unconstrained RSs. In contrast, deciding if a result function is empty can be done in polynomial time.

Corollary 3.5.2. *Given $\mathcal{A} \in \mathcal{RS}(\infty, \infty)$, it is* **coNP***-complete to decide if $\mathrm{res}_{\mathcal{A}}$ is a non-empty constant function; however, deciding if $\mathrm{res}_{\mathcal{A}} = \varnothing$ is in* **P**.

Proof. The first part of the statement follows directly from the proof of Theorem 3.5.1. Note that given a RS $\mathcal{A} = (S, A)$ and a reaction $a = (R_a, I_a, P_a) \in A$, there exists a state $T \subseteq S$ that enables a if and only if $R_a \cap I_a = \varnothing$. Since $\mathrm{res}_{\mathcal{A}} = \varnothing$ if and only if any reaction is not enabled by all states, we just need to check that $R_a \cap I_a \neq \varnothing$ for all $a \in A$. □

We remark that the same problem for Boolean Automata Network in parallel mode is **coNP**-complete for any fixed output value.

We now study the same problem for inhibitorless RSs.

Proposition 3.5.3. *Given $\mathcal{A} = (S, A), \mathcal{B} = (S, B) \in \mathcal{RS}(\infty, 0)$, if*

$$\mathrm{res}_{\mathcal{A}}(R_a) \subseteq \mathrm{res}_{\mathcal{B}}(R_a) \quad \forall a \in A, \tag{73}$$

then $\mathrm{res}_{\mathcal{A}}(T) \subseteq \mathrm{res}_{\mathcal{B}}(T)$ for all states $T \subseteq S$.

Proof. Let $T \subseteq S$ such that $T \neq R_a$ for all $a \in A$. By definition we have

$$\mathrm{res}_{\mathcal{A}}(T) = \bigcup_{a \in A : R_a \subsetneq T} \mathrm{res}_a(T) = \bigcup_{a \in A : R_a \subsetneq T} \mathrm{res}_a(R_a) = \bigcup_{a \in A : R_a \subsetneq T} \mathrm{res}_{\mathcal{A}}(R_a).$$

By monotonicity of $\mathrm{res}_{\mathcal{B}}$, if $R_a \subseteq T$ then $\mathrm{res}_{\mathcal{B}}(R_a) \subseteq \mathrm{res}_{\mathcal{B}}(T)$, so using (73) we obtain

$$\mathrm{res}_{\mathcal{A}}(T) \subseteq \bigcup_{a \in A : R_a \subsetneq T} \mathrm{res}_{\mathcal{B}}(R_a) \subseteq \mathrm{res}_{\mathcal{B}}(T)\,. \qquad \square$$

Corollary 3.5.4. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, 0)$ with a common background set S, it is in* **P** *to decide whether $\mathrm{res}_{\mathcal{A}} = \mathrm{res}_{\mathcal{B}}$.*

Proof. Applying Proposition 3.5.3 twice, it is possible to verify in polynomial time that $\mathrm{res}_{\mathcal{A}}(T) \subseteq \mathrm{res}_{\mathcal{B}}(T)$ and $\mathrm{res}_{\mathcal{B}}(T) \subseteq \mathrm{res}_{\mathcal{A}}(T)$ for all states $T \subseteq S$. □

With a proof similar to the one of Proposition 3.5.3, we obtain the following result for reactantless RSs.

Proposition 3.5.5. *Given $\mathcal{A} = (S, A), \mathcal{B} = (S, B) \in \mathcal{RS}(0, \infty)$, if*

$$\mathrm{res}_{\mathcal{A}}(S \setminus I_a) \subseteq \mathrm{res}_{\mathcal{B}}(S \setminus I_a) \quad \forall a \in A, \tag{74}$$

then $\mathrm{res}_{\mathcal{A}}(T) \subseteq \mathrm{res}_{\mathcal{B}}(T)$ for all states $T \subseteq S$.

Proof. Let $T \subseteq S$ such that $T \neq S \setminus I_a$ for all $a \in A$. By definition we have

$$\mathrm{res}_{\mathcal{A}}(T) = \bigcup_{a \in A : I_a \cap T = \varnothing} \mathrm{res}_a(T) = \bigcup_{a \in A : I_a \cap T = \varnothing} \mathrm{res}_a(S \setminus I_a) = \bigcup_{a \in A : I_a \cap T = \varnothing} \mathrm{res}_{\mathcal{A}}(S \setminus I_a).$$

If $I_a \cap T = \varnothing$ then $T \subseteq S \setminus I_a$ and, since $\mathrm{res}_{\mathcal{B}}$ is antitone, we have $\mathrm{res}_{\mathcal{B}}(T) \supseteq \mathrm{res}_{\mathcal{B}}(S \setminus I_a)$. Using (74), we obtain

$$\mathrm{res}_{\mathcal{A}}(T) \subseteq \bigcup_{a \in A : I_a \cap T = \varnothing} \mathrm{res}_{\mathcal{B}}(S \setminus I_a) \subseteq \mathrm{res}_{\mathcal{B}}(T)\,. \qquad \square$$

Corollary 3.5.6. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(0, \infty)$ with a common background set S, it is in* **P** *to decide whether* $\mathrm{res}_{\mathcal{A}} = \mathrm{res}_{\mathcal{B}}$.

3.6 BIJECTIVE RESULT FUNCTION

In this section, we study the problem of deciding if the result function of a RS is bijective. This problem is **coNP**-complete for $\mathcal{RS}(\infty, \infty)$ [124, Theorem 7]. In this section, we prove that for inhibitorless and reactantless RSs, the problem is in **P**.

3.6.1 *Bijective Inhibitorless RSs*

Proposition 3.6.1. *Given S a finite set and $f : 2^S \to 2^S$ monotone and bijective, then $|f(T)| = |T|$ for all $T \subseteq S$.*

Proof. Fix $T \subseteq S$ and set $k := |T|$, $n := |S|$. Given $T_1, T_2 \subseteq S$ such that $T_1 \subsetneq T \subsetneq T_2$ then, since f is monotone and injective, it holds

$$f(T_1) \subsetneq f(T) \subsetneq f(T_2). \tag{75}$$

We can deduce two facts from (75) and the injectivity of f:

(i) $f(T)$ strictly contains $2^k - 1$ distinct subsets of S.

(ii) $f(T)$ is strictly contained in $2^{n-k} - 1$ distinct subsets of S.

Now suppose towards a contradiction that $m := |f(T)| < k$; then $f(T)$ can strictly contain at most $2^m - 1 < 2^k - 1$ different subsets of S, contradicting (i). On the other hand, if $m > k$ then $f(T)$ is strictly contained in $2^{n-m} - 1 < 2^{n-k} - 1$ different subsets of S, and this contradicts (ii). We thus conclude that $m = k$, i.e. $|f(T)| = |T|$. $\square$

The first consequence of Proposition 3.6.1 is that bijective monotonic functions are completely determined by their values on the singletons.

Corollary 3.6.2. *Given S a finite set and $f : 2^S \to 2^S$ monotone and bijective, then for all $T \subseteq S$*

$$f(T) = \bigcup_{x \in T} f(\{x\}). \tag{76}$$

Proof. Since f is injective (thus, in particular, it is injective on singletons), we have $|\cup_{x\in T} f(\{x\})| = |T|$; and by Proposition 3.6.1 we have $|\cup_{x\in T} f(\{x\})| = |f(T)|$. By monotonicity of f, $\cup_{x\in T} f(\{x\}) \subseteq f(T)$; therefore, since the two sets have the same cardinality, they are equal. □

Proposition 3.6.3. *Given $f : 2^S \to 2^S$ injective on singletons such that $f(\varnothing) = \varnothing$, $|f(\{x\})| = 1$ for all $x \in S$ and Equation (76) holds for all $T \subseteq S$, then f is monotone and injective.*

Proof. Monotonicity follows directly from Equation (76). To prove injectivity, consider $T_1 \neq T_2$, then there exists $x \in T_1$, $x \notin T_2$. Since f is injective on singletons, $f(\{x\}) \neq f(\{y\})$ for all $y \in T_2$. Then $f(\{x\}) \in f(T_1)$ and $f(\{x\}) \notin f(T_2)$, so in particular $f(T_1) \neq f(T_2)$. □

Remark 3.6.1. Given $\mathcal{A} \in \mathcal{RS}(\infty, 0)$ such that $\mathrm{res}_{\mathcal{A}}$ is injective then $\mathrm{res}_{\mathcal{A}}$ is additive, by Proposition 3.6.3 and Corollary 3.6.2. This means that $\mathcal{A}$ can be 1-simulated by a RS in $\mathcal{RS}(1, 0)$ obtained deleting the reactions of $\mathcal{A}$ with more than two entities in the reactants.

The following sufficient and necessary conditions for a result function of an inhibitorless RS to be bijective follow from Propositions 3.6.1 and 3.6.3 and Corollary 3.6.2.

Corollary 3.6.4. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$, $\mathrm{res}_{\mathcal{A}}$ is injective if and only if the following three conditions are satisfied:*

1. $\mathrm{res}_{\mathcal{A}}(\varnothing) = \varnothing$, $|\mathrm{res}_{\mathcal{A}}(\{x\})| = 1$ *for all* $x \in S$;
2. $\mathrm{res}_{\mathcal{A}}$ *is injective on singletons;*
3. *for all* $(R, \varnothing, P) \in A$, *it holds* $\mathrm{res}_{\mathcal{A}}(R) = \bigcup_{x\in R} \mathrm{res}_{\mathcal{A}}(\{x\})$.

Proof. ($\Rightarrow$) Follows directly from Proposition 3.6.1 and Corollary 3.6.2.
($\Leftarrow$) Arguing as in the proof of Proposition 3.5.3, we obtain that for all $T \subseteq S$,

$$\mathrm{res}_{\mathcal{A}}(T) = \bigcup_{a\in A : R_a \subseteq T} \mathrm{res}_{\mathcal{A}}(R_a) = \bigcup_{a\in A : R_a \subseteq T} \bigcup_{x\in R_a} \mathrm{res}_{\mathcal{A}}(\{x\}).$$

By Condition 1 and 2, every element $x \in T$ belongs to some reaction of the form $(\{x\}, \varnothing, P_x) \in A$ with $|P_x| = 1$, so we obtain $\mathrm{res}_{\mathcal{A}}(T) = \bigcup_{x\in T} \mathrm{res}_{\mathcal{A}}(\{x\})$. The conclusion follows from Proposition 3.6.3. □

Given an inhibitorless RS, we can check the three conditions of Corollary 3.6.4 in polynomial time, obtaining the following.

Corollary 3.6.5. *Given $\mathcal{A} \in \mathcal{RS}(\infty, 0)$, deciding whether $\mathrm{res}_{\mathcal{A}}$ is bijective is in* **P**.

3.6.2 *Bijective Reactantless RSs*

We can develop results dual to Subsection 3.6.1 in the context of reactantless RSs.

Proposition 3.6.6. *Given S a finite set and $f : 2^S \to 2^S$ antitone and bijective, then $|f(T)| = |S \setminus T|$ for all $T \subseteq S$.*

Proof. We first prove that the result holds for $T = \varnothing$ and $T = S$. Suppose by contradiction that $|f(\varnothing)| < |S|$, then $f(\varnothing) \subsetneq S$ and, by antitonicity, for all $T \subseteq S$, $f(T) \subseteq f(\varnothing) \subsetneq S$; thus S does not have a preimage, a contradiction assuming f bijective. We deduce that $f(\varnothing) = S$. Suppose by contradiction, that $|f(S)| > 0$, then $f(S) \supsetneq \varnothing$ and, by antitonicity, for all $T \subseteq S$, $f(T) \supseteq f(S) \supsetneq \varnothing$; thus $\varnothing$ does not have a preimage, a contradiction assuming f bijective.

Fix $\varnothing \subsetneq T \subsetneq S$ and set $k \coloneqq |T|$, $n \coloneqq |S|$. Given $T_1, T_2 \subseteq S$ such that $T_1 \subsetneq T \subsetneq T_2$ then, since f is antitone and injective, it holds

$$f(T_1) \supsetneq f(T) \supsetneq f(T_2). \tag{77}$$

We can deduce two facts from (77) and the injectivity of f:

(i) $f(T)$ strictly contains $2^{n-k} - 1$ distinct subsets of S.

(ii) $f(T)$ is strictly contained in $2^k - 1$ distinct subsets of S.

Now suppose towards a contradiction that $m \coloneqq |f(T)| < n - k$; then $f(T)$ can strictly contain at most $2^m - 1 < 2^{n-k} - 1$ different subsets of S, contradicting (i). On the other hand, if $m > n - k$ then $f(T)$ is strictly contained in $2^{n-m} - 1 < 2^k - 1$ different subsets of S, and this contradicts (ii). We thus conclude that $m = n - k$, i.e. $|f(T)| = |S \setminus T|$. □

The first consequence of Proposition 3.6.6 is that bijective antitone functions are completely determined by their values on sets of the type $S \setminus \{x\}$.

Corollary 3.6.7. *Given S a finite set and $f : 2^S \to 2^S$ antitone and bijective, then for all $T \subseteq S$*

$$f(T) = \bigcup_{x \in S \setminus T} f(S \setminus \{x\}). \tag{78}$$

Proof. Since $|S \setminus \{x\}| = |S| - 1$, by Proposition 3.6.6, we have $|f(S \setminus \{x\})| = 1$. Since f is injective, for any $x, y \in S$, $f(S \setminus \{x\}) \neq f(S \setminus \{y\})$, thus we have $|\cup_{x \in S \setminus T} f(S \setminus \{x\})| = |S \setminus T|$; and by Proposition 3.6.6 we have $|\cup_{x \in S \setminus T} f(S \setminus \{x\})| = |f(T)|$. By antitonicity of f, for all $x \notin T$, $T \subseteq S \setminus \{x\}$ implies $f(T) \supseteq f(S \setminus \{x\})$, thus $\cup_{x \in S \setminus T} f(S \setminus \{x\}) \subseteq f(T)$; therefore, since the two sets have the same cardinality, they are equal. □

Proposition 3.6.8. *Given $f : 2^S \to 2^S$ such that*

1. *$f(\varnothing) = S, f(S) = \varnothing$ and $|f(S \setminus \{x\})| = 1$ for all $x \in S$;*
2. *$f(S \setminus \{x\}) \neq f(S \setminus \{y\})$ for all $x, y \in S$;*
3. *Equation (78) holds for all $T \subseteq S$.*

Then f is antitone and injective.

Proof. Antitonicity follows directly from Equation (78). To prove injectivity, consider $T_1 \neq T_2$, then there exists $x \in T_1$, $x \notin T_2$. By Condition 2, $f(S \setminus \{x\}) \neq f(S \setminus \{y\})$ for all $y \in S \setminus T_1$. Then $f(S \setminus \{x\}) \notin f(T_1)$ and $f(S \setminus \{x\}) \in f(T_2)$, so in particular $f(T_1) \neq f(T_2)$. □

The following sufficient and necessary conditions for a result function of a reactantless RS to be bijective follow from Propositions 3.6.6 and 3.6.8 and Corollary 3.6.7.

Corollary 3.6.9. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$, $\mathrm{res}_{\mathcal{A}}$ is injective if and only if the following three conditions are satisfied:*

1. $\mathrm{res}_{\mathcal{A}}(\varnothing) = S$, $\mathrm{res}_{\mathcal{A}}(S) = \varnothing$, $|\mathrm{res}_{\mathcal{A}}(S \setminus \{x\})| = 1$ *for all* $x \in S$;
2. $\mathrm{res}_{\mathcal{A}}(S \setminus \{x\}) \neq \mathrm{res}_{\mathcal{A}}(S \setminus \{y\})$ *for all* $x, y \in S$;
3. *for all* $(\varnothing, I, P) \in A$, *it holds* $\mathrm{res}_{\mathcal{A}}(S \setminus I) = \bigcup_{x \in I} \mathrm{res}_{\mathcal{A}}(S \setminus \{x\})$.

Proof. ($\Rightarrow$) Follows directly from Proposition 3.6.6 and Corollary 3.6.7.

($\Leftarrow$) We need only to prove that for all $T \subseteq S$, Equation (78) holds for $f = \mathrm{res}_{\mathcal{A}}$. Arguing as in the proof of Proposition 3.5.5, we obtain that for all $T \subseteq S$,

$$\mathrm{res}_{\mathcal{A}}(T) = \bigcup_{a \in A : I_a \cap T = \varnothing} \mathrm{res}_{\mathcal{A}}(S \setminus I_a) = \bigcup_{a \in A : I_a \cap T = \varnothing} \bigcup_{x \in I_a} \mathrm{res}_{\mathcal{A}}(S \setminus \{x\}).$$

By Conditions 1 and 2, for each $x \in S \setminus T$ the state $S \setminus \{x\}$ must enable a reaction of the form $(\varnothing, \{x\}, P_x) \in A$ with $|P_x| = 1$, so we can rewrite $\mathrm{res}_{\mathcal{A}}(T)$ as $\mathrm{res}_{\mathcal{A}}(T) = \bigcup_{x \in S \setminus T} \mathrm{res}_{\mathcal{A}}(S \setminus \{x\})$. The conclusion follows from Proposition 3.6.8. □

Given a reactantless RS, we can check the three conditions of Corollary 3.6.9 in polynomial time, obtaining the following.

Corollary 3.6.10. *Given $\mathcal{A} \in \mathcal{RS}(0, \infty)$, deciding whether $\mathrm{res}_{\mathcal{A}}$ is bijective is in* **P**.

3.7 CONCLUSIONS

We have determined the computational complexity of an extensive set of decision problems regarding the dynamical behaviour of reactantless and inhibitorless RS. This analysis contributes to providing a more comprehensive understanding of how problem complexity varies across different models. Our findings reveal that the simplification of models does not uniformly reduce complexity: most of the analyzed problems retain the same complexity as in the unconstrained model in both reactantless and inhibitorless RSs, some become simpler in both the constrained settings, and some others are equally difficult in unconstrained and reactantless systems but become polynomially decidable in inhibitorless systems.

We leave as an open problem to determine the computational complexity of deciding on the existence of a fixed point attractor in inhibitorless RSs.

4

FIXED POINTS AND ATTRACTORS OF ADDITIVE REACTION SYSTEMS

In this chapter, we study the complexity of the problems presented in Chapter 3 in the context of *additive* RSs. The key tool our results rely on is a variation of the so-called *influence graph*, a graph representation of the reactions of a RS that was first introduced in [59]. In particular, we determine the computational complexity of the considered problems by designing highly non-trivial polynomial-time reductions to various problems on the influence graph and by then providing polynomial-time solutions to these problems. The main contributions of this chapter are summarized in Table 6.

Problem	$\mathcal{RS}(\infty,\infty)$	$\mathcal{RS}(0,\infty)$	$\mathcal{RS}(\infty,0)$	$\mathcal{RS}(1,0)$
A given state is a fixed point attractor	**NP**-c [123]	**NP**-c [20]	**NP**-c [20]	**P** (Cor. 4.3.3)
$\exists$ fixed point	**NP**-c [123]	**NP**-c [20]	**P** [139]	
$\exists$ common fixed point	**NP**-c [123]	**NP**-c [20]	**NP**-c [20]	**P** (Cor. 4.2.6)
sharing all fixed points	**coNP**-c [123]	**coNP**-c [20]	**coNP**-c [20]	**P** (Cor. 4.2.4)
$\exists$ fixed point attractor	**NP**-c [123]	**NP**-c [20]	Unknown	**P** (Cor. 4.3.15)
$\exists$ common fixed point attractor	**NP**-c [123]	**NP**-c [20]	**NP**-c [20]	**P** (Cor. 4.3.16)
sharing all fixed points attractor	$\Pi_2^{\mathbf{P}}$-c [123]	$\Pi_2^{\mathbf{P}}$-c [20]	$\Pi_2^{\mathbf{P}}$-c [20]	**P** (Cor. 4.3.17)
$\exists$ fixed point not attractor	$\Sigma_2^{\mathbf{P}}$-c [20]	$\Sigma_2^{\mathbf{P}}$-c [20]	$\Sigma_2^{\mathbf{P}}$-c [20]	**P** (Cor. 4.3.8)
$\exists$ common fixed point not attractor	$\Sigma_2^{\mathbf{P}}$-c [20]	$\Sigma_2^{\mathbf{P}}$-c [20]	$\Sigma_2^{\mathbf{P}}$-c [20]	**P** (Prop. 4.3.13)
sharing all fixed points not attractor	**coNP**-c [20]	**coNP**-c [20]	**coNP**-c [20]	**P** (Cor. 4.3.12)

Table 6. Computational complexity of the problems studied in this chapter for different classes of RSs. $\mathcal{RS}(\infty,\infty)$, $\mathcal{RS}(0,\infty)$, $\mathcal{RS}(\infty,0)$, and $\mathcal{RS}(1,0)$ denote unconstrained, reactantless, inhibitorless and additive RSs, respectively (see Definition 2.1.1). Highlighted cells contain the results proved in this chapter.

Chapter organization. In Section 4.1, we provide basic notions on the influence graph and introduce the notation used throughout. In Section 4.2, we study problems related to the existence of fixed points. In Section 4.3, we study problems related to the existence of fixed points that are either attractors or not attractors. Finally, in Section 4.4 we discuss the obtained results.

This chapter is based on the following publication: R. Ascone, G. Bernardini, L. Manzoni. Fixed points and attractors of Additive Reaction Systems. In: *Natural Computing*, 2024. [19].

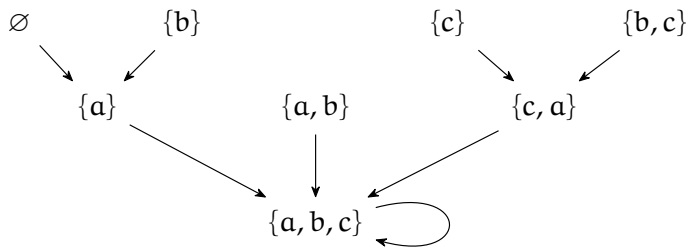


Figure 11. Graph representation of the dynamical system of $\mathcal{A} = (S, A)$ with background set $S = \{a, b, c\}$ and set of reactions $A = \{(\varnothing, \varnothing, \{a\}), (\{a\}, \varnothing, \{b, c\}), (\{c\}, \varnothing, \{c\})\}$.

4.1 INFLUENCE GRAPH FOR ADDITIVE RSS

Recall that the additive class is given by RSs whose result function is additive (see Table 3) and whose reactions have no inhibitors and at most one reactant. See Figure 11 for an example of an additive RS. We also remark that if a function $f : 2^S \to 2^S$ is additive, then it is also monotone, i.e. $X \subseteq Y \subseteq S$ implies $f(X) \subseteq f(Y)$. Therefore, since $\varnothing \subseteq T$ holds for every state $T \subseteq S$, it always holds $f(\varnothing) \subseteq f(T)$. In the example of Figure 11, the fact that $\mathrm{res}_{\mathcal{A}}(\varnothing) = \{a\}$ implies that the endpoint of every edge in the graph must be a state containing the entity a and therefore the states $\varnothing, \{b\}, \{c\}, \{b, c\}$ have in-degree zero.

Definition 4.1.1. Let $\mathcal{A} = (S, A) \in \mathcal{RS}(1, 0)$. The *influence graph* of $\mathcal{A}$ is the directed graph $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ such that $V_{\mathcal{A}} := S \cup \{\varnothing_G\}$, where $\varnothing_G \notin S$ is a vertex representing the empty state, and there is a directed edge $(x, y) \in E_{\mathcal{A}}$ if and only if one of the following holds: (i) $x, y \in S$ and $y \in \mathrm{res}_{\mathcal{A}}(\{x\})$; or (ii) $x \in S$ with $\mathrm{res}_{\mathcal{A}}(\{x\}) = \varnothing$ and $y = \varnothing_G$; or (iii) $x = \varnothing_G$ and $y \in \mathrm{res}_{\mathcal{A}}(\varnothing)$.

Remark 4.1.1. Constructing $G_{\mathcal{A}}$ requires a time polynomial in the number of entities and reactions. In contrast, computing the graph representation of the dynamical system associated with $\mathcal{A}$ requires evaluating the result function $\mathrm{res}_{\mathcal{A}}$ over all possible states, i.e. all the subsets of S, thus requiring exponential time.

Note that Definition 4.1.1 is a slight variation of the definition of influence graph given in [59]. The difference is that, in this variant, reactions may have an empty set of reactants, thus $G_{\mathcal{A}}$ has the extra vertex $\varnothing_G$.

Remark that additive RSs are equivalent to the so called Disjunctive Boolean Networks [132]. Since the rationale of *interaction graph* for BANs is similar to that of the influence graph for RSs, the results and techniques of this chapter can be applied in the framework of Boolean Networks.

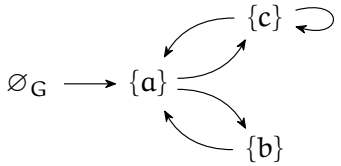


Figure 12. Influence graph $G_{\mathcal{A}}$ of the RS $\mathcal{A}$ from Figure 11.

We denote by $V_u \subseteq V_{\mathcal{A}}$ the set of vertices that are reachable from u in the influence graph $G_{\mathcal{A}}$: clearly, V_u can be computed in linear time with a breadth-first search starting from u. In the following sections, we will sometimes treat V_u as a state of the RS given by the union of the entities represented by its vertices; since $\varnothing_G$ represents the empty set, whenever $\varnothing_G \in V_u$, when interpreting V_u as a subset of S we can simply ignore this vertex.

Remark 4.1.2. Additive RSs, while conceptually simple, do not only exhibit trivial dynamics. For example, the construction in [100, Theorem 4] that proves that sup-reachability is **NP**-complete even for additive RSs includes a way of iterating over all possible assignments of a Boolean formula and verifying its satisfiability, constructing an additive RS whose dynamics can arguably be considered non-trivial.

4.2 FIXED POINTS OF ADDITIVE RSS

In this section, we study the complexity of deciding on the existence of fixed points of different kinds in an additive RS $\mathcal{A}$. To this aim, we make use of the influence graph $G_{\mathcal{A}}$: recall that any subset of vertices of $G_{\mathcal{A}}$ represents a single state of the RS. Let $C \subseteq V_{\mathcal{A}}$ be the set of vertices that are in some cycle of $G_{\mathcal{A}}$: since u is in a cycle if and only if it is reachable from itself, we formally define $C \coloneqq \{u \in V_{\mathcal{A}} \mid u \in V_u\}$. We begin noticing the following.

Lemma 4.2.1. *Let $\mathcal{A} \in \mathcal{RS}(1,0)$ and let $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ be the influence graph of $\mathcal{A}$. For any vertex $u \in V_{\mathcal{A}}$, if $u \in C$ then V_u is a fixed point of $\mathcal{A}$.*

Proof. Let $u = \varnothing_G$. If $\varnothing_G \in C$, then it must be $\mathrm{res}_{\mathcal{A}}(\varnothing) = \varnothing$, as otherwise, if it was $\mathrm{res}_{\mathcal{A}}(\varnothing) = X \neq \varnothing$, X would be produced from any state and $\varnothing_G$ would have no incoming edges by Definition 4.1.1. Thus if $\varnothing_G \in C$ it must be $V_{\varnothing_G} = \{\varnothing_G\}$. Let us now suppose $u \neq \varnothing_G$. We need to show that $\mathrm{res}_{\mathcal{A}}(V_u) = V_u$. Consider $v \in V_u$, thus v is reachable from u: let n be the length of a path from u to v. This implies that there is a path of length at most $n+1$ from u to any vertex $w \in \mathrm{res}_{\mathcal{A}}(\{v\})$, and thus $\mathrm{res}_{\mathcal{A}}(\{v\}) \subseteq V_u$ for all $v \in V_u$. By additivity, we obtain $\mathrm{res}_{\mathcal{A}}(V_u) = \bigcup_{v \in V_u} \mathrm{res}_{\mathcal{A}}(\{v\}) \subseteq V_u$.

To show the other inclusion, let $v \in V_u$, $v \neq u$. Since there exists a path from u to v, there exists a vertex $w \in V_u$ such that $(w, v) \in E_{\mathcal{A}}$, and thus $v \in \mathrm{res}_{\mathcal{A}}(\{w\})$. Therefore $V_u \setminus \{u\} \subseteq \mathrm{res}_{\mathcal{A}}(V_u)$. Since $u \in C$, there also exists a vertex $z \in V_u$ such that $(z, u) \in E_{\mathcal{A}}$, thus $u \in \mathrm{res}_{\mathcal{A}}(V_u)$. Collecting all together, $V_u \subseteq \mathrm{res}_{\mathcal{A}}(V_u)$. □

In other words, Lemma 4.2.1 states that when a vertex u belongs to a cycle, the vertices that are reachable by u give rise to a fixed point in the dynamic of the RS.

In particular, since $\mathrm{res}_{\mathcal{A}}$ is additive, the union of multiple V_u's that are fixed points is a fixed point. In the next proposition, we show that the converse is also true. To do so, for any fixed point $T \neq \varnothing$ of a RS $\mathcal{A}$ we define the set $C_T \coloneqq T \cap C$. In other words, C_T is the subset of elements of T that are part of some cycle in the influence graph, although they are not necessarily all part of the same cycle. For instance, inspect Figure 13 and let $T = \{u_1, u_2, u_3\}$. Then $C_T = \{u_1, u_2, u_3\}$, u_1 is in a distinct cycle from u_2 and u_3 and their union does not induce a cycle in $G_{\mathcal{A}}$.

Proposition 4.2.2. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(1,0)$ and $\varnothing \subsetneq T \subseteq S$ a fixed point, then*

$$T = \bigcup_{u \in C_T} V_u$$

Proof. Given any $u \in T$ we have, by monotonicity and by the definition of fixed point, $\mathrm{res}^n_{\mathcal{A}}(\{u\}) \subseteq T$ for all $n \in \mathbb{N}$. Note that since $\mathrm{res}^n_{\mathcal{A}}(\{u\})$ corresponds to all the vertices of the influence graph $G_{\mathcal{A}}$ that are reachable from u in n steps, we have $V_u \subseteq T$. In particular, we also have

$$\bigcup_{u \in C_T} V_u \subseteq T.$$

Now, suppose for a contradiction that

$$W \coloneqq \bigcup_{u \in C_T} V_u \subsetneq T.$$

Let $x \in T \setminus W$. Since $x \in T$, there exists $y \in T$ such that $x \in \mathrm{res}_{\mathcal{A}}(\{y\})$. If $y \in W$ then $y \in V_u$ for some $u \in C_T$, so y is reachable from u and x too, which implies $x \in V_u \subseteq W$, a contradiction. Then $y \notin W$. Furthermore, if $y = x$ then $x \in C_T$, and so $x \in W$, a contradiction. Putting all together, for all $x \in T \setminus W$ there exists an edge $(y, x) \in E_{\mathcal{A}}$ for some $y \neq x$ and $y \in T \setminus W$. It follows that every vertex in the subgraph $T \setminus W$ has a positive indegree, therefore it must contain a cycle [120, Corollary 3.8c]; in turn, this means that there exists a vertex $z \in T \setminus W$ such that $z \in V_z$, so $z \in W$, a contradiction. The statement follows. □

Given $\mathcal{A} \in \mathcal{RS}(1,0)$, let us define

$$F_{\mathcal{A}} \coloneqq \begin{cases} \{V_u \mid u \in C, u \neq \varnothing_G\} & \text{if } \mathrm{res}_{\mathcal{A}}(\varnothing) \neq \varnothing \\ \{V_u \mid u \in C, u \neq \varnothing_G\} \cup \{\varnothing\} & \text{otherwise} \end{cases}$$

An example of set $F_{\mathcal{A}}$ is in Figure 13. Proposition 4.2.2 and Lemma 4.2.1 imply the following theorem.

Theorem 4.2.3. *Let $\mathcal{A} = (S, A) \in \mathcal{RS}(1,0)$ and $T \subseteq S$, then T is a fixed point of $\mathcal{A}$ if and only if $T = \cup_{V_u \in W} V_u$ for some non-empty subset W of $F_{\mathcal{A}}$. In particular, $\mathcal{A}$ has a fixed point if and only if $F_{\mathcal{A}} \neq \varnothing$.*

Since computing sets V_u requires polynomial time, it immediately follows that computing $F_{\mathcal{A}}$ and thus deciding on the existence of a fixed point is in **P**. Note that the last statement also follows directly from the Knaster-Tarski Theorem [139], since $\mathrm{res}_{\mathcal{A}}$ is additive (and therefore monotonic) for $\mathcal{A} \in \mathcal{RS}(1,0)$.

Remark 4.2.1. If $V_u \in F_{\mathcal{A}}$ then V_u cannot be obtained as a union of other elements of $F_{\mathcal{A}}$. Indeed, suppose for a contradiction that $V_u = \bigcup_{i=1}^n V_{u_i}$ with $V_{u_i} \in F_{\mathcal{A}}$, $V_{u_i} \neq V_u$ for all $i = 1, \ldots, n$. If $u \in V_{u_i} \subseteq V_u$ for some i, then there exists a path from u_i to u, implying that for every $v \in V_u$ there exists a path from u_i to v, i.e. $v \in V_{u_i}$, therefore $V_{u_i} = V_u$, a contradiction. Thus $u \in V_u$ (because $u \in C$) but $u \notin \cup_{i=1}^n V_{u_i}$, hence the thesis.

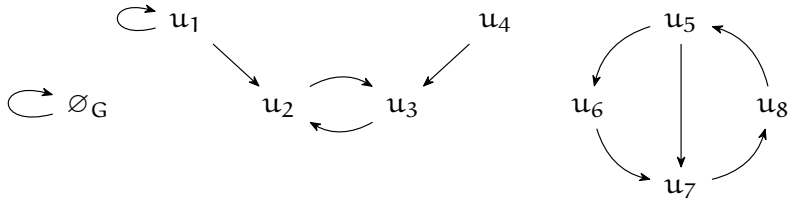


Figure 13. The influence graph of the RS $\mathcal{A}$ with background set $S = \{u_1, u_2, \ldots, u_8\}$ and set of reactions $A = \{(\{u_1\}, \varnothing, \{u_1, u_2\}), (\{u_2\}, \varnothing, \{u_3\}), (\{u_3\}, \varnothing, \{u_2\}), (\{u_4\}, \varnothing, \{u_3\}), (\{u_5\}, \varnothing, \{u_6, u_7\}), (\{u_6\}, \varnothing, \{u_7\}), (\{u_7\}, \varnothing, \{u_8\}), (\{u_8\}, \varnothing, \{u_5\})\}$. In this case $F_{\mathcal{A}} = \{V_{u_1}, V_{u_2}, V_{u_5}, \varnothing_G\}$; note that $V_{u_2} = V_{u_3}$ and $V_{u_5} = V_{u_6} = V_{u_7} = V_{u_8}$; $u_4 \notin V_{u_4}$ thus $V_{u_4} \notin F_{\mathcal{A}}$.

Corollary 4.2.4. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(1,0)$ with a common background set S, it is in* **P** *to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all fixed points.*

Proof. It suffices to construct the influence graphs $G_{\mathcal{A}}$ and $G_{\mathcal{B}}$ in polynomial time and to compute $F_{\mathcal{A}}$ and $F_{\mathcal{B}}$ with a breadth-first visit of the graphs. To decide if $\mathcal{A}$ and $\mathcal{B}$ share all the fixed points it is then enough to check whether $F_{\mathcal{A}} = F_{\mathcal{B}}$, as by Theorem 4.2.3 they generate all fixed points of $\mathcal{A}$ and $\mathcal{B}$, respectively. □

By Theorem 4.2.3, deciding on the existence of a common fixed point for $\mathcal{A}$ and $\mathcal{B}$ reduces to the problem of deciding whether there exists a nonempty subset of elements of $F_{\mathcal{A}}$ and a nonempty subset of elements of $F_{\mathcal{B}}$ such that their respective unions give the same set of entities. Formally, we introduce the following problem, that we coin Maximum Common Subset Union (MCSU):

INPUT: a finite set H and two collections of subsets of H: $\{A_i\}_{i \in I}$ and $\{B_j\}_{j \in J}$

OUTPUT: subsets of indices $I' \subseteq I$ and $J' \subseteq J$ such that $\bigcup_{i \in I'} A_i = \bigcup_{j \in J'} B_j \subseteq H$ is of maximum size

MCSU can be solved in polynomial time using the simple Algorithm 1, in which we denote $A^{I'} := \bigcup_{i \in I'} A_i$. In the beginning, Algorithm 1 computes the union of all the subsets in the respective collections and takes the intersection of the two unions. Clearly, any subset A_i containing an element which is not in this intersection cannot be part of a solution, and likewise for subsets B_j: this condition is checked at lines 5 and 7, where subsets containing elements outside the intersection are removed from the respective collections.

This process is then iterated by taking the union of the survived subsets in each collection and again the intersection of these unions (line 8). The algorithm terminates when there are no longer elements outside the intersection of the unions, i.e. the two unions are the same (line 3). Theorem 4.2.5 proves the correctness of this procedure.

Theorem 4.2.5. *Algorithm 1 is correct.*

Proof. Keeping track of the set of indices at each iteration, we obtain two sequences

$$I = I_1 \supseteq I_2 \supseteq I_3 \supseteq \cdots \supseteq I_n$$
$$J = J_1 \supseteq J_2 \supseteq J_3 \supseteq \cdots \supseteq J_n$$

Algorithm 1 MAX COMMON SUBSET UNION

Input: $\{A_i\}_{i\in I}$ and $\{B_j\}_{j\in J}$ two collections of subsets of a finite set H
Output: $I' \subseteq I$ and $J' \subseteq J$ such that $A^{I'} = B^{J'}$ is of maximum size

1: $\bar{A} \leftarrow A^I \setminus (A^I \cap B^J)$, $\bar{B} \leftarrow B^J \setminus (A^I \cap B^J)$
2: $I' \leftarrow I$, $J' \leftarrow J$
3: **while** $\bar{A} \cup \bar{B} \neq \varnothing$ **do**
4: **for all** $i \in I'$ **do**
5: **if** $A_i \cap \bar{A} \neq \varnothing$ **then** $I' \leftarrow I' \setminus \{i\}$
6: **for all** $j \in J'$ **do**
7: **if** $B_j \cap \bar{B} \neq \varnothing$ **then** $J' \leftarrow J' \setminus \{j\}$
8: $\bar{A} \leftarrow A^{I'} \setminus (A^{I'} \cap B^{J'})$, $\bar{B} \leftarrow B^{J'} \setminus (A^{I'} \cap B^{J'})$
9: **return** I', J'

with n the number of iterations made by the algorithm. We remark that at each iteration $\bar{A} \cup \bar{B} = A^{I'} \triangle B^{J'}$, therefore the algorithm stops at iteration n if and only if $A^{I_n} \triangle B^{J_n} = \varnothing$, i.e. $A^{I_n} = B^{J_n}$. Since at each iteration either $|I_n|$ or $|J_n|$ decreases, the sequences will eventually stop when $I_n = J_n = \varnothing$. To prove correctness, we first verify that if the algorithm stops when $I' = J' = \varnothing$ then there does not exist a nonempty solution to MCSU. Suppose for a contradiction that $I_n = J_n = \varnothing$, and yet there exist $I \supseteq \widetilde{I} \neq \varnothing$ and $J \supseteq \widetilde{J} \neq \varnothing$ such that $A^{\widetilde{I}} = B^{\widetilde{J}}$. Let then k be the last iteration such that $I_k \supseteq \widetilde{I}$ and h the last iteration such that $J_h \supseteq \widetilde{J}$.

Without loss of generality, suppose $k \leqslant h$. We note that going from I_k to I_{k+1} we lose at least an index that was in $\widetilde{I}$, i.e. $I_k \supseteq \widetilde{I} \not\supseteq I_{k+1}$, otherwise k would not be the last iteration such that $I_k \supseteq \widetilde{I}$. So there exists $i \in \widetilde{I} \subseteq I_k$ such that $i \notin I_{k+1}$ and thus, by line 5, there exists $x \in A_i \subseteq A^{\widetilde{I}}$ that is not in the current intersection, thus, in particular, $x \notin B^{J_k}$. Thus $x \notin B^{J_h}$, since $k \leqslant h \Rightarrow J_k \supseteq J_h$. On the other hand, $J_h \supseteq \widetilde{J}$ then $B^{J_h} \supseteq B^{\widetilde{J}} = A^{\widetilde{I}}$ so $x \in A^{\widetilde{I}} \subseteq B^{J_h}$, which is a contradiction.

Similarly, we can prove that the size of the unions of the two output collections is maximum. Suppose there exists $\widetilde{I} \subseteq I$ and $\widetilde{J} \subseteq J$ such that $A^{\widetilde{I}} = B^{\widetilde{J}}$ and $|B^{\widetilde{J}}| = |A^{\widetilde{I}}| > |B^{J_n}| = |A^{I_n}|$. As above, let k the maximum such that $I_k \supseteq \widetilde{I}$ and h the maximum such that $J_h \supseteq \widetilde{J}$. If $k = n$, then $I_n \subseteq \widetilde{I}$, thus $A^{\widetilde{I}} \subseteq A^{I_n}$, a contradiction since $|A^{\widetilde{I}}| > |A^{I_n}|$. In a similar manner, if $h = n$ we get a contradiction. Therefore $k < n$ and $h < n$, and we can conclude as in the first part of this proof. □

A naive implementation of Algorithm 1 applied to S, $F_{\mathcal{A}}$ and $F_{\mathcal{B}}$ requires time $O(|S|^3)$, as $F_{\mathcal{A}}$ and $F_{\mathcal{B}}$ each consist of at most $|S|$ subsets, each containing at most $|S|$ entities, and the algorithm performs at most $|I| + |J| \leqslant 2|S|$ iterations. This implies the following result.

Corollary 4.2.6. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(1,0)$ with a common background set S, it is in* **P** *to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point.*

4.3 FIXED POINTS ATTRACTORS AND NOT ATTRACTORS OF ADDITIVE RSS

In this section, we focus on the problem of deciding on the existence of (common) fixed points that are either attractors or not attractors. We start by giving a character-

ization of the fixed points attractors of an additive function from the power set of a finite set into itself.

Lemma 4.3.1. *Let* $f : 2^H \to 2^H$ *be an additive function,* H *a finite set, and* T *a fixed point for* f*. If* T *is an attractor then there exists* $T' \subseteq H$ *such that either* $T' \subseteq T$ *or* $T' \supseteq T$ *and* $f(T') = T$*.*

Proof. Let $T'' \subseteq H$ such that $f(T'') = T$. If $T'' \subseteq T$, we have the statement, so consider $T'' \nsubseteq T$. Define $T' \coloneqq T \cup T'' \supsetneq T$ and by additivity of f we get $f(T') = f(T'') \cup f(T) = T$. □

Definition 4.3.1. If a fixed point T is reachable from a state $T' \subsetneq T$ (resp. from $T' \supsetneq T$), we say that T is *reachable from below* (resp. *reachable from above*).

The following lemma applies the characterization of Lemma 4.3.1 to additive RS in each of the two possible cases. The proof is analogous to Remark 3.2.1.

Lemma 4.3.2. *Let* T *be a fixed point for* $\mathrm{res}_{\mathcal{A}}$*, where* $\mathcal{A} = (A, S) \in \mathcal{RS}(1, 0)$*. If* T *is reachable from below (resp. from above) then there exists* $x \in T$ *(resp.* $x \notin T$*) such that* T *is reachable from* $T \setminus \{x\}$ *(resp. from* $T \cup \{x\}$*).*

Proof. If there exists $T' \subsetneq T$ such that $\mathrm{res}_{\mathcal{A}}(T') = T$, then for all $x \in T \setminus T'$

$$T = \mathrm{res}_{\mathcal{A}}(T') \subseteq \mathrm{res}_{\mathcal{A}}(T \setminus \{x\}) \subseteq \mathrm{res}_{\mathcal{A}}(T) = T,$$

i.e. T is reachable from $T \setminus \{x\}$. The proof for when T is reachable from above can be obtained entirely analogously. □

Corollary 4.3.3. *Given* $\mathcal{A} \in \mathcal{RS}(1, 0)$*, it is in* **P** *to decide whether a given state* T *is a fixed point attractor.*

Proof. Deciding whether a given state T is a fixed point requires polynomial time (see [20]). By Lemmas 4.3.1 and 4.3.2 it then just suffices to check if the states $T \setminus \{x\}$ for any $x \in T$ or $T \cup \{y\}$ for any $y \notin T$ are attracted by T. □

Lemma 4.3.4. *Given a fixed point* T *of an additive function* $f : 2^H \to 2^H$*, for all* $X \subseteq T$ *we have that* $T \setminus f(X) \subseteq f(T \setminus X)$*.*

Proof. Since $T = f(T) = f(X \cup (T \setminus X)) = f(X) \cup f(T \setminus X)$ we have the thesis. □

The following lemma provides a sufficient condition for the union of two fixed points to be an attractor: an example is given in Figure 14.

Lemma 4.3.5. *Let* $f : 2^H \to 2^H$ *be an additive function,* H *a finite set, and* $T_1 \neq T_2$ *two fixed points for* f*. If there exists* $x \in T_1$*,* $x \notin T_2$ *such that*

$$f(T_2 \cup \{x\}) = T_2$$

then $T_1 \cup T_2$ *is a fixed point attractor reachable from below.*

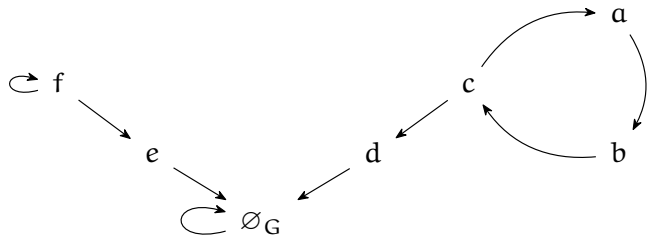


Figure 14. Example for Lemma 4.3.5 with $T_1 = \{e, f\}$, $T_2 = \{a, b, c, d\}$ and $x = e$. In this case, we can also switch the role of T_1, T_2 and choose $x = d$ since $T_1 \cup \{d\}$ reaches T_1. Note that T_1 corresponds to $V_f = \{e, f, \varnothing_G\}$ and T_2 corresponds to $V_a = \{a, b, c, d, \varnothing_G\}$.

Proof. Consider $T_2 \cup (T_1 \setminus \{x\}) \subsetneq T_1 \cup T_2$. Using Lemma 4.3.4 and the additivity of f, we obtain

$$\begin{aligned} T_1 \cup T_2 = f(T_1 \cup T_2) &\supseteq f(T_2 \cup (T_1 \setminus \{x\})) \\ &= T_2 \cup f(T_1 \setminus \{x\}) \\ &\supseteq T_2 \cup (T_1 \setminus \underbrace{f(\{x\})}_{\subseteq T_2}) = T_1 \cup T_2 \,. \end{aligned}$$

We conclude that $T_1 \cup T_2$ is reachable from $(T_2 \cup T_1) \setminus \{x\}$. □

The next proposition gives another sufficient condition for a fixed point to be an attractor.

Proposition 4.3.6. *Let $f : 2^H \to 2^H$ be an additive function, H a finite set, and T' a fixed point. If there exists a fixed point attractor $T \subseteq T'$ then T' is also an attractor.*

Proof. We consider two cases.

Case (i). T is reachable from below, i.e. there exists $x \in T$ such that $f(T \setminus \{x\}) = T$. Then we can write

$$T' \setminus \{x\} = T' \setminus \{x\} \cup T \setminus \{x\}$$

and then, using the additivity of f and Lemma 4.3.4,

$$T' \supseteq f(T' \setminus \{x\}) = f(T' \setminus \{x\}) \cup T \supseteq (T' \setminus \underbrace{f(\{x\})}_{\subseteq T}) \cup T = T',$$

therefore $f(T' \setminus \{x\}) = T'$.

Case (ii). T is reachable from above, i.e. there exists $x \notin T$ such that $f(T \cup \{x\}) = T$. If $x \in T'$ then $T' \cup T = T'$ is an attractor by Lemma 4.3.5. If instead $x \notin T'$, then $f(\{x\}) \subseteq T \subseteq T'$ and we obtain

$$f(T' \cup \{x\}) = T' \cup f(\{x\}) = T',$$

i.e. T' is reachable from $T' \cup \{x\}$. □

Inspect Figure 13 for an example: V_{u_2} is an attractor from above, as well as $V_{u_1} \supset V_{u_2}$, since it contains a fixed point attractor (Proposition 4.3.6); V_{u_5} is an attractor from below; and $\varnothing$ is the only fixed point which is not an attractor. The next lemma provides a simple sufficient condition for all fixed points to be attractors.

Lemma 4.3.7. *Let H be a finite set and $f : 2^H \to 2^H$ an additive function. If $\varnothing$ is not a fixed point, then every fixed point is an attractor.*

Proof. Let T be a fixed point. By monotonicity, $f^n(\varnothing) \subseteq T$ for all $n \in \mathbb{N}$. If $T = f^k(\varnothing)$ for some k ($k \geqslant 1$ by hypothesis), then T is reachable from $f^{k-1}(\varnothing)$. Otherwise, if $f^n(\varnothing) \subsetneq T$ for all $n \in \mathbb{N}$, let $m \in \mathbb{N}$ be the minimum index for which the following sequence stabilizes:

$$\varnothing \subsetneq f(\varnothing) \subsetneq \cdots \subsetneq f^{m-1}(\varnothing) \subsetneq f^m(\varnothing) = f^{m+1}(\varnothing).$$

Let $T' \coloneqq T \setminus f^m(\varnothing) \cup f^{m-1}(\varnothing) \subsetneq T$, then $f(T') \subseteq T$ and by additivity and Lemma 4.3.4 we obtain

$$f(T') = f(T \setminus f^m(\varnothing)) \cup f^m(\varnothing) \supseteq T \setminus f^{m+1}(\varnothing) \cup f^m(\varnothing) = T.$$

Therefore T is reachable from T'. □

Given any $\mathcal{A} = (S, A) \in \mathcal{RS}(1, 0)$, we can draw the following observations on the existence of a fixed point $T \subseteq S$ which is not an attractor. First, a direct implication of Proposition 4.3.6 is that T cannot contain any fixed point attractor and in particular, in the decomposition $T = \bigcup_{u \in C_T} V_u$ given by Proposition 4.2.2 each V_u must be not an attractor; moreover, by Lemma 4.3.7, $\varnothing$ must be a fixed point and, by Proposition 4.3.6, it must not be an attractor as well.

If we now define $F_{\mathcal{A}}^{NA} \coloneqq \{V_u \in F_{\mathcal{A}} \mid V_u \text{ not attractor}\}$, we have obtained a way to decide in polynomial time on the existence of fixed points that are not attractors, summarized by the following result.

Corollary 4.3.8. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(1, 0)$, it is in* **P** *to decide if there exists a fixed point which is not an attractor.*

Proof. We just need to check whether $F_{\mathcal{A}}^{NA} \neq \varnothing$. □

The next result provides a sufficient condition to rule out the existence of fixed points which are not attractors.

Proposition 4.3.9. *Let H be a finite set and $f : 2^H \to 2^H$ an additive, non-injective function. If H is a fixed point for f, then it is also an attractor.*

Proof. Since f is not injective, there exist $X_1, X_2 \subseteq H$ such that $f(X_1) = f(X_2)$ and $X_1 \neq X_2$. We can suppose $X_2 \neq \varnothing$. Let $H' \coloneqq H \setminus (X_1 \cup X_2)$, then $H' \cup X_1 = H \setminus X_2 \subsetneq H$. Furthermore, by Lemma 4.3.4 we have that $f(H' \cup X_1) = f(T \setminus X_2) \supseteq H \setminus f(X_2)$, and $f(X_1) \subseteq f(H' \cup X_1)$ since $X_1 \subseteq H' \cup X_1$. Collecting all together, we obtain

$$f(H' \cup X_1) \supseteq H \setminus f(X_2) \cup f(X_1) = H \setminus f(X_2) \cup f(X_2) = H,$$

thus $f(H' \cup X_1) = H$, i.e. H is reachable from $H' \cup X_1 = H \setminus X_2$. □

The following proof can be viewed as a particular case of Proposition 3.6.1.

Proposition 4.3.10. *Given H a finite set and $f : 2^H \to 2^H$ additive and bijective, then $|f(\{x\})| = 1$ for all $x \in H$.*

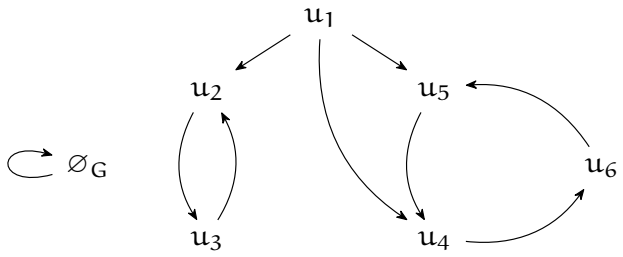


Figure 15. Example of union of elements in $F_{\mathcal{A}}^{NA} = \{V_{u_2}, V_{u_5}, \varnothing\}$ that gives a fixed point attractor $V^{u_1} = V_{u_2} \cup V_{u_5} = \{u_2, u_3, u_4, u_5, u_6\}$ reachable from $\{u_1\} \cup V^{u_1}$.

Proof. Suppose for a contradiction that there exists $x \in H$ such that $k := |f(\{x\})| > 1$. Given any subset $T \supsetneq \{x\}$, since f is injective then $f(\{x\}) \subsetneq f(T) \subseteq H$. Therefore we obtain a bijection between the subsets of H containing $\{x\}$ and the ones containing $f(\{x\})$. This is a contradiction because there are $2^{|H|-1}$ subsets of the first type and only $2^{|H|-k} < 2^{|H|-1}$ of the second type. □

We remark that given an additive function $f : 2^H \to 2^H$ and $T \subseteq H$ a fixed point for f, if the restriction of f to T $f|_T \colon 2^T \to 2^T$ is not injective then T is an attractor by Proposition 4.3.9. Therefore, given any additive RS $\mathcal{A}$, for every $V_u \in F_{\mathcal{A}}^{NA}$, $V_u \neq \varnothing$, the restriction $res_{\mathcal{A}}|_{V_u}$ must be injective, which means that $|res_{\mathcal{A}}(\{x\})| = 1$ for all $x \in V_u$ by Proposition 4.3.10. Since V_u is weakly connected by definition, we obtain the following.

Lemma 4.3.11. *Given any $V_u \in F_{\mathcal{A}}^{NA}$, the vertices of V_u form a cycle in $G_{\mathcal{A}}$. Furthermore, for any $V_{u_1}, V_{u_2} \in F_{\mathcal{A}}^{NA}$, it holds $V_{u_1} \cap V_{u_2} = \varnothing$.*

We further remark that, although the single elements of $F_{\mathcal{A}}^{NA}$ are not attractors by definition, unions of its elements might be. To find out which unions of elements of $F_{\mathcal{A}}^{NA}$ are fixed points attractors it suffices to check, for each entity $x \notin \bigcup_{V_u \in F_{\mathcal{A}}^{NA}} V_u$, whether

$$res_{\mathcal{A}}(\{x\}) \subseteq \bigcup_{V_u \in F_{\mathcal{A}}^{NA}} V_u,$$

and if so, consider the smallest subset $W \subseteq F_{\mathcal{A}}^{NA}$ such that $res_{\mathcal{A}}(\{x\}) \subseteq \bigcup_{V_u \in W} V_u$. Since the elements of $F_{\mathcal{A}}^{NA}$ are disjoint by Lemma 4.3.11, W simply consists of the $V_u \in F_{\mathcal{A}}^{NA}$ that can be reached with an edge outgoing from x and can thus clearly be computed in polynomial time.

Then $V^x := \bigcup_{V_u \in W} V_u$ is a fixed point attractor reachable from $V^x \cup \{x\}$. It follows that deciding whether two RSs share all fixed points that are not attractors requires polynomial time, as stated by the following corollary.

Corollary 4.3.12. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(1,0)$ with a common background set S, it is in* **P** *to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all fixed points which are not attractors.*

Proof. It suffices to check whether $F_{\mathcal{A}}^{NA} = F_{\mathcal{B}}^{NA}$ and whether the fixed points attractors given by unions of elements of $F_{\mathcal{A}}^{NA}$ (resp. of $F_{\mathcal{B}}^{NA}$) are the same. □

We next consider the problem of deciding whether two additive RSs have a common fixed point which is not an attractor.

Proposition 4.3.13. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(1,0)$ with a common background set S, it is in* **P** *to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point which is not an attractor.*

Proof. The first step of a polynomial-time procedure to solve the problem is to verify the following necessary condition: there exist a nonempty set of fixed points that are not attractors of $\mathcal{A}$ and a set of fixed points that are not attractors of $\mathcal{B}$ such that the respective union gives the same set of entities. This condition gives rise to an instance of the MCSU problem and can thus be verified in polynomial time using Algorithm 1 with input S, $F^{NA}_{\mathcal{A}}$ and $F^{NA}_{\mathcal{B}}$. Then, either the output is the empty set (in which case the two RSs do not have any common fixed points which are not attractors) or we obtain two subsets $\{V^{\mathcal{A}}_i\}_{i\in I} \subseteq F^{NA}_{\mathcal{A}}$ and $\{V^{\mathcal{B}}_j\}_{j\in J} \subseteq F^{NA}_{\mathcal{B}}$ such that

$$V := \bigcup_{i\in I} V^{\mathcal{A}}_i = \bigcup_{j\in J} V^{\mathcal{B}}_j$$

and V is the maximal common state given by the union of fixed points that are not attractors.

The next step is to check whether V contains a common fixed point which is not an attractor. Any common fixed point that is potentially not an attractor is given by a subset of the elements of $\{V^{\mathcal{A}}_i\}_{i\in I}$ whose union is the same as the union of a subset of the elements of $\{V^{\mathcal{B}}_j\}_{j\in J}$. Computing these sets naively would require time exponential in the number of elements in $\{V^{\mathcal{A}}_i\}_{i\in I}$ and $\{V^{\mathcal{B}}_j\}_{j\in J}$; however, we can compute them in polynomial time by making use of an auxiliary graph constructed as follows (recall that the elements of $\{V^{\mathcal{A}}_i\}_{i\in I}$ and $\{V^{\mathcal{B}}_j\}_{j\in J}$ are disjoint cycles by Lemma 4.3.11).

Let us denote by $u_{\mathcal{A}}$ and $u_{\mathcal{B}}$ the vertices corresponding to entity u in $G_{\mathcal{A}}$ and in $G_{\mathcal{B}}$, respectively; and let $V^{\mathcal{A}} := \{u_{\mathcal{A}} \mid u_{\mathcal{A}} \in \bigcup_{i\in I} V^{\mathcal{A}}_i\}$ and $V^{\mathcal{B}} := \{u_{\mathcal{B}} \mid u_{\mathcal{B}} \in \bigcup_{j\in J} V^{\mathcal{B}}_j\}$. We construct an auxiliary undirected graph $G_C = (V^{\mathcal{A}} \cup V^{\mathcal{B}}, E)$ whose vertices are connected as follows: for each element $u \in V$ there is an edge connecting the corresponding element in $u_{\mathcal{A}} \in V^{\mathcal{A}}$ and $u_{\mathcal{B}} \in V^{\mathcal{B}}$. Furthermore, connect the elements of $V^{\mathcal{A}}$ (resp. $V^{\mathcal{B}}$) replacing the directed edges of the cycles in $\{V^{\mathcal{A}}_i\}_{i\in I}$ (resp. in $\{V^{\mathcal{B}}_j\}_{j\in J}$) with undirected edges. An example is in Figure 16.

Consider then the connected components of G_C. We observe that any union of cycles $T'_{\mathcal{A}}$ of the $\mathcal{A}$-side of G_C which is in a bijection with a union of cycles of the $\mathcal{B}$-side $T'_{\mathcal{B}}$ is such that $T'_{\mathcal{A}} \cup T'_{\mathcal{B}}$ is a union of connected components of G_C; and conversely, by construction, given any connected component T, there is a bijection between the vertices in the $\mathcal{A}$-side of T and those in its $\mathcal{B}$-side, and both the $\mathcal{A}$-side $T_{\mathcal{A}}$ and the $\mathcal{B}$-side $T_{\mathcal{B}}$ of T are unions of cycles.

It follows that the connected components of G_C correspond to minimal subsets of elements of $\{V^{\mathcal{A}}_i\}_{i\in I}$ and $\{V^{\mathcal{B}}_j\}_{j\in J}$ whose unions are in a bijection; and thus to determine whether there is a common fixed point which is not an attractor it suffices first to compute the connected components of G_C, which can be done in linear time; and then to check whether there exists a connected component $T_{\mathcal{A}} \cup T_{\mathcal{B}}$ of G_C such that $T_{\mathcal{A}}$ (resp. $T_{\mathcal{B}}$) does not contain any fixed point attractor of the form V^x. □

Proposition 4.3.14. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(1,0)$ and T a fixed point attractor then either (i) there exists $V_u \subseteq T$ attractor or (ii) there exists $x \notin \bigcup_{V_u \in F^{NA}_{\mathcal{A}}} V_u$ such that $V^x \subseteq T$.*

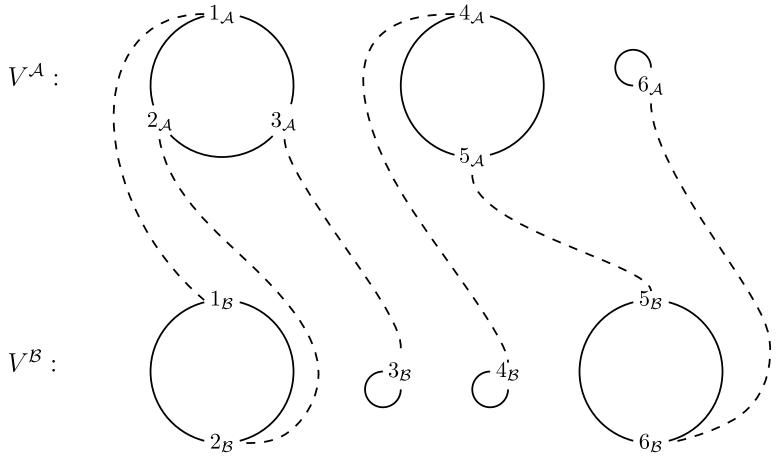


Figure 16. The graph G_c in the construction of the proof of Proposition 4.3.13. The solid edges are those corresponding to the edges in the cycles of $\{V_i^{\mathcal{A}}\}_{i\in I}$ and $\{V_j^{\mathcal{B}}\}_{j\in J}$; the dashed edges connect the pairs of corresponding entities.

Proof. Consider the decomposition $T = \bigcup_{u\in C_T} V_u$ of T given by Proposition 4.2.2. By Proposition 4.3.6, if any of the V_u's in this decomposition is an attractor, then we are in case (i) of the statement. Suppose then that for all $u \in C_T$, it holds $V_u \in F_{\mathcal{A}}^{NA}$. Then T cannot be reached from below since for every $x \in T$, there exists exactly one $V_{\tilde{u}}$ such that $x \in V_{\tilde{u}}$, as the V_u's are disjoint by Lemma 4.3.11, and thus, since $V_{\tilde{u}}$ is not an attractor, implying that $\mathrm{res}_{\mathcal{A}}(V_{\tilde{u}} \setminus \{x\}) \subsetneq V_{\tilde{u}}$, it holds

$$\mathrm{res}_{\mathcal{A}}(T \setminus \{x\}) = \bigcup_{u\in C_T} \mathrm{res}_{\mathcal{A}}(V_u \setminus \{x\}) \subsetneq \bigcup_{u\in C_T} V_u = T.$$

By Lemma 4.3.1 it follows that T must be reachable from above. Let thus $x \notin T$ be such that $\mathrm{res}_{\mathcal{A}}(\{x\}) \subseteq T$. We need to show that we are in case (ii), i.e. $x \notin \bigcup_{V_u\in F_{\mathcal{A}}^{NA}} V_u$ and $V^x \subseteq T$. To prove that $x \notin \bigcup_{V_u\in F_{\mathcal{A}}^{NA}} V_u$, suppose for a contradiction that there exists a $V_{\tilde{u}} \in F_{\mathcal{A}}^{NA}$ such that $x \in V_{\tilde{u}}$, and thus $\mathrm{res}_{\mathcal{A}}(\{x\}) \subseteq V_{\tilde{u}}$. There are two possible cases. If $V_{\tilde{u}} \subseteq T$ then $x \in T$, which is a contradiction. Otherwise, if $V_{\tilde{u}} \nsubseteq T$, it must be $V_{\tilde{u}} \cap T = \varnothing$, because by Lemma 4.3.11 the elements of $F_{\mathcal{A}}^{NA}$ are disjoint; it thus follows that $\mathrm{res}_{\mathcal{A}}(\{\tilde{x}\}) \cap T = \varnothing$, which is again a contradiction. Thus $x \notin \bigcup_{V_u\in F_{\mathcal{A}}^{NA}} V_u$ and $\mathrm{res}_{\mathcal{A}}(\{\tilde{x}\}) \subseteq \bigcup_{u\in C_T} V_u \subseteq \bigcup_{V\in F_{\mathcal{A}}^{NA}} V$; by the minimality of V^x, we finally obtain $V^{\tilde{x}} \subseteq \bigcup_{u\in C_T} V_u = T$. □

Given any additive RS $\mathcal{A}$, let $F_{\mathcal{A}}^{A} := \{V_u \in F_{\mathcal{A}} \mid V_u \text{ attractor}\} \cup \{V^x \mid x \notin \cup_{V\in F_{\mathcal{A}}^{NA}} V\}$. By Proposition 4.3.14, any fixed point attractor of $\mathcal{A}$ contains at least an element from $F_{\mathcal{A}}^{A}$. Since the latter can be computed in polynomial time, we obtain the following corollaries.

Corollary 4.3.15. *Given $\mathcal{A} \in \mathcal{RS}(1,0)$, it is in* **P** *to decide if there exists a fixed point which is not an attractor.*

Proof. We just need to check whether $F_{\mathcal{A}}^{A} \neq \varnothing$. □

Corollary 4.3.16. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(1,0)$ with a common background set S, it is in* **P** *to decide whether $\mathcal{A}$ and $\mathcal{B}$ have a common fixed point attractor.*

Proof. For every pair $V \in \mathrm{F}^{\mathrm{A}}_{\mathcal{A}}$ and $W \in \mathrm{F}^{\mathrm{A}}_{\mathcal{B}}$ we can check in polynomial time, using Algorithm 1, if there exists a common fixed point attractor containing V and W. □

Corollary 4.3.17. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(1,0)$ with a common background set S, it is in* **P** *to decide whether $\mathcal{A}$ and $\mathcal{B}$ share all fixed points attractors.*

Proof. For all fixed points attractors to be shared, it must hold $\mathrm{F}^{\mathrm{A}}_{\mathcal{A}} = \mathrm{F}^{\mathrm{A}}_{\mathcal{B}}$; and furthermore, since adding any fixed point (be it attractor or not) to a fixed point attractor results in a fixed point attractor, it must also hold $\mathrm{F}^{\mathrm{NA}}_{\mathcal{A}} = \mathrm{F}^{\mathrm{NA}}_{\mathcal{B}}$. Both these conditions can be checked in polynomial time. □

From the proof of Corollary 4.3.17 and Corollario 4.2.4, we finally obtain the following result.

Corollary 4.3.18. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(1,0)$ with a common background set S, if they share all fixed point attractors then they share all fixed points.*

4.4 CONCLUSIONS

We have considered all the problems related to fixed points and attractors that were studied in Chapter 3 for unconstrained, reactantless and inhibitorless RSs, and we have fully determined their computational complexity in the context of additive RSs. In doing so, we have provided several reductions from these problems to problems on a polynomially computable graph representation of RSs that might be useful in the future for investigating other problems related to RSs. Furthermore, since additive RSs are equivalent to the so-called Disjunctive Boolean Networks [132], the results and techniques of this chapter can be applied in the framework of Boolean Networks.

5

CYCLES AND GLOBAL ATTRACTORS OF REACTANTLESS AND INHIBITORLESS REACTION SYSTEMS

In this chapter, we study the computational complexity of deciding on the existence of fixed points and cycles that are also *global attractors* (i.e. they can be reached from every other state) in inhibitorless and reactantless RSs. All these problems were shown to be **PSPACE**-complete in unconstrained RSs [124]. We demonstrate that in the absence of reactants or inhibitors, some particular dynamical behaviours can not be obtained, e.g. a global cycle attractor of length greater than 2. Nevertheless, some problems remain **PSPACE**-complete in both the resource-bounded classes, e.g. deciding whether a given state is part of a cycle. The main contributions of this chapter are summarized in Table 7.

Problem		$\mathcal{RS}(\infty,\infty)$	$\mathcal{RS}(0,\infty)$	$\mathcal{RS}(\infty,0)$
A given state is a global attractor		**PSPACE**-c [124]	**P** (Cor. 5.3.3)	**P** (Cor. 5.1.3)
$\exists$ global fixed point attractor		**PSPACE**-c [124]	**P** (Cor. 5.3.4)	**P** (Cor. 5.1.4)
$\exists$ global cycle attractor of length at least k	$k = 2$	**PSPACE**-c [101]	**PSPACE**-c (Thm. 5.3.6)	$\nexists$ (Lemma 5.1.5)
	$k > 2$	**PSPACE**-c [101]	$\nexists$ (Pro. 5.3.5)	$\nexists$ (Lemma 5.1.5)
A given state is part of a cycle		**PSPACE**-c [124]	**PSPACE**-c (Cor. 5.4.3)	**PSPACE**-c (Thm. 5.2.7)
$\exists$ common cycle		**PSPACE**-c [124]	**PSPACE**-c (Thm. 5.4.4)	**PSPACE**-c (Thm. 5.2.9)
sharing all cycles		**PSPACE**-c [124]	**PSPACE**-c (Thm. 5.4.4)	**PSPACE**-c (Thm. 5.2.8)

Table 7. Computational complexity of the problems studied in this chapter for different classes of RSs. **PSPACE**-c is a shorthand for **PSPACE**-complete. $\mathcal{RS}(\infty,\infty)$, $\mathcal{RS}(0,\infty)$ and $\mathcal{RS}(\infty,0)$ denote unconstrained, reactantless and inhibitorless RSs, respectively (see Definition 2.1.1). Highligthed cells contain the results proved in this chapter.

Chapter organization. In Sections 5.1 and 5.3, we study the existence of a global fixed-point attractor or a global cycle attractor in inhibitorless, resp. reactantless, RSs. In Sections 5.2 and 5.4 we prove **PSPACE**-hardness for problems concerning cycles in the class of inhibitorless, resp. reactantless, RSs. Finally, in Section 5.5 we discuss the obtained results.

This chapter is based on the following publication: R. Ascone, G. Bernardini, L. Manzoni. Cycles and global attractors of Reactantless and Inhibitorless Reaction Systems. In: *Theoretical Computer Science*, 2025, [23].

5.1 GLOBAL ATTRACTORS OF INHIBITORLESS RSS

In this section, we study the complexity of deciding the existence of a global fixed-point attractor or a global cycle attractor in inhibitorless RSs.

5.1.1 *Existence of a global fixed-point attractor*

We begin with a simple observation that immediately follows from the definition of global fixed-point attractors.

Observation 5.1.1. A RS with a global fixed-point attractor cannot have any other fixed points or cycles.

In particular, Observation 5.1.1 implies that if a global fixed-point attractor exists, it is unique. Proposition 5.1.1 characterizes global fixed-point attractors for monotone functions.

Proposition 5.1.1. *Let S be a finite set of n elements, $f : 2^S \to 2^S$ a monotone function and T a fixed point for f consisting of t elements. Then, T is a global fixed-point attractor for f if and only if $f^t(\varnothing) = T = f^{n-t}(S)$.*

Proof. $\boxed{\Rightarrow}$ Consider the sequence $\varnothing \subsetneq f(\varnothing) \subsetneq \cdots \subsetneq f^m(\varnothing) = f^{m+1}(\varnothing)$: we have $|f^m(\varnothing)| \geqslant m$. If it held $f^m(\varnothing) \subsetneq T$, a fixed point would exist different from T, and therefore, T would not be a global attractor by Observation 5.1.1. Thus it must be $f^m(\varnothing) = T$; since $t = |T| = |f^m(\varnothing)| \geqslant m$, from the monotonicity of f we derive the following facts:

$$\varnothing \subseteq f^{t-m}(\varnothing) \Rightarrow f^m(\varnothing) \subseteq f^t(\varnothing);$$
$$\varnothing \subseteq T \Rightarrow f^t(\varnothing) \subseteq f^t(T).$$

Collecting all together, we obtain $T = f^m(\varnothing) \subseteq f^t(\varnothing) \subseteq f^t(T) = T$, implying that $f^t(\varnothing) = T$. Consider now the sequence $S \supsetneq f(S) \supsetneq \cdots \supsetneq f^k(S) = f^{k+1}(S)$: we have $|f^k(S)| \leqslant n - k$. If it held $f^k(S) \supsetneq T$, then there would exist a fixed point different from T, and therefore, T would not be a global attractor by Observation 5.1.1. We obtain that $f^k(S) = T$; since $t = |T| = |f^k(S)| \leqslant n - k$, from the monotonicity of f we derive the following facts:

$$S \supseteq f^{n-k-t}(S) \Rightarrow f^k(S) \supseteq f^{n-t}(S);$$
$$S \supseteq T \Rightarrow f^{n-t}(S) \supseteq f^{n-t}(T).$$

Collecting all together, we obtain $T = f^k(S) \supseteq f^{n-t}(S) \supseteq f^{n-t}(T) = T$, implying in particular that $f^{n-t}(S) = T$.

$\boxed{\Leftarrow}$ We need to prove that $T = f^t(\varnothing) = f^{n-t}(S)$ is a global attractor. Consider any state $\varnothing \subsetneq T' \subsetneq S$: by monotonicity, it holds that $T = f^t(\varnothing) \subseteq f^t(T')$ and $f^{n-t}(T') \subseteq f^{n-t}(S) = T$. We divide two cases.

Case (i): $t \leqslant n - t$. Since $T \subseteq f^t(T')$, then it holds $f^{n-2t}(T) \subseteq f^{n-2t+t}(T') \Rightarrow T \subseteq f^{n-t}(T') \subseteq T$, and therefore T' reaches T in at most $n - t$ steps.

Case (ii): $t > n - t$. Since $T \supseteq f^{n-t}(T')$, then $f^{2t-n}(T) \supseteq f^{2t-n+n-t}(T') \Rightarrow T \supseteq f^t(T') \supseteq T$, therefore T' reaches T in at most t steps. □

Proposition 5.1.1 immediately gives a criterion for deciding the existence of a global fixed-point attractor for monotone functions, formalised in Corollary 5.1.2.

Corollary 5.1.2. *Given S a finite set of n elements, and $f : 2^S \to 2^S$ monotone, there exists a global fixed-point attractor if and only if $f^t(\varnothing) = f^{n-t}(S)$ for some $0 \leqslant t \leqslant n$.*

Proposition 5.1.1 and Corollary 5.1.2 can be directly applied to inhibitorless RSs, whose result functions are always monotone. We obtain the following results.

Corollary 5.1.3. *Given a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ and a state $T \subseteq S$, deciding if T is a global fixed-point attractor of $\mathcal{A}$ is in* **P**.

Proof. Since $\mathrm{res}_{\mathcal{A}}$ is monotone [181], we can apply Proposition 5.1.1. Therefore, T is a global attractor for $\mathcal{A}$ if and only if $\mathrm{res}^t_{\mathcal{A}}(\varnothing) = T = \mathrm{res}^{n-t}_{\mathcal{A}}(S)$ where t and n are the cardinalities of T and S, respectively. For any state $U \subseteq S$, $\mathrm{res}_{\mathcal{A}}(U)$ can be computed in polynomial time: it suffices to check which reactions are enabled in U by intersecting their reactants and inhibitors with U, and then take the union of the products of the enabled functions. To decide if T is a global attractor, we only need to evaluate $\mathrm{res}_{\mathcal{A}}$ at most $|S|$ times; thus, the problem is in **P**. □

Corollary 5.1.4. *Given a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$, deciding on the existence of a global fixed-point attractor for $\mathcal{A}$ is in* **P**.

Proof. Since $\mathrm{res}_{\mathcal{A}}$ is monotone [181], we can apply Corollary 5.1.2: there exists a global attractor for $\mathcal{A}$ if and only if $\mathrm{res}^t_{\mathcal{A}}(\varnothing) = \mathrm{res}^{n-t}_{\mathcal{A}}(S)$ for some $0 \leqslant t \leqslant n$, where n is the cardinality of S. We conclude as in Corollary 5.1.3. □

5.1.2 *Existence of a global cycle attractor*

We start by giving a result that immediately follows from the Knaster-Tarski theorem [139] and excludes the existence of a global cycle attractor of length greater than one in the case of monotone functions. In particular, this implies that no global cycle attractor of length $k \geqslant 2$ can exist in the dynamics of inhibitorless RSs, as their result function is always monotone (see Table 3).

Lemma 5.1.5. *Let $f : 2^S \to 2^S$ be a monotone function. Then no global k-cycle attractor exists for any $k \geqslant 2$. Moreover, if $\mathcal{U}$ is a global attractor invariant set, then at least one of the cycles in $\mathcal{U}$ is a fixed point.*

Proof. By the Knaster-Tarki theorem, monotone functions always have a fixed point, therefore a global k-cycle attractor cannot exist for $k > 1$ by Observation 5.1.1. For the same reason, if $\mathcal{U}$ is a global attractor invariant set, then at least one of the cycles in $\mathcal{U}$ is a fixed point. □

The rest of this section provides results on the existence of global attractors consisting of two fixed points for monotone functions (thus for inhibitorless RSs). These results will be useful in Section 5.3 to prove the complexity of deciding on the existence of global cycle attractors in *reactantless* RSs. In Lemma 5.1.6, we prove that for any monotone function, a global attractor consisting of two fixed points must have a particular form.

Lemma 5.1.6. *Let $f : 2^S \to 2^S$ monotone and $\mathcal{U} = \{T_1, T_2\}$ a global attractor consisting of two fixed points, then $\mathcal{U} = \{f^n(\varnothing), f^m(S)\}$, with $n, m \geqslant 0$ such that $f^n(\varnothing) = f^{n+1}(\varnothing)$ and $f^m(S) = f^{m+1}(S)$.*

Proof. Let $n, m \geqslant 0$ such that $f^n(\varnothing) = f^{n+1}(\varnothing)$ and $f^m(S) = f^{m+1}(S)$. By monotonicity, $f^n(\varnothing) \subseteq T_i \subseteq f^m(S)$ for $i = 1, 2$. Suppose towards a contradiction that the inclusions are both strict: then $f^n(\varnothing)$ and $f^m(\varnothing)$ would be fixed points that cannot be attracted by $\mathcal{U}$, a contradiction. We obtain the statement. □

Lemma 5.1.6 implies that any global attractor consisting of two fixed points in a RS $\mathcal{A} \in \mathcal{RS}(\infty, 0)$ must be of the form $\{\mathrm{res}_{\mathcal{A}}^n(\varnothing), \mathrm{res}_{\mathcal{A}}^m(S)\}$. However, this characterization is not strong enough to give a polynomial time algorithm, and in Proposition 5.1.7 we prove that, in the special case where $\varnothing$ and S are fixed points, deciding if $\{\mathrm{res}_{\mathcal{A}}^n(\varnothing), \mathrm{res}_{\mathcal{A}}^m(S)\}$ is a global attractor for $\mathcal{A}$ is **coNP**-hard. The proof extends an idea from Theorem 3.4.3.

Proposition 5.1.7. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $\varnothing$ and S are fixed points, it is **coNP**-hard to decide if $\mathcal{U} = \{\varnothing, S\}$ is a global attractor.*

Proof. To show **coNP**-hardness, we reduce VALIDITY [214] to this problem. Given a Boolean formula $\varphi = \varphi_1 \vee \cdots \vee \varphi_m$ in DNF over the variables $V = \{x_1, \ldots, x_n\}$, let $\overline{V} := \{\overline{x_j} : x_j \in V\}$ and $\heartsuit_S := \{\heartsuit_i : 1 \leqslant i \leqslant n\}$. We define $\mathrm{pos}(\varphi_r) \subseteq V$ the set of variables that occur non-negated in φ_r and $\overline{\mathrm{neg}}(\varphi_r) \subseteq \overline{V}$ the set of variables that occur negated in φ_r. We then define a RS $\mathcal{A}$ with background set $S := V \cup \overline{V} \cup \heartsuit_S \cup \{\diamondsuit\}$ and reactions

$$(\overline{\mathrm{neg}}(\varphi_j) \cup \mathrm{pos}(\varphi_j) \cup \heartsuit_S, \varnothing, \{\diamondsuit\}) \qquad \text{for } 1 \leqslant j \leqslant m \tag{79}$$
$$(\{x_i\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, x_i\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{80}$$
$$(\{\overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\heartsuit_i, \overline{x_i}\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{81}$$
$$(\{x_i, \overline{x_i}\} \cup \heartsuit_S, \varnothing, \{\diamondsuit\}) \qquad \text{for } 1 \leqslant i \leqslant n \tag{82}$$
$$(\{\diamondsuit\} \cup \heartsuit_S, \varnothing, S). \tag{83}$$

Note that $\varnothing$ and S are fixed points; furthermore, any $T \subseteq S$, it falls in one of the following cases:

1) $\heartsuit_S \nsubseteq T$. In this case, $\mathrm{res}_{\mathcal{A}}(T) = \varnothing$, since no reaction is enabled;

2) $\diamondsuit \in T$ and $\heartsuit_S \subseteq T$. In this case, reaction (83) is enabled and thus $\mathrm{res}_{\mathcal{A}}(T) = S$;

3) T is of the form $Y \cup \heartsuit_S$, with $Y \subseteq V \cup \overline{V}$.

Thus $\varnothing$ is reached from any state that does not fully contain $\heartsuit_S$, and S from any state containing both $\heartsuit_S$ and $\diamondsuit$. Let us now focus on the states falling in case (3). For any $Y \subseteq V \cup \overline{V}$, we define $\heartsuit_Y := \{\heartsuit_i : x_i \in Y \vee \overline{x_i} \in Y\} \subseteq \heartsuit_S$. The following subcases can happen:

3.1) $\exists i$ such that both $x_i, \overline{x_i} \in Y$. In this case, the i-th reaction of Group (82) is enabled by $Y \cup \heartsuit_S$, thus $\diamondsuit \in \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$; if $\heartsuit_S \subseteq \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$ or $\heartsuit_S \nsubseteq \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$, then $\mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$ is either in case (1) or (2) above, implying that $\mathrm{res}_{\mathcal{A}}^2(Y \cup \heartsuit_S) \in \{S, \varnothing\}$.

3.2) $\exists i$ such that both $x_i, \overline{x_i} \notin Y$. Then $\heartsuit_S \nsubseteq \mathrm{res}_{\mathcal{A}}(Y \cup \heartsuit_S)$ since none of the i-th reactions in Groups (80), (81) are enabled, therefore, by case (1), $\mathrm{res}^2(Y \cup \heartsuit_S) = \varnothing$.

3.3) $x_i \in Y \Leftrightarrow \overline{x_i} \notin Y$ for every $1 \leqslant i \leqslant n$. In this case, $Y = X \cup \overline{V \setminus X}$ for some $X \subseteq V$, thus it encodes an assignment for φ where the variables in X are assigned true value and the variables in $V \setminus X$ are assigned value false. Note that being φ in DNF, it is satisfied if and only if at least one φ_i is satisfied; moreover, any clause φ_i, being a conjunction of variables, is satisfied if and only if all of its negated variables are assigned value false and all of its non-negated variables are assigned value true. Therefore, the assignment implied by $X \cup \overline{V \setminus X}$ satisfies φ if and only if $X \cup \overline{V \setminus X} \cup \heartsuit_S$ enables one of the reactions from the Group (79). Hence, if $Y = X \cup \overline{V \setminus X}$ satisfies φ then $\diamondsuit \in \mathrm{res}(Y \cup \heartsuit_S)$, implying $\mathrm{res}^2(Y \cup \heartsuit_S) = S$. If instead Y does not satisfy φ then $\mathrm{res}(Y \cup \heartsuit_S) = Y \cup \heartsuit_S$ by reactions of Groups (80) and (81).

We conclude that $\{\varnothing, S\}$ is a global attractor for $\mathcal{A}$ if and only if all the assignments satisfy φ, i.e. φ is a tautology. Since the mapping $\varphi \mapsto \mathcal{A}$ is computable in polynomial time, the problem is **coNP**-hard. □

Note that the proof of Proposition 5.1.7 also implies that even the simpler problem of deciding if $\varnothing$ and S are the only fixed points for $\mathcal{A} \in \mathcal{RS}(\infty, 0)$ is **coNP**-complete, as shown in Corollary 5.1.8.

Corollary 5.1.8. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $\varnothing$ and S are fixed points, it is* **coNP***-complete to decide if $\varnothing$ and S are the only fixed points.*

Proof. The problem is in **coNP** because there exists a simple non-deterministic algorithm which guesses a state T and then verifies in polynomial time that it is a fixed point different from $\varnothing$ and S. The **coNP**-hardness follows from the same reduction as Proposition 5.1.7. □

Corollary 5.1.8 holds even in the general case, where the fixed points are different from $\varnothing$ and S, as stated in Corollary 5.1.9.

Corollary 5.1.9. *Let $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ and $\varnothing \subsetneq T_1 \subsetneq T_2 \subsetneq S$ be fixed points such that $T_1 = \mathrm{res}^n_{\mathcal{A}}(\varnothing)$ and $T_2 = \mathrm{res}^m_{\mathcal{A}}(S)$ for some integer $n, m \geqslant 0$. It is* **coNP***-complete to decide if T_1 and T_2 are the only fixed points.*

Proof. Let $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $\varnothing$ and S are fixed points. We construct $\mathcal{B} = (S', B) \in \mathcal{RS}(\infty, 0)$ where $S' = S \cup \{\heartsuit, \spadesuit\}$ and $B = A \cup \{(\varnothing, \varnothing, \{\heartsuit\})\}$. Since $\mathrm{res}_{\mathcal{A}}(\varnothing) = \varnothing$ and $\mathrm{res}_{\mathcal{A}}(S) = S$, then $\mathrm{res}_{\mathcal{B}}(\{\heartsuit\}) = \{\heartsuit\}$ and $\mathrm{res}_{\mathcal{B}}(S \cup \{\heartsuit\}) = S \cup \{\heartsuit\}$. Furthermore, $\varnothing$ and S are the only fixed points for $\mathcal{A}$ if and only if T_1 and T_2 are the only fixed points for $\mathcal{B}$ with $T_1 = \{\heartsuit\} = \mathrm{res}_{\mathcal{B}}(\varnothing)$, $T_2 = S' \setminus \{\spadesuit\} = \mathrm{res}_{\mathcal{B}}(S')$. □

In Section 5.2 (Theorem 5.2.4), we will prove that the problem of deciding if $\{\varnothing, S\}$ is a global attractor is indeed **PSPACE**-complete.

5.2 CYCLES OF INHIBITORLESS RSS

In this section, we prove **PSPACE**-hardness for problems concerning cycles in the class of inhibitorless RSs.

Remark 5.2.1. Most of the problems studied in this article were proven to be in **PSPACE** [124] for the general class of all RSs $\mathcal{RS}(\infty,\infty)$, as explained in Section 2.2. In the following, we will thus only focus on the *completeness* of such problems.

Let M be any single-tape deterministic Turing machine using m tape cells during its computation; let Σ be the tape alphabet, Q the set of states, and $\delta : Q \times \Sigma \to Q \times \Sigma \times \{-1, 0, +1\}$ the transition function of M. Following the work in [100], we will define a RS $\mathcal{M} = (S, A) \in \mathcal{RS}(\infty, 0)$ simulating M.
Entities. The set of entities of $\mathcal{M}$ is

$$S \coloneqq S_\Sigma \cup \heartsuit_\Sigma \cup S_Q \cup \{\diamondsuit_Q\} \cup \{\spadesuit_j : 1 \leqslant j \leqslant m\}$$

where $\heartsuit_\Sigma \coloneqq \{\heartsuit_\Sigma^j : 1 \leqslant j \leqslant m\}$, $S_\Sigma \coloneqq \{a_j : a \in \Sigma, 1 \leqslant j \leqslant m\}$ and $S_Q \coloneqq \{q_j : q \in Q, 1 \leqslant j \leqslant m\}$. The entities have the following meanings:

a_j : the symbol $a \in \Sigma$ is written on cell j;

$\heartsuit_\Sigma^j$: an alphabet symbol is written on cell j;

q_j: the current state is $q \in Q$ and the head is on cell j;

$\spadesuit_j$: the head is not located on cell j;

$\diamondsuit_Q$: a state symbol is present in the configuration.

In this way, the generic configuration where M is in state $q \in Q$, the tape head is on cell i, and the tape contains the string $x = x_1 \cdots x_m$, is encoded as the following state, where $(x_j)_j$ denotes that the symbol x_j on cell j:

$$T = \{(x_j)_j : 1 \leqslant j \leqslant m\} \cup \heartsuit_\Sigma \cup \{q_i, \diamondsuit_Q\} \cup \{\spadesuit_j : 1 \leqslant j \leqslant m, j \neq i\}. \tag{84}$$

We call a state of this form *valid*, in contrast to *invalid* states that do not encode a configuration of M. This construction differs from the one proposed in [100] by the symbols $\heartsuit_\Sigma \cup \{\diamondsuit_Q\}$, which will be used to control the dynamics of $\mathcal{M}$ when the RS is in an invalid state.
Example. Consider a Turing machine M working in space $m = 4$ and the configuration where M is in state q, the tape head is located on cell 3, and the tape contains the string $abba$. The state of the RS $\mathcal{M}$ that encodes such a configuration of M is then $T = \{a_1, \heartsuit_\Sigma^1, b_2, \heartsuit_\Sigma^2, b_3, \heartsuit_\Sigma^3, a_4, \heartsuit_\Sigma^4, \spadesuit_1, \spadesuit_2, q_3, \diamondsuit_Q, \spadesuit_4\}$.
Reactions. Let us denote $\heartsuit \coloneqq \heartsuit_\Sigma \cup \{\diamondsuit_Q\}$ and $\spadesuit \coloneqq \{\spadesuit_j : 1 \leqslant j \leqslant m\}$. Each transition $\delta(q, a) = (q', a', d)$ of M, with $q, q' \in Q$, $a, a' \in \Sigma$, and $d \in \{-1, 0, +1\}$, is encoded by the following two sets of reactions:

$$(\{q_i, a_i\} \cup \heartsuit, \varnothing, \{q'_{i+d}, a'_i, \heartsuit_\Sigma^i, \diamondsuit_Q\}) \qquad \text{for } 1 \leqslant i \leqslant m \tag{85}$$

$$(\{q_i, a_i\} \cup \heartsuit, \varnothing, \{\spadesuit_j : 1 \leqslant j \leqslant m, j \neq i + d\}) \qquad \text{for } 1 \leqslant i \leqslant m. \tag{86}$$

If the tape head of M is on cell i, then the i-th reaction from Group (85) produces the entity encoding the new state q' and the new tape head position $i+d$ of M, the symbol written on cell i and the flag symbols $\heartsuit^i_\Sigma, \diamondsuit_Q$; and the i-th reaction from Group (86) produces the symbols $\spadesuit_j$ for all tape positions different from the new head position $i+d$, encoding that the head is not on those cells after the transition. Furthermore, the following reactions preserve the encoding of the tape cells where the head is *not* located, which are precisely those for which $\spadesuit_j$ has been produced before:

$$(\{\spadesuit_i, a_i\} \cup \heartsuit, \varnothing, \{a_i, \heartsuit^i_\Sigma\}) \qquad \text{for } 1 \leqslant i \leqslant m. \tag{87}$$

Note that the reactions in Groups (85), (86), (87) are the same as in [100, Section 3] with the addition of flag symbols. We now add novel reactions, whose role is to send invalid states to S, whenever multiple alphabetic symbols referring to the same tape cell or multiple state symbols are present simultaneously:

$$(\{a_i, b_i\} \cup \heartsuit, \varnothing, S) \qquad \forall a, b \in \Sigma \quad \text{for } 1 \leqslant i \leqslant m \tag{88}$$

$$(\{q_i, r_j\} \cup \heartsuit, \varnothing, S) \qquad \forall q, r \in Q \quad \text{for } 1 \leqslant i, j \leqslant m. \tag{89}$$

Furthermore, S must be reached whenever a state symbol q_i is present at the same time as the symbol $\spadesuit_i$ because this would indicate that the tape head is not on the cell the state it corresponds to:

$$(\{q_i, \spadesuit_i\} \cup \heartsuit, \varnothing, S) \qquad \forall q \in Q \quad \text{for } 1 \leqslant i \leqslant m. \tag{90}$$

Finally, we remark that $\varnothing$ and S are fixed points; moreover, if T is valid then $\mathrm{res}_{\mathcal{M}}(T)$ is also valid and encodes the next configuration of M. Let us now study the dynamics of invalid states and start with the following useful remark.

Remark 5.2.2. The set of reactants of all reactions fully contains the set $\heartsuit$; this ensures that $\mathrm{res}_{\mathcal{M}}(T) = \varnothing$ for any state T such that

$$T \cap \heartsuit_\Sigma \subsetneq \heartsuit_\Sigma \quad \vee \quad \diamondsuit_Q \notin T.$$

Lemma 5.2.1. *If $T \subseteq S$ is an invalid state, then T reaches either $\varnothing$ or S.*

Proof. Let $T \subsetneq S$ be a general state such that $\heartsuit \subseteq T$, otherwise by Remark 5.2.2 we have the thesis. We denote

$$T_\Sigma \coloneqq S_\Sigma \cap T, \quad T_Q \coloneqq S_Q \cap T, \quad T_\spadesuit \coloneqq \spadesuit \cap T,$$

thus we can write $T = \heartsuit \cup T_\Sigma \cup T_Q \cup T_\spadesuit$.

If T is valid, by Equation (84) it must be $T_Q = \{q_k\}$ for some $q \in Q$ and $1 \leqslant k \leqslant m$, thus $|T_Q| = 1$. Whenever $|T_Q| > 1$, then a reaction of type (89) is enabled, therefore it would be $\mathrm{res}_{\mathcal{M}}(T) = S$; and whenever $T_Q = \varnothing$ then no reaction of type (85) is enabled, thus either (i) $\mathrm{res}_{\mathcal{M}}(T) = S$ if some reactions from Group (88) are enabled or (ii) $\diamondsuit_Q \notin \mathrm{res}_{\mathcal{M}}(T)$, implying by Remark 5.2.2 that $\mathrm{res}^2_{\mathcal{M}}(T) = \varnothing$. Thus, in the rest of the proof we only consider invalid states such that $T_Q = \{q_k\}$. In the case where $\spadesuit_k \in T_\spadesuit$, then the k-th reaction from Group (90) is enabled, implying $\mathrm{res}_{\mathcal{M}}(T) = S$; and if $T_\spadesuit \subsetneq \spadesuit \setminus \{\spadesuit_k\}$, then there exists $\spadesuit_l \in \spadesuit \setminus (T_\spadesuit \cup \{\spadesuit_k\})$, thus the l-th reaction of Group

(87) is not enabled and $\heartsuit_\Sigma^1 \notin \mathrm{res}_{\mathcal{M}}(T)$ (if no reactions of Group (88) are enabled, otherwise $\mathrm{res}_{\mathcal{M}}(T) = S$), implying by Remark 5.2.2 that $\mathrm{res}^2_{\mathcal{M}}(T) = \varnothing$. Thus, we restrict the rest of the proof to invalid states such that $T_Q = \{q_k\}$ and $T_\spadesuit = \spadesuit \setminus \{\spadesuit_k\}$. Let us focus on T_Σ. We divide two cases in which T would not be valid:

1. if there exists i such that $a_i, b_i \in T_\Sigma$ for some $a, b \in \Sigma$, the i-th reaction in (88) is enabled, thus $\mathrm{res}_{\mathcal{M}}(T) = S$;

2. if there exists i such that $a_i \notin T_\Sigma$ for all $a \in \Sigma$, then:

   2A. if $i = k$, no reaction of type (85) is enabled, thus $\heartsuit_\Sigma^k \notin \mathrm{res}_{\mathcal{M}}(T)$;

   2B. if $i \neq k$, no reaction of type (87) is enabled, thus $\heartsuit_\Sigma^i \notin \mathrm{res}_{\mathcal{M}}(T)$;

   therefore in both cases $\mathrm{res}^2_{\mathcal{M}}(T) = \varnothing$.

The only remaining case is when for all $1 \leqslant i \leqslant m$ there exists a unique $a \in \Sigma$ such that $a_i \in T_\Sigma$; but since we restricted to the case where $T_Q = \{q_k\}$ and $T_\spadesuit = \spadesuit \setminus \{\spadesuit_k\}$, this implies that T is a valid state. We have proved that if T is not a valid state the orbit starting from T reaches $\varnothing$ or S. □

We are ready to prove Theorem 5.2.2 and Lemma 5.2.3, which are analogous to Lemma 1 and Theorem 1 from [124] for inhibitorless RS.

Theorem 5.2.2. *Let M be a single-tape Turing machine with input a string x over Σ and a unary*[1] *integer $m \geqslant |x|$. Then, there exist an inhibitorless RS $\mathcal{M} = (S, A) \in \mathcal{RS}(\infty, 0)$ and a state $X \subseteq S$ such that M reaches its final state $q^{(f)}$ in t steps on input x using at most m tape cells if and only if $\mathrm{res}^t_{\mathcal{M}}(X) = S$.*

Proof. We modify the construction of the proof of Lemma 5.2.1 so as to be able to detect when M exceeds m tape cells. To this aim, we augment S_Q with $\{q_0, q_{m+1} : q \in Q\}$. In this way, whenever one of these elements is generated by a reaction from Group (85) (thus the head of the machine exceeded the array of m cells it is using for the computation), at the next step in $\mathcal{M}$ no reaction from Group (85) is enabled. This implies that $\diamondsuit_Q$ is not generated and by Remark 5.2.2 we reach $\varnothing$. We also add a further group of reactions to ensure that if the computation on the Turing machine stops in the final state $q^{(f)}$, in the RS we reach S:

$$(q_i^{(f)}, \varnothing, S) \qquad \text{for } 1 \leqslant i \leqslant m. \tag{91}$$

Let T_I be the valid state that encodes the initial configuration of M with input x. Taking $X = T_I$ we have the thesis. □

The following lemma will be crucial to prove the next results. Fixed $k \geqslant 1$ an integer, it shows how to construct an inhibitorless RS with a cycle of length $\Omega(2^k)$ using only a polynomial (in k) number of reactions and entities.

Lemma 5.2.3. *Given any integer $k \geqslant 1$, there exists an inhibitorless reaction system $\mathcal{C}_k = (S_k, A_k) \in \mathcal{RS}(\infty, 0)$ with $|S_k| = \mathcal{O}(k)$ such that any state either reaches a cycle of length $\Omega(2^k)$ or is attracted by $\varnothing$ and S.*

1 That is, an integer represented in base 1 rather than in base 2, thus encoded by m 1-bits instead of $\log_2 m$ 0/1 bits.

Proof. Let $C = (\{w, r\}, \{0, 1, \triangleright\}, \delta)$ be the deterministic Turing machine given by the following transition function:

$$\delta(w, 1) = (w, 0, \rightarrow) \tag{92}$$
$$\delta(w, 0) = (r, 1, \leftarrow) \tag{93}$$
$$\delta(w, \triangleright) = (r, \triangleright, -) \tag{94}$$
$$\delta(r, 1) = (r, 1, \leftarrow) \tag{95}$$
$$\delta(r, 0) = (r, 0, \leftarrow) \tag{96}$$
$$\delta(r, \triangleright) = (w, \triangleright, \rightarrow). \tag{97}$$

We defined the transition function as $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\leftarrow, -, \rightarrow\}$, with the clear identification of the set $\{\leftarrow, -, \rightarrow\}$ with the set $\{-1, 0, +1\}$.

The machine C implements a counter that, for any given input integer k, increments a binary number with k digits one by one, starting from 0 and stopping when it exceeds $2^k - 1$. The binary number is stored in the k cells to the right of the symbol $\triangleright$ in reverse order, that is, the least significant digit is on the right. For instance, the string $\triangleright 10100$ represents the number 5; $\triangleright 01100$ the number 6. In this process, when C reaches the configuration with k ones on the tape (i.e. the counter hits $2^k - 1$), it writes 0 in the k cells after $\triangleright$ and stops when it reads a white space in the $(k+1)$-th cell[2].

Given C and a fixed integer k, we construct an inhibitorless RS $\mathcal{C}_k \in \mathcal{RS}(\infty, 0)$ that simulates C over k tape cells to the right of the symbol $\triangleright$. We define S_k as

$$S_k \coloneqq S_\Sigma \cup \heartsuit \cup S_Q \cup \{\spadesuit_j : 0 \leqslant j \leqslant k\} \cup \{\circlearrowleft\}$$

where $S_\Sigma \coloneqq \{0_j, 1_j : 1 \leqslant j \leqslant k\} \cup \{\triangleright_0\}$, $S_Q \coloneqq \{w_j, r_j : 0 \leqslant j \leqslant k\}$, $\heartsuit_\Sigma \coloneqq \{\heartsuit_\Sigma^j : 0 \leqslant j \leqslant k\}$, and $\heartsuit \coloneqq \heartsuit_\Sigma \cup \{\diamondsuit_Q\}$. Thus $|S_k| = 6k + 7 = \mathcal{O}(k)$. The initial configuration of C is represented by the following state in $\mathcal{C}_k$:

$$T_I = \{\triangleright_0, 0_1, \ldots, 0_k\} \cup \heartsuit_\Sigma \cup \{r_0, \diamondsuit_Q\} \cup \{\spadesuit_i : 1 \leqslant i \leqslant k\}.$$

To define the reactions of $\mathcal{C}_k$, consider the groups of reactions constructed at the beginning of Section 5.2. To encode Rules (93), (95), (96), we add to $\mathcal{C}_k$ the corresponding Groups of reactions (85), (86). Furthermore, since we want to force the symbol $\triangleright$ to be always at the first cell of the tape, to encode Rules (94) and (97) we use the following reactions:

$$(\{w_0, \triangleright_0\} \cup \heartsuit, \varnothing, \{r_0, \triangleright_0, \heartsuit_\Sigma^0, \diamondsuit_Q\})$$
$$(\{w_0, \triangleright_0\} \cup \heartsuit, \varnothing, \{\spadesuit_j : 1 \leqslant j \leqslant m\})$$
$$(\{r_0, \triangleright_0\} \cup \heartsuit, \varnothing, \{w_1, \triangleright_0, \heartsuit_\Sigma^0, \diamondsuit_Q\})$$
$$(\{r_0, \triangleright_0\} \cup \heartsuit, \varnothing, \{\spadesuit_0, \spadesuit_j : 2 \leqslant j \leqslant m\}).$$

We must also control the dynamics of $\mathcal{C}_k$ whenever C overflows: we thus need to construct a reaction specific to when we are at position $i = k$ with input symbol 1

2 To visualize a similar counter, in which the binary number is stored to the left of some flag symbol without being reversed, visit https://ideonexus.github.io/Explorable-Explanations/math/binarycountingmachine/.

and state w, i.e. exactly one step before overflowing. Hence to encode Rule (92) we construct the following reactions:

$$(\{w_i, 1_i\} \cup \heartsuit, \varnothing, \{w_{i+1}, 0_i, \heartsuit^i_\Sigma, \diamondsuit_Q\}) \qquad \text{for } 1 \leqslant i < k \tag{98}$$

$$(\{w_i, 1_i\} \cup \heartsuit, \varnothing, \{\spadesuit_j : 1 \leqslant j \leqslant k, j \neq i+1\}) \qquad \text{for } 1 \leqslant i < k \tag{99}$$

$$(\{w_k, 1_k\} \cup \heartsuit, \varnothing, \{\circlearrowleft, 0_k, \heartsuit^k_\Sigma\} \cup \spadesuit) \tag{100}$$

$$(\{\circlearrowleft\} \cup \heartsuit_\Sigma, \varnothing, T_I). \tag{101}$$

We also add the group of reactions (87) to preserve the tape. Recall that if T is a valid state then $\mathrm{res}_{\mathcal{C}_k}(T)$ encodes the next configuration of C. Because the reaction (100) generates the symbol $\circlearrowleft$ instead of $\diamondsuit_Q$ and the reaction (101) produces the initial configuration T_I, the states $\{\mathrm{res}^m_{\mathcal{C}_k}(T_I) : m \in \mathbb{N}\}$ form a cycle. The length of the cycle is $\Omega(2^k)$ because for each binary number $x_1 \cdots x_k$ from 0 to $2^k - 1$ we reach a configuration of the type

$$\{\rhd_0, (x_1)_1, \ldots, (x_k)_k\} \cup \heartsuit_\Sigma \cup \{r_0, \diamondsuit_Q\} \cup \{\spadesuit_i : 1 \leqslant i \leqslant k\}. \tag{102}$$

Remark that the way we defined the reactions guarantees that a valid state always contains $\rhd_0$.

Claim 4. Any valid state T always reaches the cycle starting from T_I.

Proof. If T encodes a configuration with state w, the computation either overflows or reaches a configuration with state r; whenever the state is r, the computation always goes back to $\rhd$ preserving the tape, thus reaches a configuration of the type (102). ■

Finally, to control invalid states, we add the following reactions (which are never enabled by valid states) that are analogous to the reactions (88), (89), (90):

$$(\{a_i, b_i\} \cup \heartsuit, \varnothing, S_k) \qquad \forall a, b \in \{0, 1, \rhd\} \quad \text{for } 0 \leqslant i \leqslant k \tag{103}$$

$$(\{q_i, q'_j\} \cup \heartsuit, \varnothing, S_k) \qquad \forall q, q' \in \{w, r\} \quad \text{for } 0 \leqslant i, j \leqslant k \tag{104}$$

$$(\{q_i, \spadesuit_i\} \cup \heartsuit, \varnothing, S_k) \qquad \forall q \in \{w, r\} \quad \text{for } 0 \leqslant i \leqslant k \tag{105}$$

$$(\{q_i, \circlearrowleft\} \cup \heartsuit, \varnothing, S_k) \qquad \forall q \in \{w, r\} \quad \text{for } 0 \leqslant i \leqslant k \tag{106}$$

where in Group (106) we treat $\circlearrowleft$ as a state of C.

Claim 5. Any invalid state reaches either $\varnothing$, S_k or T_I.

Proof. Let $T \subseteq S_k$ be any invalid state of $\mathcal{C}_k$. If $T \cap \heartsuit_\Sigma \subsetneq \heartsuit_\Sigma$ then no reaction is enabled, so we get $\mathrm{res}_{\mathcal{C}_k}(T) = \varnothing$. Analogously to the proof of Lemma 5.2.1, we study what happens for $T = \heartsuit_\Sigma \cup T_\Sigma \cup T_Q \cup T_\spadesuit \cup T_{\diamondsuit_Q}$, where $T_{\diamondsuit_Q} = T \cap \{\diamondsuit_Q\}$ and $T_Q = T \cap (Q \cup \{\circlearrowleft\})$. Consider the case where $T_{\diamondsuit_Q} = \varnothing$. In this case, no reaction can be enabled, except for (101): if T enables (101) we have $\mathrm{res}_{\mathcal{C}_k}(T) = T_I$, otherwise $\mathrm{res}_{\mathcal{C}_k}(T) = \varnothing$. Now consider the case where $T_{\diamondsuit_Q} = \{\diamondsuit_Q\}$, therefore $T = \heartsuit \cup T_\Sigma \cup T_Q \cup T_\spadesuit$. If $|T_Q| > 1$ then $\mathrm{res}_{\mathcal{C}_k}(T) = S_k$, thus we divide into three cases.

- If $T_Q = \varnothing$, if a reaction of Group (103) is enabled then $\mathrm{res}_{\mathcal{C}_k}(T) = S_k$, otherwise $(S_Q \cup \spadesuit) \cap \mathrm{res}_{\mathcal{C}_k}(T) = \varnothing$, since no reactions from Groups (85), (86) are enabled. Therefore, $\mathrm{res}_{\mathcal{C}_k}(T)$ could only enable reactions of Group (103); we conclude that either $\mathrm{res}^2_{\mathcal{C}_k}(T) = \varnothing$ or $\mathrm{res}_{\mathcal{C}_k}(T) = S_k$.

- If $T_Q = \{q_k\}$ for some k, then we proceed similarly to the proof of Lemma 5.2.1.
- If $T_Q = \{\circlearrowleft\}$, then $\text{res}_{\mathcal{C}_k}(T) = T_I \cup \text{res}_{\mathcal{C}_k}(\heartsuit \cup T_\Sigma \cup T_\spadesuit)$. If a reaction from Group (103) is enabled by $\heartsuit \cup T_\Sigma \cup T_\spadesuit$ then $\text{res}_{\mathcal{C}_k}(T) = S_k$. So suppose that none of them is enabled, then only reactions from Group (87) are enabled, thus $\text{res}_{\mathcal{C}_k}(\heartsuit \cup T_\Sigma \cup T_\spadesuit) \subseteq \heartsuit_\Sigma \cup S_\Sigma$. If $\text{res}_{\mathcal{C}_k}(\heartsuit \cup T_\Sigma \cup T_\spadesuit) \cap S_\Sigma \not\subseteq T_I$, then at the next step one reaction of Group (103) is enabled. We conclude that in this case we either reach T_I or S_k. ■

The thesis follows. □

We are now ready to prove the hardness of the problems in Table 7. From Lemma 5.1.6, we know that for $\mathcal{A} \in \mathcal{RS}(\infty, 0)$ a global attractor given by two fixed points must be of the form $\mathcal{U} = \{\text{res}^n_{\mathcal{A}}(\varnothing), \text{res}^m_{\mathcal{A}}(S)\}$ for some integer $n, m \geqslant 0$; we prove that determining if one such $\mathcal{U}$ is a global attractor is **PSPACE**-complete. We start by proving in Theorem 5.2.4 that already determining whether the special case $\mathcal{U} = \{\varnothing, S\}$ is a global attractor is **PSPACE**-complete.

Theorem 5.2.4. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $\varnothing$ and S are fixed points, it is* **PSPACE***-complete to decide if $\{\varnothing, S\}$ is a global attractor.*

Proof. We prove the hardness of the problem by a reduction from the following **PSPACE**-complete problem: given a deterministic Turing machine M, a string x and a unary integer m, does M accept x without using more than m tape cells? Let $M = (Q, \Sigma, \delta)$ be a single-tape deterministic Turing machine with input x and a unary integer $m \geqslant |x|$. We construct a RS $\mathcal{M} = (S_\mathcal{M}, A_\mathcal{M}) \in \mathcal{RS}(\infty, 0)$ using Theorem 5.2.2. Let k be an integer such that $m \cdot |Q| \cdot |\Sigma|^m \leqslant 2^k$ and construct a RS $\mathcal{C} = \mathcal{C}_k = (S_\mathcal{C}, A_\mathcal{C}) \in \mathcal{RS}(\infty, 0)$ as in Lemma 5.2.3. Let $S_\mathcal{M}$ be the background set of $\mathcal{M}$ and $S_\mathcal{C}$ be the background set of $\mathcal{C}$, such that $S_\mathcal{M} \cap S_\mathcal{C} = \varnothing$. The entities in $S_\mathcal{M}$ (resp. $S_\mathcal{C}$) and the states over $\mathcal{M}$ (resp. $\mathcal{C}$) will be denoted with $T^\mathcal{M}$ (resp. $T^\mathcal{C}$). Let $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $S = S_\mathcal{C} \cup S_\mathcal{M}$ and $A = A'_\mathcal{C} \cup A'_\mathcal{M}$ where $A'_\mathcal{C}$ is the set of reactions $A_\mathcal{C}$ from Lemma 5.2.3 (replacing S_k by S in reactions (103), (104), (106)) and $A'_\mathcal{M}$ is the set of reactions $A_\mathcal{M}$ from Theorem 5.2.2 modified as follows: the entity $\diamondsuit^\mathcal{C}_Q \in S_\mathcal{C}$ is added to the reactants of all reactions and $S_\mathcal{M}$ is replaced by S in reactions (88), (89), (90), (91). Furthermore, we replace the reaction (101) of $A'_\mathcal{C}$ with the following reaction

$$(\{\circlearrowleft^\mathcal{C}\} \cup \heartsuit^\mathcal{C}_\Sigma, \varnothing, T^\mathcal{C}_I \cup T^\mathcal{M}_I) \tag{107}$$

where the state $T^\mathcal{M}_I$ represents the initial configuration of M. In this way, whenever the counter overflows, implying that $\diamondsuit^\mathcal{C}_Q$ does not appear and $\circlearrowleft^\mathcal{C}$ is generated, none of the reactions of $\mathcal{M}$ is enabled, thus reaction (107) resets $\mathcal{C}$ and $\mathcal{M}$ to the initial state.

We now study the dynamics of $\mathcal{A}$. We will denote by $\text{res}_\mathcal{C}(T)$ (resp. $\text{res}_\mathcal{M}(T)$) the result function of $\mathcal{A}$ restricted to $A'_\mathcal{C}$ (resp. to $A'_\mathcal{M}$). Let $T = T^\mathcal{C} \cup T^\mathcal{M} \subseteq S$ given by the union of two valid states of $\mathcal{C}$ and $\mathcal{M}$. Since the background sets of the two reaction systems are disjunct, we have that $\text{res}_\mathcal{A}(T^\mathcal{C} \cup T^\mathcal{M}) = \text{res}_\mathcal{C}(T^\mathcal{C}) \cup \text{res}_\mathcal{M}(T^\mathcal{M})$. Recall that by Claim 4, there exists $t \in \mathbb{N}$ such that $\text{res}^t_\mathcal{C}(T^\mathcal{C}) = T^\mathcal{C}_I$. If the computation of M starting from the configuration $T^\mathcal{M}$ halts, then we reach the state S; otherwise, if M does not halt or uses more than m tape cells (thus $T^\mathcal{M}$ reaches $\varnothing$), the counter overflows and both M and C are reset to the initial state $T^\mathcal{C}_I \cup T^\mathcal{M}_I$. Finally, since

the counter resets after that all possible configurations over m cells of M have been possibly reached, we have that $T_I^{\mathcal{C}} \cup T_I^{\mathcal{M}}$ is part of a cycle if and only if M does not halt or uses more than m cells; furthermore, $T_I^{\mathcal{C}} \cup T_I^{\mathcal{M}}$ reaches S if and only if M halts with input x using at most m tape cells. If we can prove that, whenever we have an invalid configuration of $\mathcal{C}$ or $\mathcal{M}$, the state reaches $\varnothing$,S or $T_I^{\mathcal{C}} \cup T_I^{\mathcal{M}}$, we obtain the thesis.

Now suppose that $T^{\mathcal{M}}$ is an invalid state for $\mathcal{M}$, then $T^{\mathcal{M}}$ either reaches $\varnothing$ or S. In the former case, the dynamics is completely controlled by $T^{\mathcal{C}}$, thus either we reach $\varnothing$ or S, or we enter in the loop of $T_I^{\mathcal{C}}$, thus everything is reset to $T_I^{\mathcal{C}} \cup T_I^{\mathcal{M}}$. We must study the case where $T^{\mathcal{M}}$ is a valid state but $T^{\mathcal{C}}$ is invalid. In this case, if $T^{\mathcal{C}}$ reaches S then $T^{\mathcal{C}} \cup T^{\mathcal{M}}$ reaches S; if $T^{\mathcal{C}}$ reaches $T_I^{\mathcal{C}}$ then either $T^{\mathcal{M}}$ carries on the computation until the counter overflows or $T^{\mathcal{M}}$ reaches the final state; if $T^{\mathcal{C}}$ reaches $\varnothing$, then $\Diamond_Q^{\mathcal{C}}$ is no longer present, implying that the computation of $T^{\mathcal{M}}$ stops and $T^{\mathcal{M}} \cup T^{\mathcal{C}}$ reaches $\varnothing$.

We conclude that M accepts x using less than m tape cells if and only if $\{\varnothing, S\}$ is a global attractor for $\mathcal{A}$. Since the map $(M, x, 1^m) \mapsto \mathcal{A}$ is computable in polynomial time by the constructions of Lemma 5.2.3 and Theorem 5.2.2, we have that deciding if $\{\varnothing, S\}$ is a global attractor for an inhibitorless reaction system is **PSPACE**-hard. □

The following two corollaries can be derived from Theorem 5.2.4.

Corollary 5.2.5. *Let $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ and $\varnothing \subsetneq T_1 \subsetneq T_2 \subsetneq S$ be fixed points such that $T_1 = \mathrm{res}_{\mathcal{A}}^n(\varnothing)$ and $T_2 = \mathrm{res}_{\mathcal{A}}^m(S)$ for some integer $n, m \geqslant 0$. It is* **PSPACE***-complete to decide if $\{T_1, T_2\}$ is a global attractor.*

Proof. Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $\varnothing$ and S are fixed points, we can construct $\mathcal{B} = (S', B) \in \mathcal{RS}(\infty, 0)$ as done in Corollary 5.1.9. It follows that $\{\varnothing, S\}$ is a global attractor for $\mathcal{A}$ if and only if $\{\mathrm{res}_{\mathcal{B}}(\varnothing), \mathrm{res}_{\mathcal{B}}(S')\}$ is a global attractor for $\mathcal{B}$. □

Corollary 5.2.6. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$, deciding if there exists a global attractor given by two fixed points is* **PSPACE***-complete.*

Proof. Let $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $\varnothing$ and S are fixed points. If $\mathcal{A}$ has a global attractor given by two fixed points, it must be $\{\varnothing, S\}$, as if we have a global attractor, all the fixed points and cycles must belong to it; see Observation 5.1.1. Therefore, $\mathcal{A}$ has a global attractor given by two fixed points if and only if $\{\varnothing, S\}$ is a global attractor. Since the former problem is **PSPACE**-complete by Theorem 5.2.4, we have the thesis. □

We now move on to prove that the problems of deciding whether (i) a given state is part of a cycle, (ii) two RSs have the same set of cycles, and (iii) two RSs have at least one cycle in common are all **PSPACE**-complete when restricted to inhibitorless RSs. These results, respectively stated in Theorems 5.2.7, 5.2.8 and 5.2.9, are all almost straightforward implications of Theorem 5.2.4.

Theorem 5.2.7. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ and a state $T \subseteq S$, it is* **PSPACE***-complete to decide whether T is part of a cycle.*

Proof. We reduce from the same **PSPACE**-complete problem we used in the proof of Theorem 5.2.4: given a deterministic Turing machine M, a string x and a unary

integer m, does M accept x without using more than m tape cells? By constructing in polynomial time a RS $\mathcal{A}$ as in the proof of Theorem 5.2.4 and choosing $T = T_I^{\mathcal{C}} \cup T_I^{\mathcal{M}}$, we find that T belongs to a cycle if and only if M does not halt or uses more than m cells. □

Theorem 5.2.8. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, 0)$ over the same background set S, it is* **PSPACE***-complete to decide whether they share all their cycles.*

Proof. We reduce again from the **PSPACE**-complete problem of deciding, given a deterministic Turing machine M, a string x and a unary integer m, if M accepts x without using more than m tape cells. Following the reduction of Theorem 5.2.4, we construct the map $(M, x, 1^m) \mapsto (\mathcal{A}, \mathcal{B})$, where $\mathcal{B}$ has $(S, \varnothing, S)$ as its only reaction. In this way $\mathcal{A}$ and $\mathcal{B}$ share all cycles if and only if the only fixed points of $\mathcal{A}$ are $\varnothing$ and S, and this happens if and only if M accepts x using at most m tape cells. □

Theorem 5.2.9. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(\infty, 0)$ over the same background set S, it is* **PSPACE***-complete to decide whether they share a common cycle.*

Proof. Once again, we reduce from the **PSPACE**-complete problem of deciding, given a deterministic Turing machine M, a string x and a unary integer m, if M accepts x without using more than m tape cells. Let $M = (Q, \Sigma, \delta)$ be a single-tape deterministic Turing machine with input x and a unary integer $m \geqslant |x|$. We construct a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ as in Theorem 5.2.4. We add to the background set of $\mathcal{A}$ a new symbol $\clubsuit \notin S$. Let $\mathcal{B} = (S \cup \{\clubsuit\}, B) \in \mathcal{RS}(\infty, 0)$, where $B = A \cup \{(S, \varnothing, S \cup \{\clubsuit\}), (\varnothing, \varnothing, \heartsuit_\Sigma^{\mathcal{C}} \cup \heartsuit_\Sigma^{\mathcal{M}})\}$. In this way, $\varnothing$ and S are fixed points of $\mathcal{A}$ but not of $\mathcal{B}$. Recall that $\mathcal{A}$ has a cycle given by valid states different from $\varnothing$ and S if and only if M halts for x within m tape cells. Since $\mathcal{A}$ and $\mathcal{B}$ coincide on valid states, we deduce that $\mathcal{A}$ and $\mathcal{B}$ share a common cycle if and only if M with input x halts within m tape cells. □

5.3 GLOBAL ATTRACTORS OF REACTANTLESS RSS

5.3.1 *Existence of a global fixed-point attractor*

We begin this section by characterizing global fixed-point attractors when the function is antitone. Corollary 5.3.2, analogously to Corollary 5.1.2 for the monotone case, will then provide a criterion to decide the existence of a global fixed-point attractor for antitone functions in polynomial time.

Proposition 5.3.1. *Let S be a finite set, $f : 2^S \to 2^S$ antitone and T a fixed point for f. Then, T is a global fixed-point attractor for f if and only if T is a global fixed-point attractor for f^2.*

Proof. $[\Rightarrow]$ Since T is a fixed point for f, it is also a fixed point for f^2. We need to prove that T is a global attractor for f^2, but since for every state $T' \subseteq S$ there exists $t \in \mathbb{N}$ such that $f^t(T') = T$, then $(f^2)^t(T') = f^{2t}(T') = f^2(T) = T$.

$[\Leftarrow]$ Consider T a global fixed-point attractor for f^2. Then it must hold that $f(T) = T$, as otherwise, $f(T) \neq T$ would imply that $f^2(f(T)) = f(T)$ and thus $f(T)$ would be a fixed point for f^2 different from T, which is a contradiction by

Observation 5.1.1. $f(T) = T$ implies that T is also a global fixed-point attractor for f, because for every $T' \subseteq S$, T' reaches T in t steps through f^2, thus T' reaches T in $2t$ steps through f. □

Corollary 5.3.2. *Given S a finite set and $f : 2^S \to 2^S$ antitone, a global fixed-point attractor for f exists if and only if there exists a global fixed-point attractor for f^2.*

Proposition 5.3.1 and Corollary 5.3.2 can be applied straightforwardly to result functions of reactantless RSs, leading to the following two results.

Corollary 5.3.3. *Given a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$ and a state $T \subseteq S$, deciding if T is a global fixed-point attractor of $\mathcal{A}$ is in* **P**.

Proof. Since $\mathrm{res}_{\mathcal{A}}$ is antitone [181], Proposition 5.3.1 applies. Therefore, T is a global attractor for $\mathcal{A}$ if and only if T is a global fixed-point attractor for $\mathrm{res}^2_{\mathcal{A}}$. Since $\mathrm{res}^2_{\mathcal{A}}$ is monotone, we can proceed as in the proof of Corollary 5.1.3, and decide whether T is a global attractor simply by evaluating $\mathrm{res}_{\mathcal{A}}$ at most $2|S|$ times. □

Corollary 5.3.4. *Given a RS $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$, deciding whether there exists a global fixed-point attractor of $\mathcal{A}$ is in* **P**.

Proof. Since $\mathrm{res}_{\mathcal{A}}$ is antitone [181], Corollary 5.3.2 applies, which implies that there exists a global fixed-point attractor for $\mathrm{res}_{\mathcal{A}}$ if and only if there exists a global fixed-point attractor for $\mathrm{res}^2_{\mathcal{A}}$. We conclude as in Corollary 5.3.3. □

5.3.2 *Existence of a global cycle attractor*

We begin this section by showing, in Proposition 5.3.5, that a global k-cycle attractor cannot exist for any antitone function for any $k > 2$: see also Example 5.3.1.

Proposition 5.3.5. *Let $\mathcal{U}$ be a global cycle attractor for an antitone function $f : 2^S \to 2^S$, then there exists $T \subseteq S$ such that either $\mathcal{U} = \{T\}$ or $\mathcal{U} = \{T, f(T)\}$.*

Proof. Let $f^2(\mathcal{U}) \coloneqq \{f^2(U) : U \in \mathcal{U}\}$; this is a global attractor invariant set for f^2. Suppose $\mathcal{U}$ is a $(2k+1)$-cycle for some $k \geqslant 0$: then $f^2(\mathcal{U})$ is also a $(2k+1)$-cycle. Since by Lemma 5.1.5 every global attractor invariant set for a monotone function must contain a fixed point, and since f being antitone implies f^2 being monotone, it must be $k = 0$, and thus $\mathcal{U} = f^2(\mathcal{U}) = \{T\}$ must be a global fixed-point attractor for f^2. Suppose now $\mathcal{U}$ is a $(2k)$-cycle for some $k \geqslant 1$: then $f^2(\mathcal{U})$ consists of two k-cycles. Since one of the two cycles must be a fixed point by Lemma 5.1.5, it must be $k = 1$ and thus $\mathcal{U} = \{T, f(T)\}$ for some $T \subseteq S$. □

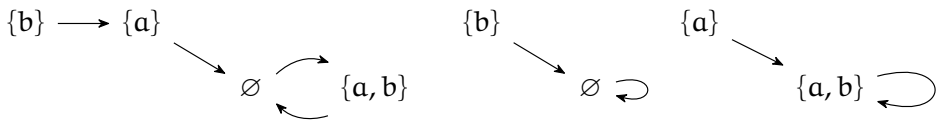
Example 5.3.1. Let $S = \{a, b\}$ and $f : 2^S \to 2^S$ given by:

$$f(\varnothing) = \{a, b\}; \quad f(\{a\}) = \varnothing; \quad f(\{b\}) = \{a\}; \quad f(\{a, b\}) = \varnothing.$$

f is clearly antitone and in the dynamics, we have a global 2-cycle attractor $\{\varnothing, S\}$: see Figure 17a. Consider now $f^2 : 2^S \to 2^S$, given by

$$f^2(\varnothing) = \varnothing; \quad f^2(\{a\}) = \{a, b\}; \quad f^2(\{b\}) = \varnothing; \quad f^2(\{a, b\}) = \{a, b\}.$$

f^2 has a global attractor consisting of two fixed points; see Figure 17b. ⌟



(a) A global 2-cycle attractor.

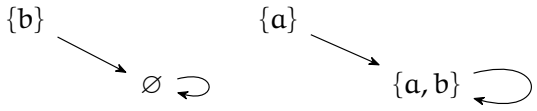
(b) A global attractor consisting of two fixed points.

Figure 17. Representation of the dynamics of Example 5.3.1.

From the proof of Proposition 5.3.5, we deduce that an antitone function $f : 2^S \to 2^S$ has a global 2-cycle attractor if and only if $f^2 : 2^S \to 2^S$ has a global attractor consisting of two fixed points.

The rest of this section is devoted to proving that deciding whether a reactantless RS has a global 2-cycle attractor reduces to the problem of Theorem 5.2.4 for inhibitorless RSs, and it is, therefore, **PSPACE**-complete as well. We begin with an example illustrating the workings of the reduction we later provide in Theorem 5.3.6.

Example 5.3.2. Let $S = \{a, b\}$ and $\mathcal{A} = (S, A)$ an inhibitorless RS where $A = \{(\{a\}, \varnothing, \{a, b\})\}$. As already seen in Example 5.3.1, in the dynamics of $\mathcal{A}$ there are two fixed points that form together a global attractor:



as in the dynamics of Figure 17b. We want to construct a reactantless RS that can reproduce the dynamics of $\mathcal{A}$ for states $\varnothing \subsetneq T \subsetneq S$ and transform the global attractor of $\mathcal{A}$, consisting of two fixed points, into a global 2-cycle attractor. We thus construct $\mathcal{B} = (S', B)$ where $S' = \{a, b, \heartsuit, \spadesuit\}$ and B is given by the following reactions:

$$(\varnothing, \{a, \heartsuit\}, \{a, \heartsuit\})$$
$$(\varnothing, \{b, \heartsuit\}, \{b, \heartsuit\})$$
$$(\varnothing, \{a, \spadesuit\}, \{a, b, \spadesuit\})$$
$$(\varnothing, \{a, b, \heartsuit\}, \{a, b, \heartsuit, \spadesuit\})$$
$$(\varnothing, \{a, b, \spadesuit\}, \{a, b, \heartsuit, \spadesuit\})$$
$$(\varnothing, \{\heartsuit, \spadesuit\}, \{a, b, \heartsuit, \spadesuit\}).$$

It is straightforward to verify that $\mathrm{res}_{\mathcal{B}}(\{b, \spadesuit\}) = \{a, \heartsuit\}$ and $\mathrm{res}_{\mathcal{B}}(\{a, \spadesuit\}) = \{b, \heartsuit\}$, thus

$$\mathrm{res}^2_{\mathcal{B}}(\{b, \spadesuit\}) = \varnothing \quad \text{and} \quad \mathrm{res}^2_{\mathcal{B}}(\{a, \spadesuit\}) = \{a, b, \spadesuit\}.$$

Note that in the original inhibitorless RS $\mathcal{A}$ we have $\mathrm{res}_{\mathcal{A}}(\{b\}) = \varnothing$ and $\mathrm{res}_{\mathcal{A}}(\{a\}) = \{a, b\}$, thus $\mathcal{B}$ can reproduce the dynamics of $\mathcal{A}$ in two steps starting from the states $\{a, \spadesuit\}$ and $\{b, \spadesuit\}$ and going through the states $\{a, \heartsuit\}$ and $\{b, \heartsuit\}$. The last three reactions of B ensure that there is a global 2-cycle attractor, as all the states except for $\{a, \spadesuit\}$, $\{b, \spadesuit\}$, and $\{b, \heartsuit\}$ reach the 2-cycle $\{\varnothing, S'\}$ in one step, which makes it a global 2-cycle attractor. The dynamics of $\mathcal{B}$ is represented in Figure 18.

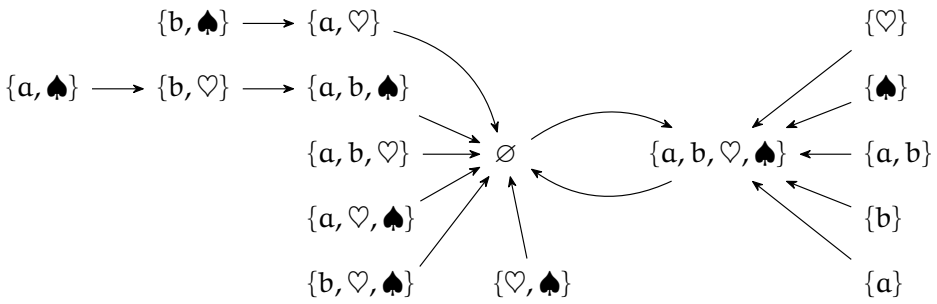


Figure 18. Dynamics of the RS $\mathcal{B}$ in Example 5.3.2.

In Theorem 5.3.6, we extend and generalize the construction of Example 5.3.2 to any $\mathcal{A} \in \mathcal{RS}(\infty, 0)$ to reduce the problem of deciding whether $\mathcal{U} = \{\varnothing, S\}$ is a global attractor in inhibitorless RSs to the problem of deciding the existence of a global 2-cycle attractor in reactantless RSs.

Theorem 5.3.6. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$, deciding if there exists a global 2-cycle attractor is* **PSPACE***-complete.*

Proof. Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ such that $\varnothing$ and S are fixed points, we want to construct in polynomial time a RS $\mathcal{B} \in \mathcal{RS}(0, \infty)$ such that $\{\varnothing, S\}$ is a global attractor for $\mathcal{A}$ if and only if there exists a global 2-cycle attractor for $\mathcal{B}$. We reduce from the problem of deciding if $\mathcal{U} = \{\varnothing, S\}$ is a global attractor in inhibitorless RSs (see Theorem 5.2.4). We construct a reactantless RS $\mathcal{B} \coloneqq (S', B)$, with $S' \coloneqq S \cup \{\heartsuit, \spadesuit\}$ and B is given by the following reactions:

$$(\varnothing, \{s, \heartsuit\}, \{s, \heartsuit\}) \qquad \text{for } s \in S \tag{108}$$
$$(\varnothing, R_a \cup \{\spadesuit\}, P_a \cup \{\spadesuit\}) \qquad \text{for } a = (R_a, \varnothing, P_a) \in A \tag{109}$$
$$(\varnothing, S \cup \{\heartsuit\}, S \cup \{\heartsuit, \spadesuit\}) \tag{110}$$
$$(\varnothing, S \cup \{\spadesuit\}, S \cup \{\heartsuit, \spadesuit\}) \tag{111}$$
$$(\varnothing, \{\heartsuit, \spadesuit\}, S \cup \{\heartsuit, \spadesuit\}). \tag{112}$$

Claim 6. All states of $\mathcal{B}$ of the forms $\{\spadesuit\}$,$\{\heartsuit\}$, $S \cup \{\heartsuit\}$, $S \cup \{\spadesuit\}$, T, and $T \cup \{\heartsuit, \spadesuit\}$, for all $T \subseteq S$, reach $\{\varnothing, S'\}$ in one step. Furthermore, $\mathrm{res}_{\mathcal{B}}(\varnothing) = S'$ and $\mathrm{res}_{\mathcal{B}}(S') = \varnothing$.

Proof. We immediately note that for any $T \subseteq S$ we have $\mathrm{res}_{\mathcal{B}}(T) = S \cup \{\heartsuit, \spadesuit\} = S'$ since reaction (112) is enabled, and $\mathrm{res}_{\mathcal{B}}(T \cup \{\heartsuit, \spadesuit\}) = \varnothing$ since no reaction is enabled. By reactions (110) and (111), we also have $\mathrm{res}_{\mathcal{B}}(\{\spadesuit\}) = \mathrm{res}_{\mathcal{B}}(\{\heartsuit\}) = S \cup \{\heartsuit, \spadesuit\} = S'$. Furthermore, since $\mathrm{res}_{\mathcal{A}}(\varnothing) = \varnothing$, then $R_a \neq \varnothing$ for each $a \in A$, thus $S \cup \{\heartsuit\}$ does not enable any reaction of Group (109), thus $\mathrm{res}_{\mathcal{B}}(S \cup \{\heartsuit\}) = \varnothing$, as well as $\mathrm{res}_{\mathcal{B}}(S \cup \{\spadesuit\}) = \varnothing$. Finally, since all the reactions are enabled by $\varnothing$, and no reaction is enabled by $S' = S \cup \{\heartsuit, \spadesuit\}$, we have that $\mathrm{res}_{\mathcal{B}}(\varnothing) = S'$ and $\mathrm{res}_{\mathcal{B}}(S') = \varnothing$. See also Figure 19 for a visual representation of the dynamics. ∎

Claim 7. The states $\{\spadesuit\}$, $\{\heartsuit\}$, $\{\heartsuit, \spadesuit\}$, S, $S \cup \{\heartsuit\}$, T, and $T \cup \{\heartsuit, \spadesuit\}$, for all $\varnothing \subsetneq T \subsetneq S$, cannot be reached from any states.

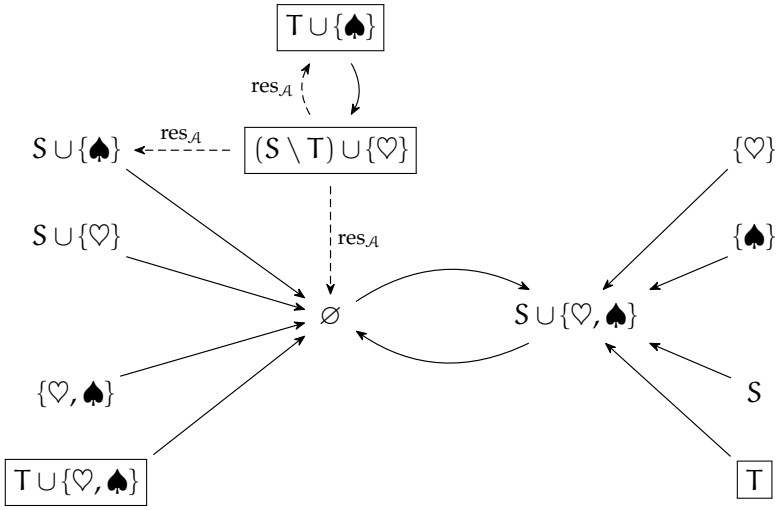


Figure 19. Dynamics of the RS $\mathcal{B}$ in the reduction of Theorem 5.3.6. The states T, $T \cup \{♡, ♠\}$, $T \cup \{♠\}$ and $(S \setminus T) \cup \{♡\}$ are a synthetic representation of the $2^S - 2$ states (one fore each $\varnothing \subsetneq T \subsetneq S$) of each type. The boxes around states $T \cup \{♠\}$ and $(S \setminus T) \cup \{♡\}$ hide the more refined dynamics for those states; dashed arcs represent the three existing possibilities for the dynamics of the states belonging to the boxes, as described after Claim 7.

Proof. From the definition of RS, there are no reactions of the type $(R_a, \varnothing, \varnothing)$, because in any case, they do not affect the dynamics of $\mathcal{A}$. Therefore $\mathrm{en}_{\mathcal{A}}(T) = \varnothing$ if and only if $\mathrm{res}_{\mathcal{A}}(T) = \varnothing$. This implies that Group (109) of the reactions of $\mathcal{B}$ does not contain any reactions of the form $(\varnothing, R_a \cup \{♠\}, \{♠\})$, implying that the state $\{♠\}$ cannot be reached from any state. With a similar reasoning we deduce that also states $\{♡\}$, $\{♡, ♠\}$, S, and T for all $\varnothing \subsetneq T \subsetneq S$ cannot be reached from any state.

Furthermore, none of the states of the form $T \cup \{♡, ♠\}$ with $\varnothing \subsetneq T \subsetneq S$ can be reached from any state: indeed, suppose for a contradiction that $\mathrm{res}_{\mathcal{B}}(T') = T \cup \{♡, ♠\}$ for some $T' \subseteq S'$ and $\varnothing \subsetneq T \subsetneq S$. In order to obtain ♡ in the product, T' must enable some reactions from Group (108); and to obtain ♠, it must also enable reactions from Group (109). This implies $♡, ♠ \notin T'$, thus $T' \subseteq S$ and thus, by Claim 6, $\mathrm{res}_{\mathcal{B}}(T') = S \cup \{♡, ♠\}$, which is a contradiction because by hypothesis $T \subsetneq S$. Finally, $S \cup \{♡\}$ cannot be reached from any state U because this would require all and only the reactions from Group (108) to be enabled in U, which can happen only if $U = \{♠\}$; but then reaction (110) is enabled as well, and indeed $\mathrm{res}_{\mathcal{B}}(\{♠\}) = S'$ by Claim 6. ■

It remains to determine the dynamics for the states of the form $T \cup \{♠\}$ and $T \cup \{♡\}$ for some $\varnothing \subsetneq T \subsetneq S$. Because of the reactions from Group (108), we obtain

$$\mathrm{res}_{\mathcal{B}}(T \cup \{♠\}) = (S \setminus T) \cup \{♡\}; \tag{113}$$

and because of the reactions from Group (109), in turn we have

$$\mathrm{res}_{\mathcal{B}}((S \setminus T) \cup \{♡\}) = \begin{cases} \mathrm{res}_{\mathcal{A}}(T) \cup \{♠\} & \text{if } \mathrm{en}_{\mathcal{A}}(T) \neq \varnothing \\ \varnothing & \text{otherwise,} \end{cases} \tag{114}$$

since $(S \setminus T) \cup \{\heartsuit\}$ enables $(\varnothing, R_a \cup \{\spadesuit\}, P_a \cup \{\spadesuit\})$ if and only if $(S \setminus T) \cap R_a = \varnothing$, which is the case if and only if $R_a \subseteq T$ and thus T enables $(R_a, \varnothing, P_a)$ in $\mathcal{A}$. As remarked in Claim 7, we have that $\mathrm{res}_{\mathcal{A}}(T) = \varnothing$ if and only if $\mathrm{en}_{\mathcal{A}}(T) = \varnothing$, which is true if and only if $\mathrm{res}_{\mathcal{B}}((S \setminus T) \cup \{\heartsuit\}) = \varnothing$. We have obtained the following formula:

$$\mathrm{res}^2_{\mathcal{B}}(T \cup \{\spadesuit\}) = \begin{cases} \mathrm{res}_{\mathcal{A}}(T) \cup \{\spadesuit\} & \text{if } \mathrm{en}_{\mathcal{A}}(T) \neq \varnothing \\ \varnothing & \text{otherwise.} \end{cases} \tag{115}$$

Therefore, iterating (115), if $\mathrm{res}^i_{\mathcal{A}}(T) \notin \{\varnothing, S\}$ for all $i = 1, \ldots, k-1$ we obtain

$$\mathrm{res}^{2k}_{\mathcal{B}}(T \cup \{\spadesuit\}) = \begin{cases} \mathrm{res}^k_{\mathcal{A}}(T) \cup \{\spadesuit\} & \text{if } \mathrm{res}^k_{\mathcal{A}}(T) \neq \varnothing \\ \varnothing & \text{otherwise.} \end{cases} \tag{116}$$

Note that the states of the form $T \cup \{\heartsuit\}$ with $\varnothing \subsetneq T \subsetneq S$ coincide with the states of the form $(S \setminus T) \cup \{\heartsuit\}$; in particular, any such state $T \cup \{\heartsuit\}$ is reached from $(S \setminus T) \cup \{\spadesuit\}$ by Equation (113), and reaches either $\varnothing$ or $\mathrm{res}_{\mathcal{A}}(S \setminus T) \cup \{\spadesuit\}$ according to Equation (114). In Figure 19, the states of the form $T \cup \{\heartsuit\}$ and $T \cup \{\spadesuit\}$ are compactly represented as boxed states, and their dynamics are not completely represented for the sake of readability.

We observe that the only candidate global 2-cycle attractor for $\mathcal{B}$ is $\{\varnothing, S'\}$, as it is a 2-cycle by Claim 6 and it is the only candidate global attractor by Claim 7 and the discussion below its proof. The next claim gives us the thesis.

Claim 8. The state $\{\varnothing, S\}$ is a global attractor for $\mathcal{A}$ if and only if $\{\varnothing, S'\}$ is a global attractor for $\mathcal{B}$.

Proof. $\boxed{\Rightarrow}$ Let $\varnothing \subsetneq T \subsetneq S$: in this case, we already proved in Claim 6 that T and $T \cup \{\heartsuit, \spadesuit\}$ reach $\{\varnothing, S'\}$ in one step. We need to prove that $T \cup \{\spadesuit\}$ and $T \cup \{\heartsuit\}$ reach $\{\varnothing, S'\}$. By hypothesis, $\exists k \in \mathbb{N}$ such that $\mathrm{res}^k_{\mathcal{A}}(T) \in \{\varnothing, S\}$. Let k be the minimum number that satisfies this property, implying that $\mathrm{res}^i_{\mathcal{A}}(T) \notin \{\varnothing, S\}$ for $i = 1, \ldots, k-1$. Since by hypothesis $\mathrm{res}^k_{\mathcal{A}}(T) \in \{\varnothing, S\}$, there are two cases: if $\mathrm{res}^k_{\mathcal{A}}(T) = S$, then, by Equation (116), $\mathrm{res}^{2k}_{\mathcal{B}}(T \cup \{\spadesuit\}) = S \cup \{\spadesuit\}$, implying that $\mathrm{res}^{2k+1}_{\mathcal{B}}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{B}}(S \cup \{\spadesuit\}) = \varnothing$ (by Claim 6). Otherwise, $\mathrm{res}^k_{\mathcal{A}}(T) = \varnothing$ and thus $\mathrm{res}^{2k}_{\mathcal{B}}(T \cup \{\spadesuit\}) = \varnothing$ by Equation (116). In any case, $T \cup \{\spadesuit\}$ reaches $\{\varnothing, S'\}$ in at most $2k+1$ steps. For the state $T \cup \{\heartsuit\}$, we can reduce to the previous case using Equation (114). Together with Claim 6, we obtain that if $\{\varnothing, S\}$ is a global attractor for $\mathcal{A}$ then $\{\varnothing, S'\}$ is a global attractor for $\mathcal{B}$.

$\boxed{\Leftarrow}$ Let $\varnothing \subsetneq T \subsetneq S$: by hypothesis, there exists $k \in \mathbb{N}$ such that $\mathrm{res}^k_{\mathcal{B}}(T \cup \{\spadesuit\}) \in \{\varnothing, S'\}$. Let k be the minimum number that satisfies that property. We want to prove that T is always attracted by $\{\varnothing, S\}$. We define two cases, depending on whether k is even or odd.

1) $k = 2m$. We have $\mathrm{res}^i_{\mathcal{A}}(T) \notin \{\varnothing, S\}$ for all $i = 1, \ldots, m-1$ as otherwise $k = 2m$ would not be the minimum. We can apply Equation (116) and obtain

$$\mathrm{res}^{2(m-1)}_{\mathcal{B}}(T \cup \{\spadesuit\}) = \mathrm{res}^{m-1}_{\mathcal{A}}(T) \cup \{\spadesuit\}.$$

Applying Equation (113) to this result, we also obtain

$$\mathrm{res}_{\mathcal{B}}^{2(m-1)+1}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{B}}^{2m-1}(T \cup \{\spadesuit\}) = S \setminus \mathrm{res}_{\mathcal{A}}^{m-1}(T) \cup \{\heartsuit\}.$$

Since $\varnothing \subsetneq \mathrm{res}_{\mathcal{A}}^{m-1}(T) \subsetneq S$, we have $\varnothing \subsetneq S \setminus \mathrm{res}_{\mathcal{A}}^{m-1}(T) \subsetneq S$, thus reaction (111) is not enabled by $(S \setminus \mathrm{res}_{\mathcal{A}}^{m-1}(T)) \cup \{\heartsuit\}$, implying in turn that $\heartsuit \notin \mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\})$, and thus $\mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\}) \neq S \cup \{\heartsuit, \spadesuit\}$. But since $\mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\}) \in \{\varnothing, S'\}$ then $\mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\}) = \varnothing$. Suppose now for a contradiction that $\mathrm{res}_{\mathcal{A}}^{m}(T) \neq \varnothing$: then it would also be $\mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{B}}((S \setminus \mathrm{res}_{\mathcal{A}}^{m-1}(T)) \cup \{\heartsuit\}) \neq \varnothing$, a contradiction. We deduce that $\mathrm{res}_{\mathcal{A}}^{m}(T) = \varnothing$, thus T is attracted by $\{\varnothing, S\}$.

2) $k = 2m + 1$. As before, $\mathrm{res}_{\mathcal{A}}^{i}(T) \notin \{\varnothing, S\}$ for $i = 1, \ldots, m-1$. Thus we have $\mathrm{res}_{\mathcal{B}}^{2m-1}(T \cup \{\spadesuit\}) = (S \setminus \mathrm{res}_{\mathcal{A}}^{m-1}(T)) \cup \{\heartsuit\}$. Since $\mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{B}}^{k-1}(T \cup \{\spadesuit\}) \neq \varnothing$ then $\mathrm{res}_{\mathcal{A}}^{m}(T) \neq \varnothing$. Thus $\mathrm{res}_{\mathcal{B}}^{2m}(T \cup \{\spadesuit\}) = \mathrm{res}_{\mathcal{A}}^{m}(T) \cup \{\spadesuit\}$. Suppose for a contradiction that $\mathrm{res}_{\mathcal{A}}^{m}(T) \subsetneq S$, then, by Equation (113), $\mathrm{res}_{\mathcal{B}}^{2m+1}(T \cup \{\spadesuit\}) = S \setminus \mathrm{res}_{\mathcal{A}}^{m}(T) \cup \{\heartsuit\} \notin \{\varnothing, S'\}$, a contradiction by the definition of k. We deduce that $\mathrm{res}_{\mathcal{A}}^{m}(T) = S$, thus T is attracted by $\{\varnothing, S\}$.

Summing up, we proved that if $\{\varnothing, S'\}$ is a global attractor for $\mathcal{B}$ then $\{\varnothing, S\}$ is a global attractor for $\mathcal{A}$. ■

Claim 8, together with Claim 6, directly implies that there exists global 2-cycle attractor for $\mathcal{B}$ if and only if $\{\varnothing, S\}$ is a global attractor for $\mathcal{A}$. We also remark that the map $\mathcal{A} \mapsto \mathcal{B}$ can be constructed in polynomial time. By Theorem 5.2.4, deciding whether $\{\varnothing, S\}$ is a global attractor is **PSPACE**-complete, thus the thesis follows. □

5.4 CYCLES OF REACTANTLESS RSS

In this section, we prove **PSPACE**-hardness for problems concerning cycles in the class of reactantless RS. To this aim, we will reduce from the same problems studied in Section 5.2 for inhibitorless RSs. We start by recalling a result from [100] that will prove useful in showing that deciding whether a state is part of a cycle for inhibitorless RS is **PSPACE**-hard.

Lemma 5.4.1 ([100]). *Let $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ be such that $\bigcup\{P_a : a \in A\} = S$ and $\mathrm{res}_{\mathcal{A}}(S) = S$. Then, there exists $\mathcal{B} = (S', A') \in \mathcal{RS}(0, 1)$ such that, for any $T \subseteq S$, the following condition holds:*

$$\forall t \in \mathbb{N} \quad \mathrm{res}_{\mathcal{B}}^{2t}(T) = \mathrm{res}_{\mathcal{A}}^{t}(T) \quad \wedge \quad S \subseteq \mathrm{res}_{\mathcal{B}}^{2t+1}(T).$$

Furthermore, the map $\mathcal{A} \mapsto \mathcal{B}$ can be computed in polynomial time.

From now on we will follow the same reduction as in [100, Theorem 9].

Remark 5.4.1. Given any $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$ we can always construct another RS in $\mathcal{RS}(\infty, 0)$ having the same behaviour of $\mathcal{A}$ and satisfying the hypothesis of Lemma 5.4.1. Indeed, let $\mathcal{A}' = (S', A') \in \mathcal{RS}(\infty, 0)$ be such that $S' \coloneqq S \cup \{\spadesuit\}$ for some $\spadesuit \notin S$, and $A' \coloneqq A \cup \{(S', \varnothing, S')\}$. Therefore, since $(S', \varnothing, S') \in A'$ we have $\bigcup\{P'_{a'} : a' \in A'\} = S'$ and $\mathrm{res}_{\mathcal{A}'}(S') = S'$. Furthermore, $\mathrm{res}_{\mathcal{A}}(T) = \mathrm{res}_{\mathcal{A}'}(T)$ for all $T \subseteq S$.

Theorem 5.4.2. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(0, 1)$ and a state $T \subseteq S$, it is* **PSPACE***-complete to decide whether T is part of a cycle.*

Proof. The proof is entirely analogous to that of [100, Theorem 9]: we reduce from the problem of deciding whether T is part of a cycle in *inhibitorless* RSs.

Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$, we first construct another RS $\mathcal{A}' = (S', A') \in \mathcal{RS}(\infty, 0)$ from $\mathcal{A}$ using Remark 5.4.1; we then construct $\mathcal{A}'' \in \mathcal{RS}(0, 1)$ from $\mathcal{A}'$ using Lemma 5.4.1. Notice that the map $\mathcal{A} \mapsto \mathcal{A}''$ can be computed in polynomial time.

Given a state $T \subseteq S$, we have $\operatorname{res}^t_{\mathcal{A}}(T) = T$ for some $t \in \mathbb{N}$ if and only if $\operatorname{res}^{2t}_{\mathcal{A}''}(T) = T$. Furthermore, it holds that $\operatorname{res}^{2s+1}_{\mathcal{A}''}(T) \neq T$ for all $s \in \mathbb{N}$, since $\spadesuit \in S' \subseteq \operatorname{res}^{2s+1}_{\mathcal{A}''}(T)$ but $\spadesuit \notin S$ and, in particular, $\spadesuit \notin T$. Therefore, T is part of a cycle in the RS $\mathcal{A}''$ if and only if the same happens in $\mathcal{A}$. Hence, by Theorem 5.2.7, deciding if T is part of a cycle for $\mathcal{RS}(0, 1)$ is **PSPACE**-hard. □

Since $\mathcal{RS}(0, 1)$ is a subclass of $\mathcal{RS}(0, \infty)$, we have the following result.

Corollary 5.4.3. *Given $\mathcal{A} = (S, A) \in \mathcal{RS}(0, \infty)$ and a state $T \subseteq S$, it is* **PSPACE***-complete to decide whether T is part of a cycle.*

The reduction of [100, Lemma 8], does not apply to the problems of sharing cycles, thus we need a new construction, provided in Theorem 5.4.4.

Theorem 5.4.4. *Given $\mathcal{A}, \mathcal{B} \in \mathcal{RS}(0, \infty)$ over the same background set S, it is* **PSPACE***-complete to decide if (1) $\mathcal{A}$ and $\mathcal{B}$ share all their cycles, and (2) $\mathcal{A}$ and $\mathcal{B}$ have a common cycle.*

Proof. Given $\mathcal{A} = (S, A) \in \mathcal{RS}(\infty, 0)$, we construct $\mathcal{A}' := (S', A') \in \mathcal{RS}(0, \infty)$, with $S' := S \cup \overline{S}$ where $\overline{S} := \{\overline{s} : s \in S\}$, i.e. the entities in $\overline{S}$ are in a one-to-one correspondence with those of S. In the following, given $T \subseteq S$ we denote by $\overline{T} := \{\overline{s} : s \in T\} \subseteq \overline{S}$. The following reactions give the set A':

$$
\begin{aligned}
&(\varnothing, \{s\}, \{\overline{s}\}) && \text{for } s \in S \\
&a' := (\varnothing, \overline{R_a}, P_a) && \text{for } a = (R_a, \varnothing, P_a) \in A.
\end{aligned}
$$

Clearly, $\mathcal{A}'$ is a reactantless RS. Given a state $V \cup \overline{U} \subseteq S'$, where $V \subseteq S$ and $\overline{U} \subseteq \overline{S}$, we notice that

$$\operatorname{res}_{\mathcal{A}'}(V \cup \overline{U}) = \operatorname{res}_{\mathcal{A}}(S \setminus U) \cup \overline{S \setminus V}, \tag{117}$$

since an entity $\overline{s} \in \overline{S}$ is generated if and only if $s \notin V$, and $a' \in A'$ is enabled if and only if $\overline{U} \cap \overline{R_a} = \varnothing \Leftrightarrow \overline{R_a} \subseteq \overline{S \setminus U} \Leftrightarrow R_a \subseteq S \setminus U$, i.e. the reaction $a \in A$ is enabled by $S \setminus U$. Furthermore, note that applying two times the reaction (117) to $V \cup \overline{S \setminus U}$, since $S \setminus (S \setminus V) = V$, we obtain:

$$\operatorname{res}^2_{\mathcal{A}'}\left(V \cup \overline{S \setminus U}\right) = \operatorname{res}_{\mathcal{A}}(V) \cup \overline{S \setminus \operatorname{res}_{\mathcal{A}}(U)}. \tag{118}$$

Let now $\mathcal{B} = (S, B) \in \mathcal{RS}(\infty, 0)$ be over the same background set S as $\mathcal{A}$; we construct $\mathcal{B}' = (S', B') \in \mathcal{RS}(0, \infty)$ in much the same way as we did to construct $\mathcal{A}'$.

Claim 1. $\mathcal{A}, \mathcal{B}$ share all their cycles if and only if $\mathcal{A}', \mathcal{B}'$ share all their cycles.

$\boxed{\Rightarrow}$ Suppose $\mathcal{A}, \mathcal{B}$ share all their cycles and let $\left\{C_i \cup \overline{S \setminus D_i} : i \in \mathbb{Z}_k\right\}$ be a cycle for $\mathcal{A}'$, i.e. $\mathrm{res}_{\mathcal{A}'}\left(C_i \cup \overline{S \setminus D_i}\right) = C_{i+1} \cup \overline{S \setminus D_{i+1}}$ for all $i \in \mathbb{Z}_k$, where $i+1$ must be interpreted modulo k. From Equation (118), we get the following relations:

$$\forall i \in \mathbb{Z}_k, \quad C_{i+2} = \mathrm{res}_{\mathcal{A}}(C_i) \quad \text{and} \quad D_{i+2} = \mathrm{res}_{\mathcal{A}}(D_i)$$

and from Equation (117) we obtain that, for all $i \in \mathbb{Z}_k$, $C_{i+1} = \mathrm{res}_{\mathcal{A}}(D_i)$, thus $D_{i+2} = C_{i+1}$. Therefore we can rewrite the cycles as $\{C_i \cup \overline{S \setminus C_{i-1}} : i \in \mathbb{Z}_k\}$. We divide two cases as follows.

(i) If k is even, then $C_0, C_2, \ldots, C_{k-2}$ and $C_1, C_3, \ldots, C_{k-1}$ are two cycles of $\mathcal{A}$. By hypothesis, they are also cycles of $\mathcal{B}$, therefore $\left\{C_i \cup \overline{S \setminus C_{i-1}} : i \in \mathbb{Z}_k\right\}$ is a cycle for $\mathcal{B}'$.

(ii) If k is odd, then $C_0, C_2, \ldots, C_{k-1}, C_1, C_3, \ldots, C_{k-2}$ is a cycle of $\mathcal{A}$, and we can conclude as in (i).

We thus proved that every cycle of $\mathcal{A}'$ is also a cycle of $\mathcal{B}'$. The same proof holds if we change the roles of $\mathcal{A}'$ and $\mathcal{B}'$, so we obtain the thesis.

$\boxed{\Leftarrow}$ Suppose $\mathcal{A}', \mathcal{B}'$ share all their cycles and let $\{E_i : i \in \mathbb{Z}_m\}$ be a cycle for $\mathcal{A}$. Consider the cycle in the dynamics of $\mathcal{A}'$ containing $E_0 \cup \overline{S \setminus E_0}$. This cycle is given by m blocks of the following type:

$$\cdots \to E_i \cup \overline{S \setminus E_i} \to E_{i+1} \cup \overline{S \setminus E_i} \to \cdots$$

for all $i \in \mathbb{Z}_m$. By hypothesis, this is also a cycle for $\mathcal{B}'$, therefore

$$\mathrm{res}_{\mathcal{B}'}\left(E_i \cup \overline{S \setminus E_i}\right) = E_{i+1} \cup \overline{S \setminus E_i} \quad \forall i \in \mathbb{Z}_m$$

and by Equation (117), we obtain that $E_{i+1} = \mathrm{res}_{\mathcal{B}}(E_i)$ for all $i \in \mathbb{Z}_m$, i.e. $\{E_i : i \in \mathbb{Z}_m\}$ is a cycle for $\mathcal{B}$. Therefore every cycle of $\mathcal{A}$ is also a cycle of $\mathcal{B}$. The same reasoning also holds if we exchange the roles of $\mathcal{A}$ and $\mathcal{B}$, so we obtain the thesis.

With similar reasoning as in Claim 1, it is also possible to prove the following claim.

Claim 2. $\mathcal{A}$ and $\mathcal{B}$ have a common cycle if and only if $\mathcal{A}'$ and $\mathcal{B}'$ have a common cycle.

We finally remark that the map $(\mathcal{A}, \mathcal{B}) \mapsto (\mathcal{A}', \mathcal{B}')$ can be constructed in polynomial time. By Theorems 5.2.8 and 5.2.9, the problems considered are **PSPACE**-hard for $\mathcal{RS}(\infty, 0)$, thus by Claims 1 and 2 the thesis follows. □

5.5 CONCLUSIONS

We have provided a complete landscape of the complexity of problems related to the existence and identification of cycles and global attractors in reactantless and inhibitorless RSs. Specifically, we proved that the problems of deciding whether a given state is a global attractor and whether there exists a fixed point attractor in a given RS, which are **PSPACE**-complete in general RSs, become polynomial when

they are restricted to the reactantless or inhibitorless case. Furthermore, again in contrast with the general case, no global cycle attractor of length at least 2 can exist in an inhibitorless RS, and no such a cycle of length strictly greater than 2 exists in a reactantless RS; and it is **PSPACE**-complete to decide whether a global cycle attractor of length exactly 2 exists in a reactantless RS. Finally, we proved that the problems of deciding if a state is part of a cycle and comparing the cycles of two rRSs (do they have a common cycle? Do they have exactly the same cycles?) remain **PSPACE**-complete even when they are restricted to the inhibitorless and reactantless case. It remains open to study the complexity of the latter problems in the case of additive RSs, where we expect the hardness to reduce, as shown in Chapter 4 for other problems. Finding other classes besides additive RSs where those problems exhibit lower complexity is also of theoretical interest.

Part II

COMPUTABILITY OF PURE REACTION AUTOMATA

6

PURE REACTION AUTOMATA

In this chapter, we introduce the new class of Pure Reaction Automata (PRA), and study a new update manner in the context of Reaction Automata (RA), called maximal reactive (mr) manner. PRA differ from the standard model in that they do not have permanence: the entities that are not consumed by the reactions happening at a certain state are not conserved in the result states. The introduction of a new model of RA raises the question of its computational power compared to the already existing models. Our results can be summarised as follows.

1. The set of languages accepted by PRA working in a maximal reactive manner contains the set of languages accepted by standard RA working in the same manner (Theorem 6.2.1), as well as the set of languages accepted by RA working in a maximal parallel (mp) manner (a manner already investigated by [261]) (Theorem 6.2.2). See Figure 20 for a graphical summary of these results.

2. The restricted class of *deterministic* PRA can compute any recursive function from $\mathbb{N}^k$ to $\mathbb{N}$ (Theorem 6.3.8).

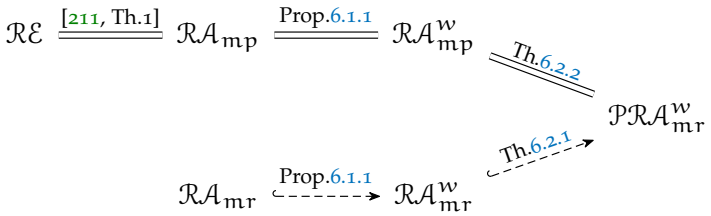


Figure 20. Graphical summary of the results of Section 6.1.1. $\mathcal{RE}$ represents the set of recursively enumerable languages. $\mathcal{RA}_X$, $\mathcal{PRA}_X$ are the set of languages recognized by RA, PRA working in manner $X \in \{mp, mr\}$. The apex w stands for weakly accepted (see Definition 6.1.5). The dashed arrow $\dashrightarrow$ denotes language set inclusion.

Chapter organization. Section 10.2 compares RA and P automata. Section 6.1 recalls the preliminary notions on RA. Section 6.2 defines and investigates the computational power of PRA as language acceptors, while Section 6.3 examines the computational power of a restricted class of PRA as devices for computing partial recursive functions. Section 6.4 summarises the results and some directions for future research.

This chapter is based on the following publication: R. Ascone, G. Bernardini, E. Formenti, F. Leiter, L. Manzoni. Pure Reaction Automata. In: *Natural Computing*, 2024, [18].

6.1 PRELIMINARIES

Let S be a finite alphabet. We denote by S^* the set of words over S, that is, all finite sequences of elements of S, with $\varepsilon \in S^*$ denoting the empty word consisting of zero letters. A *multiset* over S is defined as a function $V : S \to \mathbb{N}$ such that $V(a) \in \mathbb{N}$ is the multiplicity of $a \in S$ in the multiset; $S^\#$ denotes the set of all multisets over S. Given V and W two multisets over S, we can define a partial order given by multiset inclusion and the following multiset operators:

- inclusion: $V \leqslant W$ if $V(a) \leqslant W(a)\ \forall\ a \in S$;
- sum: $(V + W)(a) \coloneqq V(a) + W(a)\ \forall\ a \in S$;
- intersection: $(V \cap W)(a) \coloneqq \min\{V(a), W(a)\}\ \forall\ a \in S$;
- difference: $(V - W)(a) \coloneqq V(a) - W(a)\ \forall\ a \in S$ (only defined for $W \leqslant V$);
- symmetric difference: $(V \triangle W)(a) \coloneqq (V + W)(a) - (V \cap W)(a)\ \forall\ a \in S$.

We also define a belonging relation for multisets: given $V \in S^\#$ and $a \in S$, $a \in V$ if and only if $V(a) \geqslant 1$, i.e. a letter belongs to V if and only if it has positive multiplicity. This allows us to define the *set* underlying a multiset $V \in S^\#$ as the set of letters with nonzero multiplicity:

$$\mathrm{set}(V) \coloneqq \{a \in S \mid a \in V\}.$$

The following properties follow immediately for any $V, W \in S^\#$:

$$\mathrm{set}(V + W) = \mathrm{set}(V) \cup \mathrm{set}(W), \quad \mathrm{set}(V \cap W) = \mathrm{set}(V) \cap \mathrm{set}(W).$$

Furthermore, $V \leqslant W$, implies $\mathrm{set}(V) \subseteq \mathrm{set}(W)$, but the converse is not true: e.g. given $V = \{a, a, b\}$ and $W = \{a, b, c\}$, it holds $\mathrm{set}(V) \subseteq \mathrm{set}(W)$ but $V \not\leqslant W$.

A set $U \subseteq S$ naturally corresponds to a multiset V_U such that $V_U(a) = 1$ if $a \in U$ and $V_U(a) = 0$ otherwise. In particular, for each $a \in S$, we will often denote the multiset $V_{\{a\}}$ simply by a. We will denote the empty multiset by $0 \in S^\#$. The total number of elements in a multiset $V \in S$ is defined as $\|V\| \coloneqq \sum_{a \in S} V(a)$.

Remark 6.1.1. The following map:

$$\begin{aligned} S^\# &\longrightarrow \mathbb{N}^{|S|} \\ V &\longmapsto (V(a_1), \ldots, V(a_n)) \end{aligned}$$

is an isomorphism of monoids; we will denote such isomorphism by $S^\# \cong \mathbb{N}^{|S|}$.

6.1.1 *Reaction Automata*

In this section, we recall the definition of RA given by [261] and we introduce a new policy for enabling reactions.

Definition 6.1.1 (Reaction)**.** Given an alphabet of reactants S, a *reaction over* S is a triple $\mathbf{a} = (R_\mathbf{a}, I_\mathbf{a}, P_\mathbf{a})$, where $R_\mathbf{a} \in S^\#$ is the multiset of *reactants*, $I_\mathbf{a} \subseteq S$ is the set of *inhibitors* and $P_\mathbf{a} \in S^\#$ is the multiset of *products*. The set of all reactions over S is denoted by $\mathrm{rac}(S)$.

A crucial assumption in the RS model is that if a reactant is present at a certain *state* $T \in S^{\#}$, then its quantity is always enough for all the reactions that use it to take place, provided the respective inhibitors are not present. In other words, reactions do not conflict even if they share some resources. This assumption is no longer in place in the model of RA, for which there is only a certain quantity of each of the reactants. It is therefore necessary to specify a criterion (called a *manner*) that decides which of several conflicting reactions take place. In this chapter, we focus on two manners, provided in Definition 6.1.2. Before defining such manners, we need to introduce a few operations and relations among reactions.

Let $\mathbf{a} = (R_{\mathbf{a}}, I_{\mathbf{a}}, P_{\mathbf{a}}), \mathbf{b} = (R_{\mathbf{b}}, I_{\mathbf{b}}, P_{\mathbf{b}}) \in \mathrm{rac}(S)$; we define the sum of the two reactions $\mathbf{a} + \mathbf{b} \coloneqq (R_{\mathbf{a}} + R_{\mathbf{b}}, I_{\mathbf{a}} \cup I_{\mathbf{b}}, P_{\mathbf{a}} + P_{\mathbf{b}})$. Furthermore, we define a partial order over all possible reactions over S: $\mathbf{a} \leqslant_r \mathbf{b}$ if and only if $R_{\mathbf{a}} \leqslant R_{\mathbf{b}}$ and $I_{\mathbf{a}} \subseteq I_{\mathbf{b}}$. In other words, a reaction is greater than another when it is more restrictive, i.e. it requires more reactants and there are more elements capable of disabling it. With this relation, we get that $\mathbf{a} =_r \mathbf{b}$ if and only if $R_{\mathbf{a}} = R_{\mathbf{b}}$ and $I_{\mathbf{a}} = I_{\mathbf{b}}$, but we do not impose any conditions on the products: in particular, it could hold $\mathbf{a} =_r \mathbf{b}$ with $P_{\mathbf{a}} \neq P_{\mathbf{b}}$. We remark that this partial order is different from the one proposed by [109] for reactions within RSs.

Given a finite set $\mathbf{A} \subseteq \mathrm{rac}(S)$, we denote by $\langle \mathbf{A} \rangle$ the abelian semigroup generated by the elements of $\mathbf{A}$:

$$\langle \mathbf{A} \rangle \coloneqq \{\lambda_1 \mathbf{a}_1 + \cdots + \lambda_n \mathbf{a}_n \mid \mathbf{a}_i \in \mathbf{A}, \lambda_i \in \mathbb{N} \quad \forall i = 1, \ldots, n\}.$$

Note that any element of $\langle \mathbf{A} \rangle$ is a reaction that can be interpreted as a multiset of reactions from $\mathbf{A}$, where the coefficients $\lambda_1, \ldots, \lambda_n$ give the multiplicity of each reaction. We will thus denote a multiset of reactions that are enabled in a certain state as a single reaction from $\langle \mathbf{A} \rangle$.

Definition 6.1.2 (Manners)**.** Let $\mathbf{a} = (R_{\mathbf{a}}, I_{\mathbf{a}}, P_{\mathbf{a}}) \in \mathrm{rac}(S)$ and $T \in S^{\#}$, we say that $\mathbf{a}$ is *enabled* in T if $R_{\mathbf{a}} \leqslant T$ and $I_{\mathbf{a}} \cap \mathrm{set}(T) = \varnothing$. Given $\mathbf{A}$ a finite set of reactions over S and $\mathbf{a} \in \langle \mathbf{A} \rangle$ enabled in T, then:

1. $\mathbf{a}$ is enabled in T in a *maximally parallel manner* (mp) if there exists no $\mathbf{c} \in \langle \mathbf{A} \rangle$ such that $\mathbf{a} + \mathbf{c}$ is enabled in T, i.e. $\mathbf{a}$ is maximal w.r.t. addition;

2. $\mathbf{a}$ is enabled in T in a *maximally reactive manner* (mr) if there exists no $\mathbf{b} \in \langle \mathbf{A} \rangle$ such that $\mathbf{a} <_r \mathbf{b}$ and $\mathbf{b}$ is enabled in T , i.e. $\mathbf{a}$ is maximal w.r.t the partial order $\leqslant_r$.

We denote by $\mathrm{En}^X_{\mathbf{A}}(T)$ the set of reactions from $\langle \mathbf{A} \rangle$ enabled in T in manner $X \in \{\mathrm{mp}, \mathrm{mr}\}$.

Remark 6.1.2. The notion of mp manner given in Definition 6.1.2 is equivalent to the one given by [261]. Indeed, given $\alpha, \beta \in \mathbf{A}^{\#}$, consider the corresponding elements $\mathbf{a}, \mathbf{b} \in \langle \mathbf{A} \rangle$, identified by $\alpha \mapsto \sum_{\mathbf{d} \in \mathbf{A}} \alpha(\mathbf{d})\mathbf{d} \in \langle \mathbf{A} \rangle$. If $\beta > \alpha$ is enabled by T, then $\mathbf{c} = \mathbf{b} - \mathbf{a}$ is such that $\mathbf{a} + \mathbf{c} = \mathbf{b}$ is enabled by T; the viceversa is obtained in a similar way.

Remark 6.1.3. At first sight, the definitions of manners mp and mr are hard to tell apart. However, the two criteria are distinct, and in particular, mr is stronger than

mp in the sense that $\mathrm{En}_{\mathbf{A}}^{\mathrm{mr}}(T) \subseteq \mathrm{En}_{\mathbf{A}}^{\mathrm{mp}}(T)$ for any $\mathbf{A} \subseteq \mathrm{rac}(S)$ and any state T. In other words, if a reaction $\mathbf{a} \in \langle \mathbf{A} \rangle$ is enabled in a maximally reactive manner, it is also enabled in a maximally parallel manner. Indeed, the existence of $\mathbf{c} \in \langle \mathbf{A} \rangle$ s.t. $\mathbf{a} + \mathbf{c}$ is enabled in T would imply that a reaction greater than $\mathbf{a}$ is enabled in T, leading to a contradiction. The converse is not always true: see Example 6.1.1.

In RSs, the state resulting from a set of reactions is simply defined as the union of the products of all the reactions. In RA, the reactants that are not consumed by the reactions that take place remain in the resulting states. We make this concept precise in Definition 6.1.3.

Definition 6.1.3 (Result)**.** The *result* of a set of reactions $\mathbf{A}$ on a state T in a manner X is a set of states, denoted by $\mathrm{Res}_{\mathbf{A}}^{X}(T)$, defined as follows:

$$\mathrm{Res}_{\mathbf{A}}^{X}(T) = \{P_{\mathbf{a}} + (T - R_{\mathbf{a}}) \mid \mathbf{a} = (R_{\mathbf{a}}, I_{\mathbf{a}}, P_{\mathbf{a}}) \in \mathrm{En}_{\mathbf{A}}^{X}(T)\}.$$

When $\mathrm{En}_{\mathbf{A}}^{X}(T) = \varnothing$, we define $\mathrm{Res}_{\mathbf{A}}^{X}(T) = \{T\}$, that is, if no multiset of reactions from $\langle \mathbf{A} \rangle$ is enabled in T, then T remains unchanged.

Example 6.1.1. Let $S = \{w_1, w_2, \heartsuit\}$, $\mathbf{A} = \{\mathbf{a}_1 = (w_1, \varnothing, \heartsuit), \mathbf{a}_2 = (w_1 + w_2, \varnothing, w_2)\}$, and consider a state $T = w_1 + w_2$. Then the set of reactions enabled in T in a maximally parallel manner is $\mathrm{En}_{\mathbf{A}}^{\mathrm{mp}}(T) = \{\mathbf{a}_1, \mathbf{a}_2\}$, as $\nexists \mathbf{c} \in \langle \mathbf{A} \rangle$ such that $\mathbf{a}_1 + \mathbf{c}$ is enabled in T; and the set of reactions enabled in T in a maximally reactive manner is $\mathrm{En}_{\mathbf{A}}^{\mathrm{mr}}(T) = \{\mathbf{a}_2\}$, because $\mathbf{a}_2 >_r \mathbf{a}_1$ and thus $\mathbf{a}_1$ is not mr-enabled. The corresponding results in the two manners are thus $\mathrm{Res}_{\mathbf{A}}^{\mathrm{mp}}(T) = \{\heartsuit + w_2, w_2\}$, $\mathrm{Res}_{\mathbf{A}}^{\mathrm{mr}}(T) = \{w_2\}$.

We are now in a position to formally define RA.

Definition 6.1.4 ([261])**.** A *reaction automaton* (RA) $\mathcal{A}$ is a five-tuple $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, f)$, where S is a finite set of reactants, called the *background set* of $\mathcal{A}$; $\Sigma \subseteq S$ is the *input alphabet* of $\mathcal{A}$; $\mathbf{A} \subseteq \mathrm{rac}(S)$ is a finite set of reactions over S; $D_0 \in S^{\#}$ is the *initial multiset*; and $f \in S$ is a special symbol which indicates the final state.

Definition 6.1.5. Consider a RA $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, f)$, a word $w = w_1 \cdots w_n \in \Sigma^*$ and a manner $X \in \{\mathrm{mp}, \mathrm{mr}\}$. An *interactive process in $\mathcal{A}$ with input w in manner X* is an infinite sequence $\pi = D_0, \ldots, D_i, \ldots$ where

$$\begin{cases} D_{i+1} \in \mathrm{Res}_{\mathbf{A}}^{X}(w_{i+1} + D_i) & \text{for } 0 \leqslant i \leqslant n-1 \\ D_{i+1} \in \mathrm{Res}_{\mathbf{A}}^{X}(D_i) & \text{for } i \geqslant n. \end{cases}$$

By $\mathrm{IP}_X(\mathcal{A}, w)$ we denote the set of all interactive processes in $\mathcal{A}$ with input w in manner X. We say that a process π *strongly accepts* w if there exists $m \geqslant n = |w|$ such that $f \in D_m$ and $\mathrm{En}_{\mathbf{A}}^{X}(D_m) = \varnothing$ (see also Example 6.1.3). By $\mathrm{AIP}_X(\mathcal{A}, w)$ we denote the set of all processes $\pi \in \mathrm{IP}_X(\mathcal{A}, w)$ such that π strongly accepts w. The *language strongly accepted* by $\mathcal{A}$ is defined as

$$L_X(\mathcal{A}) = \{w \in \Sigma^* \mid \mathrm{AIP}_X(\mathcal{A}, w) \neq \varnothing\}.$$

The set of languages strongly accepted by RA working in manner X is denoted by $\mathcal{RA}_X$: $L \in \mathcal{RA}_X$ if and only if there exists a RA working in manner X that strongly accepts L.

We say that a process π *weakly accepts* w if there exists $m \geqslant n = |w|$ such that $f \in D_m$ and $D_k = D_m$ for all $k \geqslant m$ (see also Example 6.1.4). By $\mathrm{AIP}^w_X(\mathcal{A}, w)$ we denote the set of all processes $\pi \in \mathrm{IP}_X(\mathcal{A}, w)$ such that π weakly accepts w. The *language weakly accepted* by $\mathcal{A}$ is defined as

$$L^w_X(\mathcal{A}) = \{w \in \Sigma^* \mid \mathrm{AIP}^w_X(\mathcal{A}, w) \neq \varnothing\}.$$

The set of languages weakly accepted by RA working in manner X is denoted by $\mathcal{RA}^w_X$.

We will sometimes represent an interactive process π with the following "arrow notation", which extends the notation proposed by [261]:

$$\pi : D_0 \xrightarrow[w_1]{\mathbf{a}_1} D_1 \xrightarrow[w_2]{\mathbf{a}_2} D_2 \xrightarrow[w_3]{\mathbf{a}_3} \cdots D_{n-1} \xrightarrow[w_n]{\mathbf{a}_n} D_n \xrightarrow{\mathbf{a}_{n+1}} D_{n+1} \xrightarrow{\mathbf{a}_{n+2}} \cdots$$

where $D_{i-1} \xrightarrow[w_i]{\mathbf{a}_i} D_i$ means w_i is the input letter at state D_{i-1}, $\mathbf{a}_i \in \langle \mathbf{A} \rangle$ is the reaction enabled in $D_{i-1} + w_i$ which takes place, and $D_i \in \mathrm{Res}^X_{\mathbf{A}}(w_i + D_{i-1})$.

Example 6.1.2. If $w = \varepsilon$ the empty word, then $|w| = n = 0$, thus an interactive process accepting strongly ε is of the form $D_0 \xrightarrow{\mathbf{a}_1} D_1 \xrightarrow{\mathbf{a}_2} D_2 \xrightarrow{\mathbf{a}_3} \cdots D_m \to D_m \to D_m \cdots$, where D_m does not enable any reaction.

Note that in Definition 6.1.5 we call *strong* the acceptance condition proposed by [261]; the weak notion of acceptance will be needed for the definition of PRA (see Section 6.2).

In Proposition 6.1.1 we show that the weak acceptance criterion extends the strong criterion in the following sense: given an automaton that strongly accepts a certain language, it is always possible to construct another automaton that weakly accepts the same language. Examples 6.1.3 and 6.1.4 provide intuition on this result before we formally prove it in Proposition 6.1.1: Example 6.1.3 provides an automaton accepting words with the strong criterion, while Example 6.1.4 constructs another automaton that accepts exactly the same language as in Example 6.1.3 but using the weak criterion.

Example 6.1.3. Given an input alphabet $\Sigma = \{a, b\}$, a background set $S = \{a, b, s_0, s_1, f\}$ and a set of reactions $\mathbf{A} = \{\mathbf{a}_1 = (s_0 + a, \varnothing, s_1), \mathbf{a}_2 = (s_1 + b, \varnothing, s_0), \mathbf{a}_3 = (s_0, \varnothing, f), \mathbf{a}_4 = (b, \varnothing, b)\}$, let $\mathcal{A} = (S, \Sigma, \mathbf{A}, s_0, f)$ be a RA working in mr manner that accepts words with the strong criterion. We show that the language strongly accepted by $\mathcal{A}$ is $L_{mr}(\mathcal{A}) = \{(ab)^n : n \in \mathbb{N}\}$. Consider the input word $w = abab \in \Sigma^*$, we obtain the following process:

$$s_0 \xrightarrow[a]{\mathbf{a}_1} s_1 \xrightarrow[b]{\mathbf{a}_2} s_0 \xrightarrow[a]{\mathbf{a}_1} s_1 \xrightarrow[b]{\mathbf{a}_2} s_0 \xrightarrow{\mathbf{a}_3} f.$$

The computation stops since $\mathrm{En}^{mr}_{\mathbf{A}}(f) = \varnothing$. A trivial extension of this argument proves that $\{(ab)^n : n \in \mathbb{N}\} \subseteq L_{mr}(\mathcal{A})$.

To show the other inclusion, observe that a word $w \notin \{(ab)^n : n \in \mathbb{N}\}$ if and only if at least one of the following cases happens: (i) aa occurs in w; (ii) bb occurs in w; (iii) w starts with b; (iv) w ends with a. In case (i), when the second consecutive a is fed to the process, it ends up in the state $s_1 + a$: the computation stops because

$\mathrm{En}^{mr}_{\mathbf{A}}(s_1 + a) = \varnothing$, but since $f \notin s_1 + a$, w is not strongly accepted. In case (ii), after reading the second consecutive b, the process reaches the state $s_0 + b$: then $s_0 + b \xrightarrow[b]{\mathbf{a}_3+\mathbf{a}_4} b + f$ and $\mathrm{En}^{mr}_{\mathbf{A}}(b + f) = \{\mathbf{a}_4\}$, thus the process keeps looping in this state and w is not strongly accepted because $\mathrm{En}^{mr}_{\mathbf{A}}(b + f) \neq \varnothing$. The same happens in case (iii), when b is added to the initial state s_0. Finally, in case (iv), after reading the last a, the process ends up in state s_1: the computation stops because $\mathrm{En}^{mr}_{\mathbf{A}}(s_1) = \varnothing$, but since $f \notin s_1$, w is not strongly accepted. We can thus conclude that $L_{mr}(\mathcal{A}) = \{(ab)^n : n \in \mathbb{N}\}$.

Note that using the weak notion of acceptance instead of the strong one for the reaction automaton $\mathcal{A}$ of Example 6.1.3 would imply $\{(ab)^n : n \in \mathbb{N}\} \subsetneq L^{w}_{mr}(\mathcal{A})$, since for example the string $w = b \notin \{(ab)^n : n \in \mathbb{N}\}$ is weakly accepted by the following process:

$$s_0 \xrightarrow[b]{\mathbf{a}_3+\mathbf{a}_4} b + f \xrightarrow{\mathbf{a}_4} b + f \xrightarrow{\mathbf{a}_4} \cdots .$$

Example 6.1.4 shows that it is anyway possible to construct another RA $\mathcal{B}$ such that $L^{w}_{mr}(\mathcal{B}) = \{(ab)^n : n \in \mathbb{N}\}$.

Example 6.1.4. Given an input alphabet $\Sigma = \{a, b\}$ and a background set $S = \{a, b, s_0, s_1, \spadesuit, \clubsuit, f\}$ let $\mathcal{B} = (S, \Sigma, \mathbf{B}, s_0 + \clubsuit, f)$ be a RS working in mr manner and using the weak acceptance criterion, with $\mathbf{B}$ consisting of the following reactions:

$$\begin{aligned}
\mathbf{a}^{\spadesuit}_1 &= (s_0 + a, \{\spadesuit\}, s_1 + \spadesuit) & \mathbf{a}^{\clubsuit}_1 &= (s_0 + a, \{\clubsuit\}, s_1 + \clubsuit)\\
\mathbf{a}^{\spadesuit}_2 &= (s_1 + b, \{\spadesuit\}, s_0 + \spadesuit) & \mathbf{a}^{\clubsuit}_2 &= (s_1 + b, \{\clubsuit\}, s_0 + \clubsuit)\\
\mathbf{a}^{\spadesuit}_3 &= (s_0, \{\spadesuit\}, f + \spadesuit) & \mathbf{a}^{\clubsuit}_3 &= (s_0, \{\clubsuit\}, f + \clubsuit)\\
\mathbf{a}^{\spadesuit}_4 &= (b, \{\spadesuit\}, b + \spadesuit) & \mathbf{a}^{\clubsuit}_4 &= (b, \{\clubsuit\}, b + \clubsuit)\\
\mathbf{r}^{\spadesuit} &= (\spadesuit, \varnothing, 0) & \mathbf{r}^{\clubsuit} &= (\clubsuit, \varnothing, 0)\,.
\end{aligned}$$

We show that $\mathcal{B}$ weakly accepts the same language that is strongly accepted by $\mathcal{A}$ in Example 6.1.3. Consider $w = ab \in \{(ab)^n : n \in \mathbb{N}\}$. We obtain the following process:

$$s_0 + \clubsuit \xrightarrow[a]{\mathbf{a}^{\spadesuit}_1+\mathbf{r}^{\clubsuit}} s_1 + \spadesuit \xrightarrow[b]{\mathbf{a}^{\clubsuit}_2+\mathbf{r}^{\spadesuit}} s_0 + \clubsuit \xrightarrow{\mathbf{a}^{\spadesuit}_3+\mathbf{r}^{\clubsuit}} f + \spadesuit \xrightarrow{\mathbf{r}^{\spadesuit}} f.$$

The computation stops since $\mathrm{En}^{mr}_{\mathbf{B}}(f) = \varnothing$, thus $\mathrm{Res}^{mr}_{\mathbf{B}}(f) = \{f\}$ and w is weakly accepted. Consider now the string $w = aa \notin \{(ab)^n : n \in \mathbb{N}\}$. We obtain the following process:

$$s_0 + \clubsuit \xrightarrow[a]{\mathbf{a}^{\spadesuit}_1+\mathbf{r}^{\clubsuit}} s_1 + \spadesuit \xrightarrow[a]{\mathbf{r}^{\spadesuit}} s_1 + a$$

and the computation stops since $\mathrm{En}^{mr}_{\mathbf{B}}(s_1 + a) = \varnothing$; aa is not weakly accepted because $f \notin s_1 + a$. Similarly, it is easy to see that any string in case (i) of Example 6.1.3 is not weakly accepted. Consider the string $w = b \notin \{(ab)^n : n \in \mathbb{N}\}$. We obtain the following process:

$$s_0 + \clubsuit \xrightarrow[b]{\mathbf{a}^{\spadesuit}_3+\mathbf{a}^{\spadesuit}_4+\mathbf{r}^{\clubsuit}} b + f + \spadesuit \xrightarrow{\mathbf{a}^{\clubsuit}_4+\mathbf{r}^{\spadesuit}} b + f + \clubsuit \xrightarrow{\mathbf{a}^{\spadesuit}_4+\mathbf{r}^{\clubsuit}} b + f + \spadesuit \to \cdots$$

The states $b + f + \clubsuit$ and $b + f + \spadesuit$ keep alternating, so the string is not weakly accepted. Similarly, one can prove that any string starting with b (case (iii) of Example 6.1.3) is not weakly accepted. Working out the other cases in a similar fashion, it is easy to show that $L^w_{mr}(\mathcal{B}) = \{(ab)^n : n \in \mathbb{N}\}$.

In Example 6.1.4 we were able to simulate the processes of $\mathcal{A}$ using the following rule: for any step m, if $En^X_{\mathbf{A}}(D_m) = \varnothing$ then the computation stops also in $\mathcal{B}$; if, instead, $En^X_{\mathbf{A}}(D_m) \neq \varnothing$, then the computation in $\mathcal{B}$ keeps alternating between states containing $\spadesuit$ and states containing $\clubsuit$. The proof of the following Proposition 6.1.1 relies on a generalization of this argument.

Proposition 6.1.1. *Given any reaction automaton $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, f)$ working in a manner $X \in \{mp, mr\}$, there exists a reaction automaton $\mathcal{B}$ working in manner X such that $L_X(\mathcal{A}) = L^w_X(\mathcal{B})$.*

Proof. Let $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, f)$ and let $\spadesuit, \clubsuit \notin S$ be two symbols that are not in the alphabet of $\mathcal{A}$. We define $\mathcal{B} = (S \cup \{\spadesuit, \clubsuit\}, \Sigma, \mathbf{B}, D_0 + \clubsuit, f)$ the RA such that $\mathbf{B} \coloneqq \mathbf{A}^\spadesuit \cup \mathbf{A}^\clubsuit \cup \{\mathbf{r}^\spadesuit, \mathbf{r}^\clubsuit\}$, where

$$\begin{aligned}
\mathbf{A}^\spadesuit &= \{\mathbf{a}^\spadesuit = (R, I \cup \{\spadesuit\}, P + \spadesuit) \mid \mathbf{a} = (R, I, P) \in \mathbf{A}\} \\
\mathbf{A}^\clubsuit &= \{\mathbf{a}^\clubsuit = (R, I \cup \{\clubsuit\}, P + \clubsuit) \mid \mathbf{a} = (R, I, P) \in \mathbf{A}\} \\
\mathbf{r}^\spadesuit &= (\spadesuit, \varnothing, 0) \\
\mathbf{r}^\clubsuit &= (\clubsuit, \varnothing, 0)\,.
\end{aligned}$$

Given $\mathbf{a} = \sum_{j=1}^n \lambda_j \mathbf{a}_j \in \langle \mathbf{A} \rangle$ we define $c_{\mathbf{a}} = \sum_{j=1}^n \lambda_j$, $\mathbf{a}^\spadesuit = \sum_{j=1}^n \lambda_j \mathbf{a}_j^\spadesuit \in \langle \mathbf{A}^\spadesuit \rangle$ and similarly $\mathbf{a}^\clubsuit = \sum_{j=1}^n \lambda_j \mathbf{a}_j^\clubsuit \in \langle \mathbf{A}^\clubsuit \rangle$.

We want to prove that $\mathcal{B}$ weakly accepts the same language that is strongly accepted by $\mathcal{A}$. Given a process $\pi \in IP_X(\mathcal{A}, w)$

$$\pi : D_0 \xrightarrow[w_1]{\mathbf{a}_1} D_1 \xrightarrow[w_2]{\mathbf{a}_2} D_2 \xrightarrow[w_3]{\mathbf{a}_3} \dots D_{n-1} \xrightarrow[w_n]{\mathbf{a}_n} D_n \xrightarrow{\mathbf{a}_{n+1}} D_{n+1} \cdots$$

if n is even, there is a one-to-one correspondence with the process $\bar{\pi} \in IP_X(\mathcal{B}, w)$

$$\begin{aligned}
\bar{\pi} : D_0 + \clubsuit \xrightarrow[w_1]{\mathbf{a}_1^\spadesuit + \mathbf{r}^\clubsuit} D_1 + c_{\mathbf{a}_1} \spadesuit \xrightarrow[w_2]{\mathbf{a}_2^\clubsuit + c_{\mathbf{a}_1} \mathbf{r}^\spadesuit} D_2 + c_{\mathbf{a}_2} \clubsuit \xrightarrow[w_3]{\mathbf{a}_3^\spadesuit + c_{\mathbf{a}_2} \mathbf{r}^\clubsuit} \cdots \\
\cdots D_{n-1} + c_{\mathbf{a}_{n-1}} \spadesuit \xrightarrow[w_n]{\mathbf{a}_n^\clubsuit + c_{\mathbf{a}_{n-1}} \mathbf{r}^\spadesuit} D_n + c_{\mathbf{a}_n} \clubsuit \xrightarrow{\mathbf{a}_{n+1}^\spadesuit + c_{\mathbf{a}_n} \mathbf{r}^\clubsuit} \cdots .
\end{aligned}$$

If n is odd, a process corresponding to π can be obtained similarly. Since $\spadesuit, \clubsuit$ are not present in any of the reactant sets of $\mathbf{A}^\spadesuit$ and $\mathbf{A}^\clubsuit$, the reaction $\mathbf{r}^\spadesuit$ (respectively, $\mathbf{r}^\clubsuit$) is enabled as many times as the multiplicity of $\spadesuit$ (respectively, $\clubsuit$) in any given state; this is independent of the manner $X \in \{mp, mr\}$ we are working with. In particular, any process in $IP_X(\mathcal{B}, w)$ ends in a state D_m s.t. $D_k = D_m$ for all $k > m$ if and only if at some step no reactions are enabled (as otherwise, $\spadesuit$ and $\clubsuit$ would keep alternating). Therefore $AIP_X(\mathcal{A}, w)$ is in a bijection with $AIP^w_X(\mathcal{B}, w)$. We conclude that $L_X(\mathcal{A}) = L^w_X(\mathcal{B})$. □

Corollary 6.1.2. *$\mathcal{RA}_{mr} \subseteq \mathcal{RA}^w_{mr}$ and $\mathcal{RA}_{mp} \subseteq \mathcal{RA}^w_{mp}$.*

Proof. Follows directly from Proposition 6.1.1. □

Corollary 6.1.3. *Every recursively enumerable language is weakly accepted by a reaction automaton working in a maximally parallel manner.*

Proof. Follows from Proposition 6.1.1 and [211, Corollary 1]. □

We thus proved that the weak acceptance criterion extends the strong criterion. This will be useful in the next section to demonstrate the computational power of PRA.

6.2 PURE REACTION AUTOMATA

In this section, we introduce a different kind of reaction automata, which differs from the model introduced by [261] by how the result of a reaction is defined: instead of transferring the reactants that are not consumed by the reactions in the result states, we define the next states to consist only of the products of the reactions, similar to what is done in reaction systems. We make this concept formal in Definition 6.2.1.

Definition 6.2.1 (Pure result)**.** The *pure result* of a finite set of reactions $\mathbf{A}$ on a state T in a manner X is

$$\widehat{\mathrm{Res}}^X_{\mathbf{A}}(T) = \{P_{\mathbf{a}} \mid \mathbf{a} = (R_{\mathbf{a}}, I_{\mathbf{a}}, P_{\mathbf{a}}) \in \mathrm{En}^X_{\mathbf{A}}(T)\},$$

and we define $\widehat{\mathrm{Res}}^X_{\mathbf{A}}(T) = \{0\}$ when $\mathrm{En}^X_{\mathbf{A}}(T) = \varnothing$.

Example 6.2.1. Let $S = \{w_1, w_2, \heartsuit\}$, $\mathbf{A} = \{\mathbf{a}_1 = (w_1, \varnothing, \heartsuit), \mathbf{a}_2 = (w_1 + w_2, \varnothing, w_2)\}$ and consider a state $T = w_1 + w_2$ as in Example 6.1.1. Recall that $\mathrm{Res}^{mp}_{\mathbf{A}}(T) = \{\heartsuit + w_2, w_2\}$ (since $\mathrm{En}^{mp}_{\mathbf{A}}(T) = \{\mathbf{a}_1, \mathbf{a}_2\}$); in contrast, the pure result of T in mp manner is $\widehat{\mathrm{Res}}^{mp}_{\mathbf{A}}(T) = \{\heartsuit, w_2\}$. In particular, reaction $\mathbf{a}_1$ does not consume the reactant w_2 that is present in T: the pure result of $\mathbf{a}_1$ only consists of $\heartsuit$ and w_2 is lost, while in Example 6.1.1 w_2 was preserved in the result.

We name this new kind of reaction automata *Pure Reaction Automata* (PRA). We define interactive processes in PRA in much the same way as standard RA, as specified by Definition 6.2.2.

Definition 6.2.2. Let $\mathcal{M} = (S, \Sigma, \mathbf{A}, D_0, f)$ be a PRA, $w = w_1 \cdots w_n \in \Sigma^*$ and $X \in \{mp, mr\}$. An interactive process in $\mathcal{M}$ with input w in manner X is an infinite sequence $\pi = D_0, \ldots, D_i, \ldots$ where

$$\begin{cases} D_{i+1} \in \widehat{\mathrm{Res}}^X_{\mathbf{A}}(w_{i+1} + D_i) & \text{for } 0 \leqslant i \leqslant n-1 \\ D_{i+1} \in \widehat{\mathrm{Res}}^X_{\mathbf{A}}(D_i) & \text{for } i \geqslant n. \end{cases}$$

Exactly as for RA, we say that π weakly accepts w if there exists $m \geqslant n = |w|$ such that $f \in D_m$ and $D_k = D_m$ for all $k \geqslant m$. We also define $\mathrm{IP}_X(\mathcal{M}, w)$, $\mathrm{AIP}^w_X(\mathcal{M}, w)$, and $L^w_X(\mathcal{M})$ in the same way as for RA. The set of languages weakly accepted by PRA working in manner X is denoted by $\mathcal{PRA}^w_X$.

Example 6.2.2. Given an input alphabet $\Sigma = \{a, b\}$, a background set $S = \{a, b, s_0, s_1, f\}$ and a set of reactions $\mathbf{A} = \{\mathbf{a}_1 = (s_0, \{b\}, s_1), \mathbf{a}_2 = (s_1 + b, \varnothing, s_0), \mathbf{a}_3 = (s_0, \{a, b\}, f), \mathbf{a}_4 = (f, \varnothing, f)\}$, let $\mathcal{M} = (S, \Sigma, \mathbf{A}, s_0, f)$ be a PRA working in mr manner. We show that the language (weakly) accepted by $\mathcal{M}$ is $L^w_{mr}(\mathcal{M}) = \{(ab)^n : n \in \mathbb{N}\}$. Consider the input word $w = abab \in \Sigma^*$, we obtain the following process:

$$s_0 \xrightarrow[a]{\mathbf{a}_1} s_1 \xrightarrow[b]{\mathbf{a}_2} s_0 \xrightarrow[a]{\mathbf{a}_1} s_1 \xrightarrow[b]{\mathbf{a}_2} s_0 \xrightarrow{\mathbf{a}_3} f \xrightarrow{\mathbf{a}_4} f \xrightarrow{} \cdots .$$

Therefore the string $abab$ is accepted. Note that the pure result of reaction $\mathbf{a}_1$ applied at states $s_0 + a$ consists only of s_1, even if a is not consumed by the reaction. We also remark that the step $s_0 \xrightarrow{\mathbf{a}_3} f$ generates f instead of s_1 because $\mathbf{a}_3 >_r \mathbf{a}_1$ and thus $\mathrm{En}^{mr}_{\mathbf{B}}(s_0) = \{\mathbf{a}_3\}$. A trivial extension of this argument proves that $\{(ab)^n : n \in \mathbb{N}\} \subseteq L^w_{mr}(\mathcal{M})$. Consider now the string $w = aa \notin \{(ab)^n : n \in \mathbb{N}\}$, as in Example 6.1.4. We obtain the following process:

$$s_0 \xrightarrow[a]{\mathbf{a}_1} s_1 \underset{a}{\to} 0 \to 0 \to \cdots$$

and the computation stops since $\mathrm{En}^{mr}_{\mathbf{B}}(s_1 + a) = \mathrm{En}^{mr}_{\mathbf{B}}(0) = \varnothing$, thus aa is not accepted. Working out the other cases listed in Example 6.1.3 similarly, it is easy to see that $L^w_{mr}(\mathcal{M}) = \{(ab)^n : n \in \mathbb{N}\}$.

Note that a non-pure automaton with the same sets of reactions, initial state and final element would not accept, e.g., the string $ab \in \{(ab)^n : n \in \mathbb{N}\}$ since the only interactive process with input ab is:

$$s_0 \xrightarrow[a]{\mathbf{a}_1} s_1 + a \xrightarrow[b]{\mathbf{a}_2} s_0 + a \xrightarrow{\mathbf{a}_1} s_1 + a \to s_1 + a \cdots$$

and the computation stops without accepting ab since, as previously remarked, $\mathrm{En}^{mr}_{\mathbf{B}}(s_1 + a) = \varnothing$.

Theorem 6.2.1. *Given any reaction automaton $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, f)$ working in a maximally reactive manner, there exists a pure reaction automaton $\mathcal{M}$ working in a maximally reactive manner such that $L^w_{mr}(\mathcal{A}) = L^w_{mr}(\mathcal{M})$.*

Proof. We define a PRA $\mathcal{M} = (S \cup S', \Sigma, \mathbf{A}', D_0, f)$ operating in a maximally reactive manner such that:

- $S' = \{x' \mid x \in S\}$ is a set in a bijection with the elements of S (a "copy" of S). From now on, given a multiset X over S, X' will be naturally defined as the multiset consisting of the copies of the elements of X;
- Σ is the same input alphabet as $\mathcal{A}$;
- $\mathbf{A}' = \mathbf{A}_p \cup \mathbf{A}_c$, where $\mathbf{A}_p = \{(x, \varnothing, x + x') \mid x \in S\}$ and $\mathbf{A}_c = \{(R + R', I', P + P') \mid (R, I, P) \in \mathbf{A}\}$;
- D_0 is the same initial state as $\mathcal{A}$;
- $f \in S$ is the same final element as $\mathcal{A}$.

Claim 9. For any state W of $\mathcal{A}$, it holds $W \in \mathrm{Res}^{mr}_{\mathbf{A}}(V)$ if and only if $W + W' \in \widehat{\mathrm{Res}}^{mr}_{\mathbf{A}'}(V + V')$.

Proof. Given a reaction $\mathbf{a} \in \mathbf{A}$, we denote the corresponding reaction in $\mathbf{A}_c$ by $\mathbf{a}'$. We prove the two implications.

- $\Rightarrow$) If $W \in \mathrm{Res}^{mr}_{\mathbf{A}}(V)$ then $\exists \mathbf{a} = (R_{\mathbf{a}}, I_{\mathbf{a}}, P_{\mathbf{a}}) \in \mathrm{En}^{mr}_{\mathbf{A}}(V)$ such that $P_{\mathbf{a}} + V - R_{\mathbf{a}} = W$. Consider $\mathbf{a}'' \coloneqq \mathbf{a}' + (V - R_{\mathbf{a}}, \varnothing, V - R_{\mathbf{a}} + V' - R'_{\mathbf{a}}) = (V + R'_{\mathbf{a}}, I'_{\mathbf{a}}, W + W') \in \langle \mathbf{A}' \rangle$. Clearly, $V + V'$ enables $\mathbf{a}''$; we want to show that it is mr enabled. Suppose for a contradiction that there exists $\mathbf{b}'' \in \langle \mathbf{A}' \rangle$ such that $\mathbf{b}'' >_r \mathbf{a}''$ and $\mathbf{b}''$ is enabled by $V + V'$, then $\mathbf{b}''$ is of the form $\mathbf{b}'' = \mathbf{b}' + (R, \varnothing, R + R')$ for some $\mathbf{b} = (R_{\mathbf{b}}, I_{\mathbf{b}}, P_{\mathbf{b}}) \in \langle \mathbf{A} \rangle$ and some $R \in S^{\#}$. Looking at the reactants, we have that $V + V' \geqslant R_{\mathbf{b}} + R'_{\mathbf{b}} + R \geqslant V + R'_{\mathbf{a}}$, which implies $R_{\mathbf{b}} + R = V \Rightarrow R_{\mathbf{b}} \leqslant V$ and $R'_{\mathbf{b}} \geqslant R'_{\mathbf{a}}$. Furthermore, looking at the inhibitors, we obtain that $I'_{\mathbf{b}} \supseteq I'_{\mathbf{a}}$, thus $\mathbf{b} >_r \mathbf{a}$ and $\mathbf{b}$ is enabled by V. Since $\mathbf{a}$ is mr enabled we get a contradiction, thus $\mathbf{a}'' \in \mathrm{En}^{mr}_{\mathbf{A}'}(V + V')$, hence $W + W' \in \widehat{\mathrm{Res}}^{mr}_{\mathbf{A}'}(V + V')$.
- $\Leftarrow$) If $W + W' \in \widehat{\mathrm{Res}}^{mr}_{\mathbf{A}'}(V + V')$ then $\exists \mathbf{a}'' = \mathbf{a}' + (R, \varnothing, R + R') \in \mathrm{En}^{mr}_{\mathbf{A}'}(V + V')$, for some $\mathbf{a} \in \langle \mathbf{A} \rangle$ and $R \in S^{\#}$, such that $P_{\mathbf{a}} + R = W$. We notice that $V - R_{\mathbf{a}} = R$, as otherwise for any $x \in V - R_{\mathbf{a}} - R$ the reaction $\mathbf{a}'' + (x, \varnothing, x + x')$ would be enabled by $V + V'$ and would be strictly greater than $\mathbf{a}''$, in contradiction to the fact that $\mathbf{a}''$ is mr enabled. Clearly, $\mathbf{a} \in \langle \mathbf{A} \rangle$ is enabled by V. We now prove it is also mr enabled. Suppose for a contradiction that there exists $\mathbf{b} \in \langle \mathbf{A} \rangle$ such that $\mathbf{b} >_r \mathbf{a}$ and $\mathbf{b}$ is enabled by V, then $\mathbf{b}'' \coloneqq \mathbf{b}' + (V - R_{\mathbf{b}}, \varnothing, V - R_{\mathbf{b}} + V' - R'_{\mathbf{b}})$ would be enabled by $V + V'$ and $\mathbf{b}'' >_r \mathbf{a}''$, a contradiction. Finally we can conclude that $\mathbf{a} \in \mathrm{En}^{mr}_{\mathbf{A}}(V)$, hence $P_{\mathbf{a}} + V - R_{\mathbf{a}} = W \in \mathrm{Res}^{mr}_{\mathbf{A}}(V)$. ∎

Let us now make some considerations about the workings of $\mathcal{M}$. Consider any state V, let R be s.t. $(R, I, P) \in \mathbf{A}$ for some I and P and $(R \cup R') \subseteq V + V'$; then, for any $x \in R$, the corresponding reaction $(x, \varnothing, x + x') \in \mathbf{A}_p$ cannot be mr enabled, as it is smaller than $(R + R', I', P + P') \in \mathbf{A}_c$. Thus a reaction $(x, \varnothing, x + x') \in \mathbf{A}_p$ will only be applied in two cases: either (i), the multiplicity of x in V is greater than that of x', thus there is at least one "spare" x that will not be used by any reaction from $\mathbf{A}_c$; or (ii), there is no $(R \cup R') \subseteq V + V'$ such that $(R, I, P) \in \mathbf{A}$. Note that case (i) can only happen in the initial state D_0, in which there are no elements from S', or when x is added to a state because it is a letter of an input word: this is because every state other than D_0 is the product of some reaction in $\langle \mathbf{A}' \rangle$, thus, by definition, it is of the form $P + P'$ for some $P \in S^{\#}$. Case (ii) ensures that if x is not consumed by any reaction from $\langle \mathbf{A_c} \rangle$, then it is conserved in the next state by a reaction from $\mathbf{A}_p$, simulating what would happen in the RA $\mathcal{A}$ when a reactant is not consumed by any reaction from $\mathbf{A}$.

Consider now an input word $w = w_1 \cdots w_n \in \Sigma^*$. We notice that when a letter w_i of w is added to the $(i-1)$-th state of a process in $\mathcal{M}$, it immediately reacts through the corresponding reaction $(w_i, \varnothing, w_i + w'_i) \in \mathbf{A}_p$ because it occurs case (i) above. This helps us to find the desired correspondence between processes of $\mathcal{A}$ and processes of $\mathcal{M}$ accepting w. Let $\pi = D_0, D_1, \ldots, D_i, \ldots \in \mathrm{IP}_{mr}(\mathcal{A}, w)$: the corresponding process $E_0, E_1 \ldots, E_n \in \mathrm{IP}_{mr}(\mathcal{M}, w)$ is obtained as follows. $E_1 \coloneqq D_0 +$

$D'_0 + w_1 + w'_1 \in \widehat{\mathrm{Res}}^{mr}_{\mathbf{A}'}(D_0 + w_1)$ since the only reactions that are enabled are those in $\langle \mathbf{A}_p \rangle$. In the next steps we have, by Claim 9 and the observations here above:

$$\begin{cases} E_{k+1} = D_k + D'_k + w_{k+1} + w'_{k+1} \in \widehat{\mathrm{Res}}^{mr}_{\mathbf{A}'}(E_k + w_{k+1}) & \forall k = 1, \dots, n-1 \\ E_{k+1} = D_k + D'_k \in \widehat{\mathrm{Res}}^{mr}_{\mathbf{A}'}(E_k) & \forall k \geqslant n. \end{cases}$$

Hence the processes of $\mathcal{M}$ mimic those of $\mathcal{A}$ with one step of delay, i.e. $\pi' \coloneqq D_0, E_1, \dots, E_i, \dots \in \mathrm{IP}_{mr}(\mathcal{M}, w)$. If $\pi \in \mathrm{AIP}^w_{mr}(\mathcal{A}, w)$ then there exists $m \geqslant |w|$ such that $D_k = D_m$ for all $k \geqslant m$ and $f \in D_m$; this holds true if and only if $E_k = E_{m+1}$ for all $k \geqslant m+1$ and $f \in E_{m+1}$. Therefore $\pi \in \mathrm{AIP}^w_{mr}(\mathcal{A}, w)$ if and only if $\pi' \in \mathrm{AIP}^w_{mr}(\mathcal{M}, w)$. Since the accepting condition is the same, and a state maintaining f in $\mathcal{A}$ must correspond to a state maintaining f in $\mathcal{M}$, we have that $\mathrm{AIP}^w_{mr}(\mathcal{A}, w)$ is in a natural bijection with $\mathrm{AIP}^w_{mr}(\mathcal{M}, w)$, and hence we obtain the thesis. □

We next prove in Theorem 6.2.2 that for any RA working in mp manner, there exists a PRA working in mr manner that weakly accepts the same languages. Example 6.2.3 shows that this result cannot be achieved using the same construction as in the proof of Theorem 6.2.1, thus we will provide a more involved reduction, an example of which is laid out in Example 6.2.4.

Example 6.2.3. Given $\Sigma = \{w_1, w_2\}$, $S = \{w_1, w_2, \heartsuit\}$ and $\mathbf{A} = \{\mathbf{a}_1 = (w_1, \varnothing, \heartsuit), \mathbf{a}_2 = (w_1 + w_2, \varnothing, w_2)\}$, let $\mathcal{A} = (S, \Sigma, \mathbf{A}, w_1, \heartsuit)$ be a RA working in mp manner. Consider the PRA $\mathcal{M} = (S \cup S', \Sigma, \mathbf{A}', w_1, \heartsuit)$ as in the proof of Theorem 6.2.1, thus with $\mathbf{A}' = \{\mathbf{a}'_1, \mathbf{a}'_2\} \cup \{\mathbf{a}_x = (x, \varnothing, x + x') \mid x \in S\}$. Then w_2 belongs to the language weakly accepted by $\mathcal{A}$ in a maximally parallel manner, i.e. $w_2 \in \mathrm{L}^w_{mp}(\mathcal{A})$ (see Definition 6.1.5), but it does not belong to the language accepted by $\mathcal{A}$ in a maximally reactive manner, i.e. $w_2 \notin \mathrm{L}^w_{mr}(\mathcal{M})$.

Indeed, as seen in Example 6.1.1, the set of reactions from $\mathcal{A}$ enabled in state $T = w_1 + w_2$ in a maximally parallel manner is $\mathrm{En}^{mp}_{\mathbf{A}}(w_1 + w_2) = \{\mathbf{a}_1, \mathbf{a}_2\}$, thus the result is $\mathrm{Res}^{mp}_{\mathbf{A}}(w_1 + w_2) = \{\heartsuit + w_2, w_2\}$ and w_2 is accepted by the interactive process $D_0 + w_2 = w_1 + w_2 \xrightarrow{\mathbf{a}_1} \heartsuit + w_2 \to \heartsuit + w_2 \to \cdots \in \mathrm{AIP}^w(w_2, \mathcal{A})$.

In contrast, since the set of reactions enabled in $T = w_1 + w_2$ in a maximally recative manner is $\mathrm{En}^{mr}_{\mathbf{A}}(w_1 + w_2) = \{\mathbf{a}_2\}$ (see Example 6.1.1), then the result is $\mathrm{Res}^{mr}_{\mathbf{A}}(w_1 + w_2) = \{w_2\}$, thus by Claim 9 we obtain that the set of reactions from $\mathcal{M}$ enabled in $T + T' = w_1 + w'_1 + w_2 + w'_2$ in a maximally reactive manner is $\mathrm{En}^{mr}_{\mathbf{A}'}(w_1 + w'_1 + w_2 + w'_2) = \{\mathbf{a}'_2\}$ and the pure result is $\widehat{\mathrm{Res}}^{mr}_{\mathbf{A}}(w_1 + w'_1 + w_2 + w'_2) = \{w_2 + w'_2\}$. Therefore there is only one process with input w_2, namely: $D_0 + w_2 = w_1 + w_2 \xrightarrow{\mathbf{a}_{w_1} + \mathbf{a}_{w_2}} w_1 + w'_1 + w_2 + w'_2 \xrightarrow{\mathbf{a}'_2} w_2 + w'_2 \xrightarrow{\mathbf{a}_{w_2}} w_2 + w'_2 \xrightarrow{\mathbf{a}_{w_2}} \cdots$; since $\heartsuit \notin w_1 + w'_2$, this is not an accepting process, thus w_2 is not in the language weakly accepted by $\mathcal{M}$ in a maximally reactive manner, i.e. $w_2 \notin \mathrm{L}^w_{mr}(\mathcal{M})$.

Example 6.2.4. Consider $\mathcal{A} = (S, \Sigma, \mathbf{A}, w_1, \heartsuit)$ the RA from Example 6.2.3. We make two copies of each element of the background set, the i-th copy being in a natural

correspondence with reaction $\mathbf{a}_i$: $S^i = \{w_1^i, w_2^i, \heartsuit^i\}$ for $i \in \{1,2\}$. Let $\mathbf{A}'$ consist of the following reactions over $S \cup S^1 \cup S^2$:

$$\begin{aligned}
\mathbf{a}_1' &\coloneqq (w_1 + w_1^1, \varnothing, \heartsuit + \heartsuit^1 + \heartsuit^2) \\
\mathbf{a}_2' &\coloneqq (w_1 + w_2 + w_1^2 + w_2^2, \varnothing, w_2 + w_2^1 + w_2^2) \\
\mathbf{a}_x &\coloneqq (x, \varnothing, x + x^1 + x^2) \qquad \text{for all } x \in S.
\end{aligned}$$

We define the PRA $\mathcal{M}' = (S \cup S^1 \cup S^2, \Sigma, \mathbf{A}', w_1, \heartsuit)$. We have that $\mathrm{En}_{\mathbf{A}'}^{\mathrm{mr}}(w_1 + w_1^1 + w_1^2 + w_2 + w_2^1 + w_2^2) = \{\mathbf{a}_2', \mathbf{a}_1' + \mathbf{a}_{w_2}\}$ since $\mathbf{a}_2'$ and $\mathbf{a}_1' + \mathbf{a}_{w_2}$ are not comparable and are both maximal. Therefore we obtain the following accepting process with input w_2:

$$\begin{aligned}
D_0 + w_2 = w_1 + w_2 &\xrightarrow{\mathbf{a}_{w_1} + \mathbf{a}_{w_2}} w_1 + w_1^1 + w_1^2 + w_2 + w_2^1 + w_2^2 \\
&\xrightarrow{\mathbf{a}_1' + \mathbf{a}_{w_2}} \heartsuit + \heartsuit^1 + \heartsuit^2 + w_2 + w_2^1 + w_2^2 \\
&\xrightarrow{\mathbf{a}_\heartsuit + \mathbf{a}_{w_2}} \heartsuit + \heartsuit^1 + \heartsuit^2 + w_2 + w_2^1 + w_2^2 \xrightarrow{\mathbf{a}_\heartsuit + \mathbf{a}_{w_2}} \cdots .
\end{aligned}$$

We have obtained that $w_2 \in L_{\mathrm{mr}}^{\mathrm{w}}(\mathcal{M}')$. In the proof of the following theorem, we will extend this construction.

Theorem 6.2.2. *Given a reaction automaton $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, f)$ working in a maximally parallel manner, there exists a pure reaction automaton $\mathcal{M}$ working in a maximally reactive manner such that $L_{\mathrm{mp}}^{\mathrm{w}}(\mathcal{A}) = L_{\mathrm{mr}}^{\mathrm{w}}(\mathcal{M})$.*

Proof. Let $\mathbf{A} = \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$. We make $k = |\mathbf{A}|$ copies of each element of the background set, the i-th copy being in a natural correspondence with reaction $\mathbf{a}_i$: $S^i = \{x^i \mid x \in S\}$ for each $i = 1, 2, \ldots, |\mathbf{A}|$. We define a PRA $\mathcal{M} = (S \cup S', \Sigma, \mathbf{A}', D_0, f)$ working in a maximally reactive manner such that:

- $S' \coloneqq S^1 \cup \cdots \cup S^k$;
- Σ is the same input alphabet as $\mathcal{A}$;
- $\mathbf{A}' \coloneqq \mathbf{A}_p \cup \mathbf{A}_c$, where

$$\begin{aligned}
\mathbf{A}_p &\coloneqq \{(x, \varnothing, x + x^1 + x^2 + \cdots + x^k) \mid x \in S\} \\
\mathbf{A}_c &\coloneqq \{\mathbf{a}_i' \coloneqq (R_{\mathbf{a}_i} + R_{\mathbf{a}_i}^i, I_{\mathbf{a}_i}^i, P_{\mathbf{a}_i} + P_{\mathbf{a}_i}^1 + \cdots + P_{\mathbf{a}_i}^k) \mid \mathbf{a}_i = (R_{\mathbf{a}_i}, I_{\mathbf{a}_i}, P_{\mathbf{a}_i}) \in \mathbf{A}\};
\end{aligned}$$

- D_0 is the same initial state as $\mathcal{A}$;
- f is the same final element as $\mathcal{A}$.

Furthermore, for any $T \in S^{\#}$ we define $T' \coloneqq \sum_{j=1}^{k} T^j \in (S')^{\#}$, where T^j consists of the j-th copy of every element of T. In particular, x' denotes $x^1 + x^2 + \cdots + x^k$ for any $x \in S$.

Claim 10. $W \in \mathrm{Res}_{\mathbf{A}}^{\mathrm{mp}}(V)$ if and only if $W + W' \in \widehat{\mathrm{Res}}_{\mathbf{A}'}^{\mathrm{mr}}(V + V')$.

Proof. We prove the two implications.

⇒) If $W \in \mathrm{Res}_{\mathbf{A}}^{mp}(V)$ then $\exists \mathbf{a} = \sum_{j=1}^{k} \lambda_j \mathbf{a}_j \in \mathrm{En}_{\mathbf{A}}^{mp}(V)$ such that $P_{\mathbf{a}} + V - R_{\mathbf{a}} = W$. Consider $\mathbf{a}'' \coloneqq \sum_{j=1}^{k} \lambda_j \mathbf{a}'_j + (V - R_{\mathbf{a}}, \varnothing, V - R_{\mathbf{a}} + V' - R'_{\mathbf{a}}) \in \langle \mathbf{A}' \rangle$, then $V + V'$ enables $\mathbf{a}''$; we want to show that it is mr enabled. Let $\mathbf{b}'' \in \langle \mathbf{A}' \rangle$ such that $\mathbf{b}'' >_r \mathbf{a}''$ and suppose for a contradiction that $\mathbf{b}''$ is enabled by $V + V'$. We have that $\mathbf{b}'' = \sum_{j=1}^{k} \mu_j \mathbf{a}'_j + (R, \varnothing, R + R')$ for some $\mathbf{b} = \sum_{j=1}^{k} \mu_j \mathbf{a}_j \in \langle \mathbf{A} \rangle$ and some $R \in S^{\#}$, thus looking at the reactants we obtain $R = V - R_{\mathbf{b}}$. Furthermore, since $\mathbf{b}'' >_r \mathbf{a}''$, there exists i such that $\lambda_i < \mu_i$, thus $\mathbf{a} + (\mu_i - \lambda_i)\mathbf{a}_i \leqslant_r \mathbf{b}$ is enabled by V, a contradiction since $\mathbf{a}$ is enabled in maximally parallel manner. Finally, we can conclude that $\mathbf{a}'' \in \mathrm{En}_{\mathbf{A}'}^{mr}(V + V')$, hence $W + W' \in \widehat{\mathrm{Res}}_{\mathbf{A}'}^{mr}(V + V')$.

⇐) If $W + W' \in \widehat{\mathrm{Res}}_{\mathbf{A}'}^{mr}(V + V')$ then $\exists \mathbf{a}'' = \mathbf{a}' + (R, \varnothing, R + R') \in \mathrm{En}_{\mathbf{A}'}^{mr}(V + V')$, for some $\mathbf{a} \in \langle \mathbf{A} \rangle$ and $R \in S^{\#}$, such that $P_{\mathbf{a}} + R = W$. Note that $V - R_{\mathbf{a}} = R$, as otherwise, for any $x \in V - R_{\mathbf{a}} - R$, the reaction $\mathbf{a}'' + (x, \varnothing, x + x')$ would be enabled by $V + V'$ and strictly greater than $\mathbf{a}''$, in contradiction to the fact that $\mathbf{a}$ is maximally enabled. We immediately remark that $\mathbf{a} \in \langle \mathbf{A} \rangle$ is enabled by V. We want to prove that is mp enabled. Let $\mathbf{c} \in \langle \mathbf{A} \rangle$ such that $\mathbf{a} + \mathbf{c}$ is enabled by V, then $(\mathbf{a} + \mathbf{c})'' >_r \mathbf{a}''$ is enabled by $V + V'$, a contradiction. Finally we can conclude that $\mathbf{a} \in \mathrm{En}_{\mathbf{A}}^{mp}(V)$, hence $P_{\mathbf{a}} + V - R_{\mathbf{a}} = P_{\mathbf{a}} + R = W \in \mathrm{Res}_{\mathbf{A}}^{mp}(V)$. ■

We conclude as in Theorem 6.2.1. □

Corollary 6.2.3. *$\mathcal{RA}_{mr} \subseteq \mathcal{RA}^{w}_{mr} \subseteq \mathcal{PRA}^{w}_{mr}$ and $\mathcal{RA}_{mp} \subseteq \mathcal{RA}^{w}_{mp} \subseteq \mathcal{PRA}^{w}_{mr}$.*

Proof. Follows directly from Theorems 6.2.1 and 6.2.2 and Corollary 6.1.2. □

Corollary 6.2.4. *Every recursively enumerable language is weakly accepted by a pure reaction automaton working in a maximally reactive manner.*

Proof. Follows directly from Corollary 6.1.3 and Theorem 6.2.2. □

6.3 COMPUTING FUNCTIONS WITH PURE REACTION AUTOMATA

In this section, we introduce a new angle to investigate the computing power of PRA. Inspired by existing work on chemical reaction networks (see [72, 77]) we will show that PRA can be seen as machines to compute partial functions from $\mathbb{N}^k$ into $\mathbb{N}$. For more details on partial recursive functions and related computability issues, we refer the reader to [162].

Definition 6.3.1. Given a partial function $\phi : \mathbb{N}^n \rightharpoonup \mathbb{N}$, consider an alphabet $\Sigma \coloneqq \{a_1, \ldots, a_n\}$ so that $\Sigma^{\#} \cong \mathbb{N}^n$. A PRA $\mathcal{M} = (S, \Sigma, \mathbf{A}, D_0, f)$ *computes* ϕ in a manner $X \in \{mp, mr\}$ if:

- when $\phi(x_1, \ldots, x_n)$ is defined, it holds $\phi(x_1, \ldots, x_n) = y$ if and only if when the linear combination $x_1 a_1 + \cdots + x_n a_n$ is added to the initial state D_0 for any process $D_0 + x_1 a_1 + \cdots + x_n a_n, \ldots, D_k, \ldots$ there exists $k \in \mathbb{N}$ such that $D_k = D_m \forall m \geqslant k, yf \leqslant D_k, (y+1)f \not\leqslant D_k$;
- $\phi(x_1, \ldots, x_n)$ is undefined if and only if all processes starting from $D_0 + a_1 + \cdots + x_n a_n$ satisfy $\forall k \in \mathbb{N} : D_k \neq D_{k+1}$, i.e. none of the processes stabilizes.

In other words, a PRA computes a partial function ϕ if and only if, interpreting $(x_1, \ldots, x_n) \in \mathbb{N}^n$ as the coefficients of a linear combination of the elements in Σ and adding this combination to the initial state, one of the following happens: if ϕ is not defined for $(x_1, \ldots, x_n)$, no process that starts from there stabilizes; otherwise, any process stabilizes on a state that contains f with multiplicity y, where $\phi(x_1, \ldots, x_n) = y$.

We will focus on a special class of PRA, specified in Definition 6.3.2, such that there is exactly one process that can start from each possible initial state. Throughout this section, we assume that all PRA operate in a maximally reactive manner.

Definition 6.3.2. Given $\mathcal{M} = (S, \Sigma, \mathbf{A}, D_0, S_f)$ a PRA working in a manner $X \in \{mp, mr\}$, we say that $\mathcal{M}$ is *deterministic* (DPRA) if and only if for any reachable state V, the pure result $\widehat{\mathrm{Res}}^X_{\mathbf{A}}(V)$ consists of one element only.

The next lemmas provide a few examples of partial functions that can be computed with DPRA working in a maximally reactive manner.

Lemma 6.3.1. *There exists a DPRA computing the sum function* $\phi : \mathbb{N}^n \to \mathbb{N}$, $\phi(x_1, x_2) \coloneqq x_1 + x_2$.

Proof. Let $\Sigma \coloneqq \{a_1, a_2\}$, $S \coloneqq \{a_1, a_2, f\}$ and $\mathbf{A} \coloneqq \{\mathbf{b}_1 = (a_1, \varnothing, f)$, $\mathbf{b}_2 = (a_2, \varnothing, f)$, $\mathbf{b}_3 = (f, \varnothing, f)\}$. $\mathcal{M} \coloneqq (S, \Sigma, \mathbf{A}, 0, f)$ is clearly deterministic and it computes ϕ as, for any $(x_1, x_2) \in \mathbb{N}^2$, adding $x_1 a_1 + x_2 a_2$ to the initial state 0 gives rise to the process

$$x_1 a_1 + x_2 a_2 \xrightarrow{x_1 \mathbf{b}_1 + x_2 \mathbf{b}_2} (x_1 + x_2) f \xrightarrow{(x_1 + x_2)\mathbf{b}_3} (x_1 + x_2) f \longrightarrow \cdots .$$

□

Lemma 6.3.2. *There exists a DPRA computing the difference function* $\phi : \mathbb{N}^n \rightharpoonup \mathbb{N}$, $\phi(x_1, x_2) \coloneqq x_1 - x_2$.

Proof. Clearly, $\phi(x_1, x_2)$ is defined if and only if $x_1 \geqslant x_2$. Let $\Sigma \coloneqq \{a_1, a_2\}$, $S \coloneqq \{a_1, a_2, a_1', a_2', \diamondsuit, f\}$ and let $\mathbf{A}$ consist of the following six reactions:

$$\begin{aligned}
\mathbf{b}_1 &= (a_1, \varnothing, a_1') \\
\mathbf{b}_2 &= (a_2, \varnothing, a_2' + \diamondsuit) \\
\mathbf{b}_3 &= (a_2', \varnothing, a_2) \\
\mathbf{b}_4 &= (a_1', \varnothing, f) \\
\mathbf{b}_5 &= (a_1' + a_2' + \diamondsuit, \varnothing, 0) \\
\mathbf{b}_6 &= (f, \varnothing, f).
\end{aligned}$$

We claim that $\mathcal{M} \coloneqq (S, \Sigma, \mathbf{A}, 0, f)$ computes ϕ. Indeed, let $(x_1, x_2) \in \mathbb{N}^2$ such that $x_1 \geqslant x_2$, then adding $x_1 a_1 + x_2 a_2$ to the initial state 0 gives rise to the process

$$\begin{aligned}
x_1 a_1 + x_2 a_2 &\xrightarrow{x_1 \mathbf{b}_1 + x_2 \mathbf{b}_2} x_1 a_1' + x_2 a_2' + x_2 \diamondsuit \\
&\xrightarrow{(x_1 - x_2)\mathbf{b}_4 + x_2 \mathbf{b}_5} (x_1 - x_2) f \xrightarrow{(x_1 - x_2)\mathbf{b}_6} (x_1 - x_2) f \longrightarrow \cdots .
\end{aligned}$$

On the other hand, if $(x_1, x_2) \in \mathbb{N}^2$ is such that $x_1 < x_2$, then the process becomes

$$\begin{aligned} x_1 a_1 + x_2 a_2 &\xrightarrow{x_1 \mathbf{b}_1 + x_2 \mathbf{b}_2} x_1 a_1' + x_2 a_2' + x_2 \diamondsuit \\ &\xrightarrow{(x_2 - x_1)\mathbf{b}_3 + x_1 \mathbf{b}_5} (x_2 - x_1) a_2 \\ &\xrightarrow{(x_2 - x_1)\mathbf{b}_2} (x_2 - x_1) a_2' + (x_2 - x_1)\diamondsuit \\ &\xrightarrow{(x_2 - x_1)\mathbf{b}_3} (x_2 - x_1) a_2 \longrightarrow \cdots, \end{aligned}$$

thus the process gets stuck in an unstable configuration. Note that the proof relies heavily on maximal reactivity: $\mathbf{b}_5 >_r \mathbf{b}_3 + \mathbf{b}_4$ since $a_1' + a_2' + \diamondsuit > a_1' + a_2'$. The role of the element $\diamondsuit$ is precisely to make $\mathbf{b}_5$ maximal in this case. □

We will prove that the class of functions that DPRA operating in mr can compute is universal, in the sense of Church-Turing's Thesis. We start with the following basic lemma.

Lemma 6.3.3. *The constant function equal to 0, the successor function and the projection functions can be computed by a DPRA.*

Proof. Constant function. The constant function $C_0 : \mathbb{N}^k \to \mathbb{N}$, $C_0(x_1, \ldots, x_k) \coloneqq 0\ \forall (x_1, \ldots, x_k) \in \mathbb{N}^k$ is computed by the DPRA $\mathcal{M} = (\Sigma \cup \{f\}, \Sigma, \varnothing, 0, f)$. It is straightforward to verify that $x_1 a_1 + \cdots + x_k a_k$ gives rise to the process $x_1 a_1 + \cdots + x_k a_k \to 0 \to 0 \to \cdots$.

Successor function. The successor function $S : \mathbb{N} \to \mathbb{N}$, $S(x) \coloneqq x + 1\ \forall x \in \mathbb{N}$, is computed by the DPRA $\mathcal{M} = (S, \Sigma, \mathbf{A}, \triangleright, f)$, where $S = \Sigma \cup \{\triangleright, f\}$, $\Sigma = \{a\}$ and $\mathbf{A} = \{(\triangleright, \varnothing, f), (a, \varnothing, f), (f, \varnothing, f)\}$. It is straightforward to verify that $\triangleright + xa$ gives rise to the process $\triangleright + xa \to (x+1)f \to (x+1)f \to \cdots$.

Projection functions. Given $k, n \in \mathbb{N}$ with $1 \leqslant n \leqslant k$, the projection function $P_n^k : \mathbb{N}^k \to \mathbb{N}$, $P_n^k(x_1, \ldots, x_k) \coloneqq x_n\ \forall (x_1, \ldots, x_k) \in \mathbb{N}^k$, is computed by the DPRA $\mathcal{M} = (S, \Sigma, \mathbf{A}, 0, f)$ where $S = \Sigma \cup \{f\}$, $\Sigma = \{a_1, \ldots, a_k\}$ and $\mathbf{A} = \{(a_n, \varnothing, f), (f, \varnothing, f)\}$. It is straightforward to verify that $x_1 a_1 + \cdots + x_k a_k$ gives rise to the process $x_1 a_1 + \cdots + x_k a_k \to x_n f \to x_n f \to \cdots$. □

Note that so far we have not made use of inhibitors in the reactions; however, they will become crucial in proving the following results.

Remark 6.3.1. If a process within a DPRA reaches the state $D_k = 0$, then $D_{k+1} = 0$. Furthermore, by determinism, if $D_k = D_{k+1}$ then $D_{k+m} = D_k$ for all $m \in \mathbb{N}$, thus to decide whether the process stabilizes it suffices to find the smallest $k \in \mathbb{N}$ such that $D_k = D_{k+1}$.

Definition 6.3.3 (Normalized DPRA)**.** A DPRA $\mathcal{M} = (S, \Sigma, \mathbf{A}, D_0, f)$ operating in a maximally reactive manner which computes a partial function ϕ is called *normalized* if the following two conditions hold:

1. there exists $h \in S$ such that, whenever $\phi(x_1, \ldots, x_k) = y$, the process $D_0 + x_1 a_1 + \cdots + x_n a_n, \ldots, D_k, \ldots$ stabilizes in a state of the type $P + h + yf$, for some $P \in S^{\#}$;

2. h and f are not present in the multiset of reactants of any reaction in $\mathbf{A}$.

A normalized DPRA computing ϕ will be denoted by $\mathcal{M}(\phi, h, f)$.

The following lemma proves that to determine the computational power of DPRA as function acceptors, it suffices to study normalized DPRA.

Lemma 6.3.4. *Given any DPRA computing a partial function ϕ, there exists a normalized DPRA computing ϕ.*

Proof. Given $\mathcal{M} = (S, \Sigma, \mathbf{A}, D_0, f)$ computing ϕ, $\Sigma = \{a_1, \dots, a_n\}$, $S = \Sigma \cup \{s_1, \dots, s_m, f\}$, we define $\widehat{\mathcal{M}} = (\widehat{S}, \Sigma, \mathbf{A}', \triangleright + \diamondsuit + \spadesuit, g)$ as follows:

- $\widehat{S} \coloneqq S^0 \cup S^1 \cup S^2 \cup S^3 \cup \Sigma \cup \{\triangleright, g, g^1, h, \square, \diamondsuit, \spadesuit, \#^1, \#^3\}$, where $S^i \coloneqq \{s^i : s \in S\}$ for all $i \in \{0, 1, 2, 3\}$;
- Σ is the same input alphabet as $\mathcal{M}$;
- $\mathbf{A}' \coloneqq \mathbf{A}_{in} \cup \mathbf{A}^0 \cup \mathbf{A}_{\spadesuit} \cup \mathbf{A}_f$, where

$$\mathbf{A}_{in} \coloneqq \begin{cases} (a, \varnothing, a^0 + a^1 + \#^1) & \text{for each } a \in \Sigma, \\ (\triangleright, \varnothing, D_0^0 + D_0^1 + \|D_0\|\#^1 + \spadesuit) & \\ (\diamondsuit, \{\square\}, \diamondsuit) & \end{cases}$$

$$\mathbf{A}^0 \coloneqq \begin{cases} (R^0, I^0 \cup S^1 \cup \{\square\}, P^0 + P^1 + \|P^1\|\#^1) & \text{for each } (R, I, P) \in \mathbf{A} \\ (R^0, I^0 \cup S^2 \cup \{\square\}, P^0 + P^2) & \text{for each } (R, I, P) \in \mathbf{A} \end{cases}$$

$$\mathbf{A}_{\spadesuit} \coloneqq \begin{cases} (\#^1 + a^1, \{\square\}, \#^3 + a^3) & \text{for each } a \in S \\ (a^2, \{\square\}, \spadesuit) & \text{for each } a \in S \\ (a^3, \{\square\}, \spadesuit) & \text{for each } a \in S \\ (a^2 + a^3 + \#^3, \{\square\}, 0) & \text{for each } a \in S \end{cases}$$

$$\mathbf{A}_f \coloneqq \begin{cases} (\#^1 + f^1, \{\spadesuit, \square\}, g) & \\ (\diamondsuit, S^2 \cup \{\spadesuit, \square, \triangleright\}, \square) & \\ (g, \varnothing, g + g^1) & \\ (\square, \varnothing, \square + h) & \end{cases}$$

- $\triangleright + \diamondsuit + \spadesuit$ is the new initial state;
- g is the new final element.

Before analyzing the processes within $\mathcal{M}$, let us give an intuition about the role of each group of reactions and each symbol in $\widehat{S}$. The reactions of type $\mathbf{A}^0$ mimic the reactions of $\mathcal{M}$: at every step of the process, they produce two copies of the state that would have been reached by the process in $\mathcal{M}$. One of the copies always consists of elements from S^0, the second copy consists of elements from S^1 in the first step, from S^2 in the second step, and they keep alternating because the two groups of reactions in $\mathbf{A}^0$ are inhibited by elements from S^1 and S^2, respectively. We call the steps in which elements from S^1 are produced *of type 1*; the steps in which elements from S^2

are produced are *of type 2*. In steps of type 1, the reactions from the first group also produce the symbol $\#^1$ with a multiplicity equal to the number of elements from S^1 that are produced; $\#^1$ is thus a counter for the elements from S^1 in the next state.

The first two groups of reactions from $\mathbf{A}_{in}$ only happen at the beginning of the computation (indeed $\triangleright$ symbolizes starting the computation), their role being to produce two copies, consisting of elements from S^0 and S^1, respectively, of all the input elements and a copy of the initial state of $\mathcal{M}$, so as to allow the reactions from $\mathbf{A}^0$ to take place; they also produce the symbol $\spadesuit$, which must be produced as long as the computation has not reached a stationary state. The last reaction produces the symbol $\diamondsuit$, whose role is explained later, at every step, until there is a signal for the computation to stop.

The reactions in the first group of $\mathbf{A}_\spadesuit$ have the role of transforming every element from S^1 in the current state into its copy from S^3 in the next state, producing also a counter $\#^3$ that will have the same multiplicity as $\#^1$. The rest of the reactions are used to compare the elements from S^2 with those from S^3 in the current state: the symbol $\spadesuit$ is produced as long as they differ, and it is no longer produced as soon as they are equal. This comparing mechanism heavily relies on maximal reactivity, because it always holds $(a^2 + a^3 + \#^3, \{\square\}, 0) >_r (a^2, \{\square\}, \spadesuit) + (a^3, \{\square\}, \spadesuit)$ for any $a \in S$: this implies that if the elements from S^2 are a copy of those from S^3, none of the reactions from the second and third group of $\mathbf{A}_\spadesuit$ will be maximally enabled (note that $\#^3$ will be present in the exact same quantity as the multiplicity of the elements from S^3) thus $\spadesuit$ will not be produced. The role of the $\#^3$ counters is really important since they give a way to prioritize the last group of reactions in $\mathbf{A}_\spadesuit$.

The reactions from $\mathbf{A}_f$ are needed to produce the last states at the end of the computation. The first two take place as soon as $\spadesuit$ is no longer present in a state, thus when the computation needs to stop. If f was generated by the last reaction when the process stops in the original automaton $\mathcal{M}$, then f^1 is present as well (together with the right multiplicity of the counter $\#^1$), thus the first reaction from $\mathbf{A}_f$ takes place: indeed, it is always greater than the first reaction from $\mathbf{A}_\spadesuit$ (because its set of inhibitors also contains $\spadesuit$), thus it is enabled in a maximal reactive manner. The second reaction from $\mathbf{A}_f$ is maximally enabled as well (it is strictly greater than the last reaction from $\mathbf{A}_{in}$) thus it produces $\square$, while $\diamondsuit$ is no longer produced. The role of $\square$ is to disable all the reactions from $\mathbf{A}^0$; the role of $\diamondsuit$ is to enable the reaction that produces $\square$ whenever needed. Finally, the last two reactions from $\mathbf{A}_f$ are enabled, and they produce h and y copies of $g + g^1$.

Let us now analyze the processes within $\widehat{\mathcal{M}}$. Given $\pi = D_0 + x_1 a_1 + \cdots + x_n a_n, \ldots, D_k, \ldots$ a process in $\mathcal{M}$, we denote $X_0 := D_0 + x_1 a_1 + \cdots + x_n a_n$. We obtain the following process for $\widehat{\mathcal{M}}$:

$$\begin{aligned} \triangleright + \spadesuit + \diamondsuit + x_1 a_1 + \cdots + x_n a_n &\longrightarrow X_0^0 + X_0^1 + \|X_0\|\#^1 + \spadesuit + \diamondsuit \\ &\longrightarrow D_1^0 + D_1^2 + X_0^3 + \|X_0\|\#^3 + \diamondsuit \\ &\longrightarrow D_2^0 + D_2^1 + \|D_2\|\#^1 + \|X_0 \vartriangle D_1\|\spadesuit + \diamondsuit \\ &\longrightarrow \cdots . \end{aligned}$$

Suppose π stabilizes, then by Remark 6.3.1 there exists a k such that $D_k = D_{k+1}$ but $D_{k-1} \neq D_k$. We first consider the case in which $D_k \neq 0$.

If k is odd, we have:

$$\begin{aligned}
&D_k^0 + D_k^2 + D_{k-1}^3 + \|D_{k-1}\|\#^3 + \diamondsuit \longrightarrow \\
&\qquad \longrightarrow D_k^0 + D_k^1 + \|D_k\|\#^1 + \|D_k \vartriangle D_{k-1}\|\spadesuit + \diamondsuit \\
&\qquad \longrightarrow D_k^0 + D_k^2 + D_k^3 + \|D_k\|\#^3 + \diamondsuit \\
&\qquad \longrightarrow D_k^0 + D_k^1 + \|D_k\|\#^1 + \underbrace{\|D_k \vartriangle D_k\|}_{=0}\spadesuit + \diamondsuit \\
&\qquad \longrightarrow \square + D_k^0 + D_k^2 + D_k^3 - yf^3 + \|D_k - yf\|\#^3 + yg + yg^1 + \diamondsuit \\
&\qquad \longrightarrow \square + h + yg + yg^1 \longrightarrow \square + h + yg + yg^1 \longrightarrow \cdots .
\end{aligned}$$

If k is even, we obtain the same process as above, except it starts from the second row. Note that if $D_k = 0$, thus $D_{k-1} \neq 0$, then the computation in $\mathcal{M}$ stops returning the value $y = 0$. We divide again two cases:

- if k is odd then $D_{k-1}^3 + \|D_{k-1}\|\#^3 + \diamondsuit \to \|D_{k-1}\|\spadesuit + \square \to \square + h \to \square + h \to \cdots$;
- if k is even then $\|D_{k-1} \vartriangle D_{k-2}\|\spadesuit + \diamondsuit \to \diamondsuit \to \square \to \square + h \to \square + h \to \cdots$.

Recall that Condition 1 of Definition 6.3.3 requires the state on which the process stabilizes to be of the form $P + h + yf$. In all the previous cases, this condition is satisfied with $P = \square + yg \in \widehat{S}^{\#}$ and $f = g^1$.

If, instead, the process π in $\mathcal{M}$ does not stabilize (and in particular, we have that $D_k \neq 0$ for all k), then $\spadesuit$ will never disappear, thus $\square$ will never be generated, and thus the corresponding process in $\widehat{\mathcal{M}}$ will never stabilize as well. □

Building on Lemma 6.3.4, we can prove that DPRA compute a much larger class of partial functions.

Lemma 6.3.5. *Given* $\phi : \mathbb{N}^k \rightharpoonup \mathbb{N}$ *and* $\psi_i : \mathbb{N}^n \rightharpoonup \mathbb{N}$ *for all* $i \in \{1, \dots, k\}$*, all computable by DPRA, the composition function* $\kappa : \mathbb{N}^n \rightharpoonup \mathbb{N}$ *defined by*

$$\kappa(x_1, \dots, x_n) \coloneqq \phi(\psi_1(x_1, \dots, x_n), \dots, \psi_k(x_1, \dots, x_n))$$

is also computable by a DPRA.

Proof. Let $\Sigma = \{a_1, \dots, a_n\}$ and $\Sigma^N = \{b_1^N, \dots, b_k^N\}$ be two disjoint sets such that $\Sigma^{\#} \cong \mathbb{N}^n$ and $(\Sigma^N)^{\#} \cong \mathbb{N}^k$; we additionally define $\Sigma^i = \{a_j^i : a_j \in \Sigma\}$ for all $i \in \{1, \dots, k\}$. We consider $\mathcal{N} \coloneqq \mathcal{N}(\phi, h^N, f^N) = (S^N, \Sigma^N, \mathbf{A}^N, D_0^N, f^N)$ the normalized DPRA computing ϕ and $\mathcal{M}^i \coloneqq \mathcal{M}(\psi_i, h^i, f^i) = (S^i, \Sigma^i, \mathbf{A}^i, D_0^i, f^i)$ the normalized DPRA computing ψ_i for all $i = 1, \dots, k$; furthermore, we assume that the respective background sets S^N, $S^1, \dots, S^k$ are disjoint, and we will label the corresponding reactions and elements with the same apex. We denote $\mathcal{K} \coloneqq (S, \Sigma, \mathbf{A}, D_0, f^N)$ the DPRA defined by:

- $S \coloneqq S^N \cup S^1 \cup \cdots \cup S^k \cup \Sigma \cup \{\rhd^M, \rhd^N, \square^M, \square^N\}$, where $\rhd^M, \rhd^N, \square^M, \square^N$ are new elements;
- $\Sigma \coloneqq \{a_1, \dots, a_n\}$ is the input alphabet;

- $\mathbf{A} \coloneqq \mathbf{A}_{in} \cup \mathbf{A}_c \cup \mathbf{A}_\square^N \cup \mathbf{A}_\square^1 \cup \cdots \cup \mathbf{A}_\square^k$, where

$$\mathbf{A}_{in} \coloneqq \begin{cases} (\triangleright^M, \varnothing, D_0^1 + \cdots + D_0^k), \\ (\triangleright^N, \varnothing, D_0^N), \\ (\square^N, \varnothing, \square^N) \\ (a_i, \varnothing, a_i^1 + \cdots + a_i^k) \qquad \text{for each } i \in \{1, \ldots, n\} \end{cases}$$

$$\mathbf{A}_c \coloneqq \begin{cases} (h^1 + \cdots + h^k + \square^N, \{\square^M\}, \triangleright^N + \square^M) \\ (f^i, \square^N, b_i^N) \qquad \text{for each } i \in \{1, \ldots, k\} \end{cases}$$

$$\mathbf{A}_\square^N \coloneqq \left\{ (R_a^N, I_a^N \cup \{\square^N\}, P_a^N) \quad \text{for each } (R_a^N, I_a^N, P_a^N) \in \mathbf{A}^N \right.$$

$$\mathbf{A}_\square^i \coloneqq \left\{ (R_a^i, I_a^i \cup \{\square^M\}, P_a^i) \quad \text{for each } (R_a^i, I_a^i, P_a^i) \in \mathbf{A}^i \right\} \quad \forall i \in \{1, \ldots, k\};$$

- $D_0 \coloneqq \triangleright^M + \square^N$ is the initial state;
- f^N is the same final element as $\mathcal{N}$.

We want to analyse the process starting with $\triangleright^M + \square^N + x_1 a_1 + \cdots + x_n a_n$. At the first step, only the reactions in $\mathbf{A}_{in}$ are enabled; thus k copies of each of the input elements are generated (one for each of the alphabets Σ^i), as well as the initial states $D_0^1, \ldots, D_0^k$ of $\mathcal{M}^1, \ldots, \mathcal{M}^k$. The element $\square^N$ is preserved. Now each of the $\mathcal{M}^i$ is simulated by the reactions in $\mathbf{A}_\square^i$, and $\square^N$ is preserved by the third reaction of $\mathbf{A}_{in}$ until the reaction $(h^1 + \cdots + h^k + \square^N, \{\square^M\}, \triangleright^N + \square^M) >_r (\square^N, \varnothing, \square^N)$ from $\mathbf{A}_c$ becomes enabled, i.e. when the computation of all $\mathcal{M}^i$ terminates and produces all the elements h^i (note that here it is crucial to operate in a mr manner). Recall that by Condition 2 from Definition 6.3.3, none of the elements h^i are present in any reactant multiset, thus no combination of the reactions from $\mathbf{A}_\square^i$ can be greater than the first reaction from $\mathbf{A}_c$. When this reaction takes place, $\square^N$ is replaced by $\triangleright^N + \square^M$, thus the reactions $(\triangleright^N, \varnothing, D_0^N)$ and $(f^i, \square^N, b_i^N)$ produce the initial state and the input for $\mathcal{N}$, while $\square^M$ will block any reactions in $\mathbf{A}_\square^1, \ldots, \mathbf{A}_\square^k$.

Finally, in the successive computation, there will be no element of $S^1 \cup \cdots \cup S^k$ left, and $\mathcal{N}$ will start its computing process using, as desired, the outputs of the other k machines as inputs. □

Lemma 6.3.6. *Given* $\phi : \mathbb{N}^{k+2} \rightharpoonup \mathbb{N}$ *and* $\psi : \mathbb{N}^k \rightharpoonup \mathbb{N}$, *both computable by DPRA, the primitive recursion* $\rho : \mathbb{N}^{k+1} \rightharpoonup \mathbb{N}$ *defined by*

$$\rho(0, x_1, \ldots, x_k) \coloneqq \psi(x_1, \ldots, x_k)$$

and

$$\rho(y+1, x_1, \ldots, x_k) \coloneqq \phi(y, \rho(y, x_1, \ldots, x_k), x_1, \ldots, x_k) \quad \forall y \geqslant 1$$

is also computable by a DPRA.

Proof. Let $\Sigma_k = \{a_1, \ldots, a_k\}$ such that $\Sigma_k^\# \cong \mathbb{N}^k$ and let $a, a_0 \notin \Sigma_k$ be two extra symbols. We define two additional alphabets $\Sigma^M \coloneqq \{a_1^M, \ldots, a_k^M\}$ and $\Sigma^N = \{a^N, a_0^N, a_1^N, \ldots, a_k^N\}$, so that $(\Sigma^N)^\# \cong \mathbb{N}^{k+2}$.

We consider $\mathcal{N} \coloneqq \mathcal{N}(\phi, h^N, f) = (S^N, \Sigma^N, \mathbf{A}^N, D_0^N, f)$ the normalized DPRA computing ϕ and $\mathcal{M} \coloneqq \mathcal{M}(\psi, h^M, f) = (S^M, \Sigma^M, \mathbf{A}^M, D_0^M, f)$ the normalized DPRA computing ψ; furthermore, we assume that the respective background sets S^N, S^M are disjoint except for the final element, i.e. $S^N \cap S^M = \{f\}$; we will label the corresponding reactions and elements with the same apex. We define $\mathcal{R} \coloneqq (S, \Sigma, \mathbf{A}, D_0, f')$ the DPRA such that:

- $S \coloneqq S^N \cup S^M \cup \Sigma \cup \Sigma'_k \cup \{\triangleright, \triangleright^M, \triangleright^N, \square^M, \square^N, \square^M_{in}, \square^N_{in}, \circlearrowleft^N, \#, \#', f'\}$, where $\Sigma'_k = \{a'_1, \ldots, a'_k\}$;
- $\Sigma \coloneqq \{a, a_1, \ldots, a_k\}$ is the input alphabet;
- $\mathbf{A} \coloneqq \mathbf{A}_{in} \cup \mathbf{A}_m \cup \mathbf{A}_c \cup \mathbf{A}^N_\square \cup \mathbf{A}^M_\square$, where

$$\mathbf{A}_{in} \coloneqq \begin{cases} (\triangleright, \varnothing, \triangleright^M) \\ (\triangleright^M, \{\square^M, \square^M_{in}\}, D_0^M + \square^M_{in}) \\ (\triangleright^N, \{\square^N, \square^N_{in}\}, D_0^N + \square^N_{in}) \\ (f', \{\square^N, \square^N_{in}\}, a_0^N) \\ (\#', \{\square^N, \square^N_{in}\}, a^N) \\ (a'_i, \{\square^M, \square^M_{in}\}, a_i^M) & \text{for each } i \in \{1, \ldots, k\} \\ (a'_i, \{\square^N, \square^N_{in}\}, a_i^N) & \text{for each } i \in \{1, \ldots, k\} \end{cases}$$

$$\mathbf{A}_m \coloneqq \begin{cases} (a_i, \varnothing, a_i + a'_i) \quad \text{for each } i \in \{1, \ldots, k\} \\ (a, \varnothing, a) \\ (\square^N, \varnothing, \square^N) \\ (\square^M, \varnothing, \square^M) \\ (\square^N_{in}, \varnothing, \square^N_{in}) \\ (\square^M_{in}, \varnothing, \square^M_{in}) \\ (\#, \varnothing, \# + \#') \\ (f, \varnothing, f') \end{cases}$$

$$\mathbf{A}_c \coloneqq \begin{cases} (h^M + a, \varnothing, \circlearrowleft^N) \\ (h^N + a, \varnothing, \circlearrowleft^N + \square^N) \\ (\circlearrowleft^N + \square^N, \varnothing, \square^M + \triangleright^N) \\ (\circlearrowleft^N + \square^N_{in} + \square^N, \varnothing, \# + \#' + \triangleright^N) \end{cases}$$

$$\mathbf{A}^N_\square \coloneqq \{(R^N_a, I^N_a \cup \{\square^N\}, P^N_a) : (R^N_a, I^N_a, P^N_a) \in \mathbf{A}^N\}$$
$$\mathbf{A}^M_\square \coloneqq \{(R^M_a, I^M_a \cup \{\square^M\}, P^M_a) : (R^M_a, I^M_a, P^M_a) \in \mathbf{A}^M\};$$

- $D_0 \coloneqq \triangleright + \square^N$ is the new initial state;
- f' is the new final element.

First, we note that the reactions in $\mathbf{A}_m$ are inhibitorless. Thus, they will occur, maintaining the respective elements and generating a copy of them using the alphabet Σ'_k, whenever their reactants are present, as they do not conflict with any other reaction. The elements from Σ'_k will be later translated into the corresponding elements from Σ^M or Σ^N, depending on the phase of the process, by the last two reaction groups from $\mathbf{A}_{in}$.

We now explain how the machine works. In the initial state $\triangleright + \square^N + ya + x_1a_1 + \cdots + x_ka_k$, the reaction

$$(\triangleright, \varnothing, \triangleright^M) + y(a, \varnothing, a) + \sum_{i=1}^{k} x_i(a_i, \varnothing, a_i + a'_i)$$

is maximally enabled; in particular, it generates $\triangleright^M$, which in the next step enables the generation of the initial state of $\mathcal{M}$. Moreover, the element $\square^M_{in}$ is generated and later preserved via the reaction $(\square^M_{in}, \varnothing, \square^M_{in}) \in \mathbf{A}_m$, preventing the initial state of $\mathcal{M}$ from being generated multiple times. The process that would take place in $\mathcal{M}$ is then simulated with the reactions from $\mathbf{A}^M_\square$; whenever $\mathcal{M}$ terminates, it produces a single element h^M. We remark that ya is preserved in any state via $(a, \varnothing, a) \in \mathbf{A}_m$, then we have to consider two cases.

$y = 0$. No reaction from $\mathbf{A}_c$ nor $\mathbf{A}^N_\square$ is enabled , so the system reaches a stable state and the result is the same as the result given by $\mathcal{M}$ with input $(x_1, \ldots, x_k)$.

$y \geqslant 1$. The reaction $(h^M + a, \varnothing, \circlearrowleft^N)$ is enabled (being strictly greater than one instance of $(a, \varnothing, a)$, thus reducing the multiplicity of a, which is initially equal to y, by 1); in the next step, the reaction $(\circlearrowleft^N + \square^N, \varnothing, \square^M + \triangleright^N)$ will take place, thus eliminating $\square^N$ and introducing $\square^M$, whose role is to block all reactions from $\mathbf{A}^M_\square$ (simulating $\mathcal{M}$) and to enable the reactions from $\mathbf{A}^N_\square$ (simulating $\mathcal{N}$). At this point, the initial state of $\mathcal{N}$ is generated by reactions in $\mathbf{A}_{in}$, receiving as input $(0, \psi(x_1, \ldots, x_k), x_1, \ldots, x_k)$, as the reaction $(f', \{\square_N, \square^N_{in}\}, a^N_0)$ transfers the output of $\mathcal{M}$ to the second coordinate of ϕ, and since # is not present, and consequently #′ neither, the first coordinate is 0. The process will continue until $\mathcal{N}$ eventually produces an output. If a is no longer present ($y - 1 = 0$), the output will be $\phi(0, \psi(x_1, \ldots, x_k), x_1, \ldots, x_k)$, as desired; if a is still present ($y > 1$), the reaction $(h^N + a, \varnothing, \circlearrowleft^N + \square^N)$ will take place, and as a consequence, the whole simulation of $\mathcal{N}$ will be reinitialized by $\square^N$. The only reactions that will take place are those from $\mathbf{A}_m$ maintaining and copying the input, the reaction $(f, \varnothing, f')$ saving the output of $\mathcal{N}$, and the reaction $(\circlearrowleft^N + \square^N_{in} + \square^N, \varnothing, \# + \#' + \triangleright^N)$ that will reinitialize $\mathcal{N}$ in the next step. Since it will receive the multiplicity of #′ as the first coordinate, the multiplicity of f' as the second one and then the rest of the saved input, the machine will then calculate $\phi(1, \phi(0, \psi(x_1, \ldots, x_k), x_1, \ldots, x_k), x_1, \ldots, x_k)$. This process will repeat until the occurrences of a are exhausted (recall that its multiplicity decreases by 1 at every iteration), so we have proved that $\mathcal{R}$ computes ρ, the primitive recursion operator of ϕ and ψ. □

Lemma 6.3.7. *Given* $\phi : \mathbb{N}^{k+1} \rightharpoonup \mathbb{N}$*, computable by a DPRA, the minimization operator* $\mu_\phi : \mathbb{N}^k \rightharpoonup \mathbb{N}$ *defined by*

$$\mu_\phi(x_1, \dots, x_k) \coloneqq \begin{cases} z & \text{if } \phi(z, x_1, \dots, x_k) = 0 \text{ and } \phi(y, x_1, \dots, x_k) > 0 \text{ for all } y < z \\ \perp & \text{otherwise} \end{cases}$$

where $\perp$ *means that the operator is undefined, is also computable by a DPRA.*

Proof. We consider, with the same notation as in the previous lemmas, $\mathcal{N} \coloneqq \mathcal{N}(\phi, h^N, f^N) = (S^N, \Sigma^N, \mathbf{A}^N, D_0^N, f^N)$ the normalized DPRA computing ϕ, with $\Sigma^N = \{a_0^N, a_1^N, \dots, a_k^N\}$. We define $\mathcal{M} \coloneqq (S, \Sigma, \mathbf{A}, D_0, f')$ the DPRA given by:

- $S \coloneqq S^N \cup \Sigma \cup \{\triangleright, \square^N, \square_{in}^N, f, f'\}$;
- $\Sigma \coloneqq \{a_1, \dots, a_k\}$ is the new input alphabet;
- $\mathbf{A} \coloneqq \mathbf{A}_{in} \cup \mathbf{A}_m \cup \mathbf{A}_c \cup \mathbf{A}_\square^N$, where

$$\mathbf{A}_{in} \coloneqq \begin{cases} (\triangleright, \{\square^N, \square_{in}^N\}, D_0^N + \square_{in}^N) \\ (f, \{\square^N, \square_{in}^N\}, a_0^N) \\ (a_i, \{\square^N, \square_{in}^N\}, a_i^N) & \text{for each } i \in \{1, \dots, k\} \end{cases}$$

$$\mathbf{A}_m \coloneqq \begin{cases} (a_i, \varnothing, a_i) & \text{for each } i \in \{1, \dots, k\} \\ (f', \varnothing, f + f') \\ (\square_{in}^N, \varnothing, \square_{in}^N) \end{cases}$$

$$\mathbf{A}_c \coloneqq \begin{cases} (h^N + f^N, \varnothing, \square^N) \\ (\square^N + \square_{in}^N, \varnothing, f + f' + \triangleright) \end{cases}$$

$$\mathbf{A}_\square^N \coloneqq \{(R_a^N, I_a^N \cup \{\square^N\}, P_a^N) : (R_a^N, I_a^N, P_a^N) \in \mathbf{A}^N\};$$

- $D_0 \coloneqq \triangleright$ is the new initial state;
- f' is the new final element.

Much like in previous proofs, the reactions from $\mathbf{A}_m$ maintain the input elements. We now analyze the process of $\mathcal{M}$ starting from the state $\triangleright + x_1 a_1 + \cdots + x_k a_k$. At the beginning of the computation, using the first reaction from $\mathbf{A}_{in}$, the initial state of $\mathcal{N}$ is generated, receiving an input with 0 as the first coordinate: indeed, the value of the first coordinate is given by the multiplicity of a_0^N, which at the first step is not produced because f is not present in the initial state. Then $\mathcal{N}$ is simulated with the reactions from $\mathbf{A}_\square^N$. If $\mathcal{N}$ terminates and produces h^N, there are two possible cases.

1. f^N is not present. Then, since neither f' is present, the state remains unchanged, and hence the result is 0.
2. f^N is present. This indicates that $\mathcal{N}$ has returned a non-zero value, so the reaction $(h^N + f^N, \varnothing, \square^N)$ takes place, inhibiting every reaction from $\mathbf{A}_\square^N$ and enabling $(\square^N + \square_{in}^N, \varnothing, f' + f + \triangleright)$ in the next step. Since the former reaction

is greater than $(\square_{in}^{N}, \varnothing, \square_{in}^{N})$, the element $\square_{in}^{N}$ is not generated and the reaction generating the initial state of $\mathcal{N}$ is enabled again, receiving as input $(1, x_1, \ldots, x_k)$, because now f is present with multiplicity 1, thus a_0^N is generated with multiplicity 1. This process is iterated until $\mathcal{N}$ eventually outputs 0: at each iteration, the multiplicity of f' is increased by 1 (as well as those of f, via the second reaction from $\mathbf{A}_m$, and of a_0^N via the second reaction from $\mathbf{A}_{in}$), thus when the process terminates, the result is precisely the multiplicity of f'.

We proved that $\mathcal{M}$ computes the minimization operator of ϕ. □

We have arrived at the main result of this section.

Theorem 6.3.8. *The class of partial functions computed by deterministic pure reaction automata operating in a maximally reactive manner coincides with the class of general recursive functions.*

Proof. By Lemma 6.3.3, deterministic pure reaction automata can compute the basic functions: namely, the constant function equal to 0, the successor function and the projection functions. Furthermore, the partial functions computed by deterministic pure automata are closed under composition (Lemma 6.3.5), primitive recursion (Lemma 6.3.6) and minimization (Lemma 6.3.7), therefore they coincide with the class of general recursive functions. □

6.4 CONCLUSIONS

In this chapter, we introduced and studied a new criterion, the *maximally reactive manner*, for selecting the reactions that take place in a computation step of a reaction automaton. We also defined a new variant of RA, the *Pure* Reaction Automata, where there is no permanence, mimicking (in this aspect) the behaviour of RSs. We studied the relation between pure and classical RA working in a maximally reactive manner, showing that the absence of permanence is not a strong limitation: for every reaction automaton working in a maximally parallel (or maximally reactive) manner recognizing a certain language, there always exists a PRA working in a maximally reactive manner recognizing the same language. When seen as devices for computing partial functions, *deterministic* PRA working in a maximally reactive manner are able to compute all general recursive functions.

An interesting direction for future research is to further investigate the relation between the different manners since the choice of one or the other can potentially change the computational power of a RA (pure or not). The choice of different manners could also give rise to interesting results for the chemical version of RA, in which the set of inhibitors is constrained to be empty, as we will show in Chapter 7. The role of determinism is also to be further explored since it can be another factor that impacts the computational power of RA, possibly also determining which kinds of functions can be computed under time and space (size of the multisets) constraints. RSs were also explored with additional extensions and restrictions, like adding a duration for reactions or forcing each reaction to have only one reactant or one inhibitor. Similar questions can be asked about RA: what is their effect on the computational power? Can we still obtain universality in all cases?

7

CHEMICAL PURE REACTION AUTOMATA

In this chapter, we continue the study presented in Chapter 6 and introduce a new class of Reaction Automata, called *Chemical Pure Reaction Automata* (CPRA). CPRA combine characteristics of Chemical Reaction Automata (CRA), introduced in [209], with those of the PRA. Just like PRA, CPRA lack permanence: result states consist solely of the reaction products, with any unconsumed reactants being discarded. We determine the computational power of the proposed model as a language acceptor, both in the deterministic and non-deterministic setting. Our results can be summarised as follows (see Figure 21).

1. Deterministic CPRA are not Turing complete (Theorem 7.2.2). To prove this result, we make an unusual and somewhat surprising use of Dickson's Lemma, a standard result in commutative algebra.
2. Non-deterministic CPRA are strictly more powerful than deterministic CPRA. In particular, the set of languages accepted by CPRA in the maximally parallel manner contains the set of languages accepted by the standard CRA of [209] working in the same manner (Theorem 7.3.1).

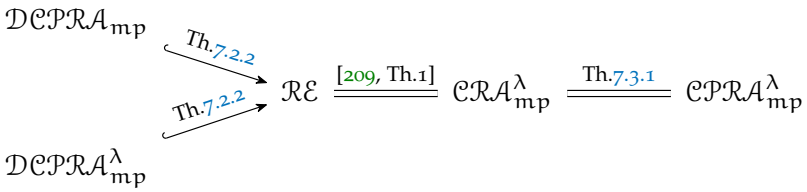


Figure 21. Graphical summary of the results of this chapter. $\mathcal{RE}$ represents the set of recursively enumerable languages. $\mathcal{DCPRA}_{mp}$, $\mathcal{CPRA}_{mp}$, $\mathcal{CRA}_{mp}$ are the set of languages recognized by DCPRA, CPRA, CRA working in maximally parallel (mp) manner. The apex w means weakly accepted (see Definition 7.1.8). The arrow $\hookrightarrow$ denotes language set strict inclusion.

Chapter organization. In Section 7.1, we introduce some preliminary definitions related to CRA and the definition of a specific topology over the states that will be used in the technical sections. In Section 7.2, we consider deterministic CPRA and prove that they are not Turing complete. In Section 7.3, we study non-deterministic CPRA and prove that they are strictly more powerful than deterministic CPRA. In Section 7.4, we state some open problems and draw conclusions over our results.

This chapter is based on the following publication: R. Ascone, G. Bernardini, F. Leiter, L. Manzoni. Chemical Pure Reaction Automata in Maximally Parallel Manner. In: *Journal of Membrane Computing*, 2025, [22].

7.1 PRELIMINARIES

7.1.1 *Topology on Multisets*

In this section, we define a topology on $S^\#$ and we prove that, under this topology, every subset of $S^\#$ is compact. These results will be useful in Section 7.2 to determine the computational power of deterministic chemical pure reaction automata.

Definition 7.1.1. Given $V \in S^\#$, we define $\mathcal{U}_V := \{W \in S^\# \mid V \leqslant W\}$, and $\mathfrak{T}$ the topology over $S^\#$ generated by the family $\mathfrak{B} := \{\mathcal{U}_V \mid V \in S^\#\}$.

Remark 7.1.1. The family $\mathfrak{B}$ is a base for the topology $\mathfrak{T}$. Indeed, $\mathfrak{B}$ covers $S^\#$ since $S^\# = \mathcal{U}_0$; and given any $\mathcal{U}_V, \mathcal{U}_W \in \mathfrak{B}$, the intersection $\mathcal{U}_V \cap \mathcal{U}_W$ is equal to $\mathcal{U}_T$ where $T(a) = \max\{V(a), W(a)\}$ for all $a \in S$. Moreover, note that $V \leqslant W$ implies $\mathcal{U}_V \supseteq \mathcal{U}_W$.

Given $S = \{x_1, \ldots, x_n\}$ a finite alphabet of n letters, let $K[S] = K[x_1, \ldots, x_n]$ be the ring of polynomials in n variables $x_1, \ldots, x_n$ over the field K and let $\Pi(S) = \Pi(x_1, \ldots, x_n)$ be the set of all monomials in $K[S]$.

Remark 7.1.2. The following map:

$$\begin{aligned} S^\# &\longrightarrow \Pi(S) \\ V &\longmapsto x_1^{V(x_1)} \cdot x_2^{V(x_2)} \cdot \ldots \cdot x_n^{V(x_n)} =: x^V \end{aligned}$$

is an isomorphisms of monoids. Furthermore, we have that $V \leqslant W$ if and only if x^V divides x^W.

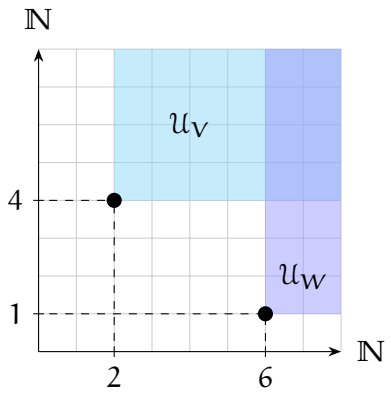


Figure 22. Visual representation, under the isomorphism of Remark 7.1.2, of the open sets corresponding to $V = 2a + 4b \mapsto (2, 4) \in \mathbb{N}^2$ and $W = 6a + b \mapsto (6, 1) \in \mathbb{N}^2$.

Definition 7.1.2 (monomial ideal)**.** An ideal $I \subseteq K[x_1, \ldots, x_n]$ is *monomial* if there exists $A \subseteq \mathbb{N}^n$ such that I is generated by the family of monomials $\{x^\alpha \mid \alpha \in A\}$, and it is denoted by $I = \langle x^\alpha : \alpha \in A \rangle$.

We next report, for completeness, two results by Cox et al. we rely upon. Lemma 7.1.1 can be found at p70 of [85] (therein it is called Lemma 2); Theorem 7.1.2 is Theorem 5 at p74 of [85].

Lemma 7.1.1 ([85]). *Let* $I = \langle x^\alpha : \alpha \in A \rangle$ *be a monomial ideal, then a monomial* x^β *lies in* I *if and only if* x^β *is divisible by* x^α *for some* $\alpha \in A$.

Theorem 7.1.2 (Dickson's lemma [85]). *Let* $I = \langle x^\alpha : \alpha \in A \rangle \subseteq K[x_1, \dots, x_n]$ *be a monomial ideal, then there exists* $\alpha_1, \dots, \alpha_k \in A$ *such that* $I = \langle x^{\alpha_1}, \dots, x^{\alpha_k} \rangle$.

As a corollary of Dickson's Lemma, we prove the following result.

Theorem 7.1.3. *Let* $X \subseteq S^\#$, *then for every open covering of* X *given by elements of the basis* $\mathfrak{B}$, *there exists a finite subcovering of* X.

Proof. Let $\mathfrak{U} = \{\mathcal{U}_V \mid V \in Y \subseteq S^\#\} \subseteq \mathfrak{B}$ be a covering of X. By Dickson's Lemma, the monomial ideal $I = \langle x^V : V \in Y \rangle$ equals $\langle x^{V_1}, \dots, x^{V_k} \rangle$ for some $V_1, \dots, V_k \in Y$. We claim that $\{\mathcal{U}_{V_1}, \dots, \mathcal{U}_{V_k}\}$ is a finite subcovering of X. Indeed, given $W \in X$ there exists $V \in Y$ such that $W \in \mathcal{U}_V$, thus $V \leqslant W$. As pointed out in Remark 7.1.2, $V \leqslant W$ if and only if x^V divides x^W. By Lemma 7.1.1, we have $x^W \in I = \langle x^{V_1}, \dots, x^{V_k} \rangle$, hence (again applying Lemma 7.1.1) we get that x^W is divisible by x^{V_i} for some $i = 1, \dots, k$. Therefore $W \geqslant V_i$, i.e. $W \in \mathcal{U}_{V_i}$. □

Corollary 7.1.4. *Every subset of* $S^\#$ *is compact for the topology* $\mathfrak{T}$.

Corollary 7.1.5. *Given a sequence of multisets* $\{V_n\}_{n \in \mathbb{N}} \subseteq S^\#$, *there exist* $N, M \in \mathbb{N}, N \geqslant M$, *such that* $V_N \geqslant V_M$.

Proof. The family of open sets $\{\mathcal{U}_{V_n}\}_{n \in \mathbb{N}}$ is a covering for $\{V_n\}_{n \in \mathbb{N}}$. Since by Corollary 7.1.4 every subset is compact, there exists a finite subcovering $\{\mathcal{U}_{V_{N_1}}, \dots, \mathcal{U}_{V_{N_k}}\}$ of $\{V_n\}_{n \in \mathbb{N}}$. Then, given any $N > \max_{i=1,\dots,k} N_i$, there exists $i \in \{1, \dots, k\}$ such that $V_N \in \mathcal{U}_{V_{N_i}}$, i.e. $V_N \geqslant V_{N_i}$. □

7.1.2 *Chemical Reaction Automata*

A chemical reaction is a reaction (as defined in Chapter 6) without inhibitors. To simplify notation we will just identify a reaction $\mathbf{a} = (R_\mathbf{a}, \varnothing, P_\mathbf{a})$ with the tuple $(R_\mathbf{a}, P_\mathbf{a})$.

Definition 7.1.3 (Chemical reaction)**.** Given an alphabet of reactants S, a *chemical reaction over* S is a pair $\mathbf{a} = (R_\mathbf{a}, P_\mathbf{a})$, where $R_\mathbf{a} \in S^\#$ is the multiset of *reactants* and $P_\mathbf{a} \in S^\#$ is the multiset of *products*. The set of all chemical reactions over S is denoted by $\mathrm{crac}(S)$.

All the definitions of Section 6.1.1 have a counterpart for chemical reactions. For the sake of rigour, we will restate the definitions that differ the most from those of Section 6.1.1.

Let $\mathbf{a} = (R_\mathbf{a}, P_\mathbf{a})$, $\mathbf{b} = (R_\mathbf{b}, P_\mathbf{b}) \in \mathrm{crac}(S)$. A partial order over all possible chemical reactions over S can be naturally defined as $\mathbf{a} \leqslant_r \mathbf{b}$ if and only if $R_\mathbf{a} \leqslant R_\mathbf{b}$. We also define the sum of the two chemical reactions as $\mathbf{a} + \mathbf{b} \coloneqq (R_\mathbf{a} + R_\mathbf{b}, P_\mathbf{a} + P_\mathbf{b})$. Given a finite set $\mathbf{A} \subseteq \mathrm{crac}(S)$, we denote by $\langle \mathbf{A} \rangle$ the Abelian semigroup generated by the elements of $\mathbf{A}$.

Definition 7.1.4. Given $\mathbf{a} = (R_\mathbf{a}, P_\mathbf{a}) \in \mathrm{crac}(S)$ and $T \in S^\#$, $\mathbf{a}$ is *enabled* in T if $R_\mathbf{a} \leqslant T$. Furthermore, given $\mathbf{A}$ a finite set of chemical reactions over S and $\mathbf{a} \in \langle \mathbf{A} \rangle$ enabled in T, then $\mathbf{a}$ is enabled in a *maximally parallel manner (mp)* if there exists no $\mathbf{c} \in \langle \mathbf{A} \rangle$ such that $\mathbf{a} + \mathbf{c}$ is enabled in T, i.e. $\mathbf{a}$ is maximal w.r.t. addition. $\mathrm{En}^{\mathrm{mp}}_\mathbf{A}(T)$ denotes the set of reactions from $\langle \mathbf{A} \rangle$ enabled in a state T in mp manner.

Definition 7.1.5 ([209])**.** The *result* of a set of chemical reactions $\mathbf{A}$ on a state T in mp manner is the set of states

$$\mathrm{Res}^{mp}_{\mathbf{A}}(T) = \{P_a + (T - R_a) \mid \mathbf{a} = (R_a, P_a) \in \mathrm{En}^{mp}_{\mathbf{A}}(T)\}.$$

In the case where $\mathrm{En}^{mp}_{\mathbf{A}}(T) = \varnothing$, $\mathrm{Res}^{mp}_{\mathbf{A}}(T)$ is undefined.

Definition 7.1.6 ([209])**.** A *chemical reaction automaton* (CRA) $\mathcal{A}$ is a tuple $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, S_f)$, where S is a finite set of reactants, called the *background set* of $\mathcal{A}$; $\Sigma \subseteq S$ is the *input alphabet* of $\mathcal{A}$; $\mathbf{A} \subseteq \mathrm{crac}(S)$ is a finite set of chemical reactions over S; $D_0 \in S^{\#}$ is the initial multiset; and $S_f \subseteq S^{\#}$ is a set of final multisets.

Definition 7.1.7. Consider a CRA $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, S_f)$ and a word $w = w_1 \cdots w_n \in \Sigma^*$. An *interactive process in $\mathcal{A}$ with input w in mp manner* is an infinite sequence $\pi = D_0, \ldots, D_i, \ldots$ where

$$\begin{cases} D_{i+1} \in \mathrm{Res}^{mp}_{\mathbf{A}}(w_{i+1} + D_i) & \text{for } 0 \leqslant i \leqslant n-1 \\ D_{i+1} \in \mathrm{Res}^{mp}_{\mathbf{A}}(D_i) & \text{for } i \geqslant n. \end{cases}$$

$\mathsf{IP}_{mp}(\mathcal{A}, w)$ denotes the set of all such interactive processes in $\mathcal{A}$ with input w. We say that a process π *accepts* w if there exists $m \geqslant n = |w|$ such that $D_m \in S_f$. By $\mathrm{AIP}_{mp}(\mathcal{A}, w)$ we denote the set of all processes $\pi \in \mathsf{IP}_{mp}(\mathcal{A}, w)$ such that π accepts w. The *language accepted* by $\mathcal{A}$ is defined as

$$L_{mp}(\mathcal{A}) = \{w \in \Sigma^* \mid \mathrm{AIP}_{mp}(\mathcal{A}, w) \neq \varnothing\}.$$

The set of languages accepted by CRA working in mp manner is denoted by $\mathcal{CRA}_{mp}$: a language $L \in \mathcal{CRA}_{mp}$ if and only if there exists a chemical reaction automaton working in mp manner that accepts L.

Remark that the acceptance condition for CRA and RA differs. In the former case, we must reach a given multiset from a chosen set of multisets of final states S_f; instead, in the latter case, we only need to produce a given element f.

The following definition extends Definition 7.1.7 to the case where the symbols of the input word may be interleaved with an arbitrary number of gaps, modelled as an extra symbol $\lambda \notin \Sigma$.

Definition 7.1.8. Let $\Sigma_\lambda = \Sigma \cup \{\lambda\}$ and $w = w_1 \cdots w_n$ an input word over Σ. An interactive process π is said to be in λ-*input mode* when it allows as input a sequence of characters from Σ_λ $b_1, \ldots, b_m$ such that $m \geqslant n$ and the symbols of w form a subsequence of $b_1, \ldots, b_m$ interleaved with occurrences of λ: more formally, $\exists\, 1 \leqslant i_1 < i_2 < \ldots < i_n \leqslant m$ such that $b_{i_j} = w_j \ \forall j = 1, \ldots, n$ and $b_i = \lambda \ \forall i \in [1, m] \setminus \{i_1, i_2, \ldots, i_n\}$. The notation $\mathsf{IP}^{\lambda}_{X}(\mathcal{A}, w)$, $\mathrm{AIP}^{\lambda}_{X}(\mathcal{A}, w)$, $L^{\lambda}_{X}(\mathcal{A})$ and $\mathcal{CRA}^{\lambda}_{X}$ naturally extends the corresponding notation of Definition 7.1.7 to λ-input mode.

An ordinary interactive process, where no λ-input is used, will be said to be *real-time* to distinguish from λ-input mode processes.

Okubo et al. [209] proved that the computational power of chemical reaction automata working in mp manner in λ-input mode is equivalent to that of Turing machines: we report this result in Theorem 7.1.6 for completeness ($\mathcal{RE}$ denotes recursively enumerable languages).

Theorem 7.1.6. *[209, Theorem 1]* $\mathcal{CRA}^{\lambda}_{mp} = \mathcal{RE}$.

7.2 DETERMINISTIC CHEMICAL PURE REACTION AUTOMATA

In Definition 6.2.1 of the previous chapter, we introduced a different notion of result, where the reactants that are not consumed by the reactions are lost. An analogous definition holds in the chemical reaction setting.

Definition 7.2.1 (Pure result)**.** The *pure result* of a finite set of chemical reactions $\mathbf{A}$ on a state T in mp manner is

$$\widehat{\mathrm{Res}}_{\mathbf{A}}^{mp}(T) = \{P_a \mid \mathbf{a} = (R_a, P_a) \in \mathrm{En}_{\mathbf{A}}^{mp}(T)\},$$

and if $\mathrm{En}_{\mathbf{A}}^{mp}(T) = \varnothing$, $\widehat{\mathrm{Res}}_{\mathbf{A}}^{mp}(T)$ is undefined.

Example 7.2.1 (Pure result)**.** Let $\mathbf{A} = \{\mathbf{r}_1 = (a, b), \mathbf{r}_2 = (b + a, a)\} \in \mathrm{crac}(\{a, b\})$ and $T = 2b + a$, then $\mathrm{En}_{\mathbf{A}}^{mp}(T) = \{\mathbf{r}_1, \mathbf{r}_2\}$, thus $\widehat{\mathrm{Res}}_{\mathbf{A}}^{mp}(T) = \{b, a\}$. In contrast, $\mathrm{Res}_{\mathbf{A}}^{mp}(T) = \{3b, b + a\}$.

We name this new kind of reaction automata *Chemical Pure Reaction Automata* (CPRA). We define interactive processes in CPRA in much the same way as standard CRA as specified by Definition 7.2.2.

Definition 7.2.2. Let $\mathcal{M} = (S, \Sigma, \mathbf{A}, D_0, S_f)$ be a CPRA, $w = w_1 \cdots w_n \in \Sigma^*$. An interactive process in $\mathcal{M}$ with input w in mp manner is an infinite sequence $\pi = D_0, \ldots, D_i, \ldots$ where

$$\begin{cases} D_{i+1} \in \widehat{\mathrm{Res}}_{\mathbf{A}}^{mp}(w_{i+1} + D_i) & \text{for } 0 \leqslant i \leqslant n-1 \\ D_{i+1} \in \widehat{\mathrm{Res}}_{\mathbf{A}}^{mp}(D_i) & \text{for } i \geqslant n. \end{cases}$$

Exactly as for CRA, we say that π accepts w if there exists $m \geqslant n = |w|$ such that $D_m \in S_f$. We also define $\mathrm{IP}_{mp}(\mathcal{M}, w)$, $\mathrm{AIP}_{mp}(\mathcal{M}, w)$, and $\mathrm{L}_{mp}(\mathcal{M})$ in the same way as for CRA. The set of languages accepted by CPRA working in mp manner is denoted by $\mathcal{CPRA}_{mp}$.

The definition of λ-input mode process (Definition 7.1.8) holds also in the pure case. In particular, given a CPRA $\mathcal{M}$ and w input word, we will use the following notations $\mathrm{IP}_{mp}^{\lambda}(\mathcal{M}, w)$, $\mathrm{AIP}_{mp}^{\lambda}(\mathcal{M}, w)$, $\mathrm{L}_{mp}^{\lambda}(\mathcal{M})$. The set of languages accepted by CPRA working in mp manner with λ-input mode is denoted by $\mathcal{CPRA}_{mp}^{\lambda}$. We will sometimes represent an interactive process π with the following "arrow notation", as in the previous chapter:

$$\pi : D_0 \xrightarrow[w_1]{\mathbf{a}_1} D_1 \xrightarrow[w_2]{\mathbf{a}_2} D_2 \xrightarrow[w_3]{\mathbf{a}_3} \cdots D_{n-1} \xrightarrow[w_n]{\mathbf{a}_n} D_n \xrightarrow{\mathbf{a}_{n+1}} D_{n+1} \xrightarrow{\mathbf{a}_{n+1}} \cdots$$

where $D_{i-1} \xrightarrow[w_i]{\mathbf{a}_i} D_i$ means w_i is the input letter at state D_{i-1}, $\mathbf{a}_i \in \langle \mathbf{A} \rangle$ is the reaction enabled in $D_{i-1} + w_i$ which takes place, and $D_i \in \widehat{\mathrm{Res}}_{\mathbf{A}}^{mp}(w_i + D_{i-1})$.

The restricted class of *deterministic* CPRA, formalized in Definition 7.2.3, requires that the pure result of any reachable state consists of one state only.

Definition 7.2.3. Given $\mathcal{M} = (S, \Sigma, \mathbf{A}, D_0, S_f)$ a CPRA working in mp manner, we say that $\mathcal{M}$ is *deterministic* (DCPRA) if and only if for any reachable state V, the pure result $\widehat{\mathrm{Res}}_{\mathbf{A}}^{mp}(V)$ consists of one element only.

The set of languages accepted by DCPRA working in mp manner is denoted by $\mathcal{DCPRA}_{mp}$, while $\mathcal{DCPRA}^{\lambda}_{mp}$ denotes the languages accepted with λ-input mode. We remark that the notion of determinism given by Definition 7.2.3 differs from the notion given by Okubo et al. [210] for CRA, as their definition is too restrictive for the objects of our study. Note that any CPRA with the reactions defined in Example 7.2.1 is not deterministic. In contrast, Example 7.2.2 showcases a simple deterministic CPRA according to Definition 7.2.3.

Example 7.2.2. Given a background set $S = \{a, b, s_0, s_1, f, \clubsuit\}$, an input alphabet $\Sigma = \{a, b\}$, and a set of chemical reactions $\mathbf{A} = \{\mathbf{r}_1 = (s_0 + a, s_1), \mathbf{r}_2 = (s_0 + b, \clubsuit), \mathbf{r}_3 = (s_1 + b, s_0), \mathbf{r}_4 = (s_1 + a, \clubsuit), \mathbf{r}_5 = (\clubsuit, \clubsuit), \mathbf{r}_6 = (f, f)\}$, let $\mathcal{A} = (S, \{a, b\}, \mathbf{A}, s_0 + f, \{f\})$ be a CPRA working in mp manner. It is possible to show that $\mathcal{A}$ is deterministic and that the language accepted by $\mathcal{A}$ is $L_{mp}(\mathcal{A}) = \{(ab)^n \mid n \geqslant 0\}$. Indeed, if we consider the input word $w = abab \in \Sigma^*$, we get the accepting process:

$$s_0 + f \xrightarrow[a]{\mathbf{r}_1+\mathbf{r}_6} s_1 + f \xrightarrow[b]{\mathbf{r}_3+\mathbf{r}_6} s_0 + f \xrightarrow[a]{\mathbf{r}_1+\mathbf{r}_6} s_1 + f \xrightarrow[b]{\mathbf{r}_3+\mathbf{r}_6} s_0 + f \xrightarrow{\mathbf{a}_6} f.$$

Instead, if we choose $w = abba \in \Sigma^*$, we get the process:

$$s_0 + f \xrightarrow[a]{\mathbf{r}_1+\mathbf{r}_6} s_1 + f \xrightarrow[b]{\mathbf{r}_3+\mathbf{r}_6} s_0 + f \xrightarrow[b]{\mathbf{r}_2+\mathbf{r}_6} \clubsuit + f \xrightarrow[a]{\mathbf{r}_5+\mathbf{r}_6} \clubsuit + f \xrightarrow{\mathbf{r}_5+\mathbf{r}_6} \clubsuit + f \xrightarrow{\mathbf{r}_5+\mathbf{r}_6} \cdots$$

which clearly does not accept $abba$ since $f + \clubsuit$ is not a final state.

In this section, we investigate the computational power of DCPRA and prove that this class is not Turing complete. Observe that RSs can be seen as a simpler kind of PRA in which the multiplicities of reactants are not considered. We can thus think of inhibitorless RSs (Definition 2.1.1) as a simpler version of the CPRA. A consequence of [181, Prop. 20] is that the result function for inhibitorless RSs is monotone (Table 3). In Lemma 7.2.1, we extend this result for CPRA working in mp manner.

Lemma 7.2.1. *Let* $\mathbf{A}$ *be a set of chemical reactions over* $S^{\#}$, $D, D' \in S^{\#}$ *be two multisets such that* $D \leqslant D'$, $\widehat{\mathrm{Res}}^{mp}_{\mathbf{A}}(D) = \{P\}$, $\widehat{\mathrm{Res}}^{mp}_{\mathbf{A}}(D') = \{P'\}$, *then* $P \leqslant P'$.

Proof. Let $\mathbf{a} = (R_{\mathbf{a}}, P) \in \mathrm{En}^{mp}_{\mathbf{A}}(D)$. Since $D' \geqslant D$, $\mathbf{a}$ is also enabled by D'. Since the reactions happen according to the mp manner (see Definition 7.1.4) there exists $\mathbf{a}' = (R_{\mathbf{a}'}, P') \in \mathrm{En}^{mp}_{\mathbf{A}}(D')$ and $\mathbf{c} \in \langle \mathbf{A} \rangle \cup \{(0, 0)\}$ such that $\mathbf{a}' = \mathbf{a} + \mathbf{c}$. We conclude that $P' = P + P_{\mathbf{c}}$, thus $P' \geqslant P$. □

Remark 7.2.1. Lemma 7.2.1 does not hold if we apply the standard notion of result (Definition 7.1.5) instead of pure result (Definition 7.2.1). Indeed, let $\mathcal{A} = \{(a, a), (3b, b)\}$ be two chemical reactions over $S = \{a, b\}$ and consider states $D' = a + 3b$ and $D = a + 2b$. Then $\mathrm{Res}^{mp}(D) = \{a + 2b\} = \{D\}$ and $\mathrm{Res}^{mp}(D') = \{a + b\}$, thus $a + 2b > a + b$ despite being $D < D'$.

Theorem 7.2.2. *Given any deterministic pure chemical reaction automaton* $\mathcal{A}$, *it is always possible to determine in a finite number of steps, for any finite input, whether* $\mathcal{A}$ *will halt.*

Proof. Let $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, S_f)$ be a DCPRA working in mp manner and $\pi = D_0, D_1, \ldots, D_i, \ldots \in \mathrm{IP}_{mp}(\mathcal{A}, w)$ a process, where $w = w_1 \cdots w_n \in \Sigma^*$ is the input word. Since there is a finite number of final states, we describe an algorithm that

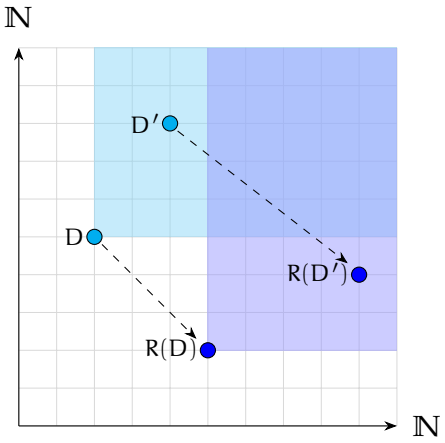


Figure 23. Visual representation of Lemma 7.2.1.

works for a single, fixed final state F, and then notice that it can be applied separately for each of the final states of $\mathcal{A}$ to obtain the statement. Consider the map

$$\begin{aligned} R : S^{\#} &\longrightarrow S^{\#} \\ T &\longmapsto R(T) \in \widehat{\mathrm{Res}}^{mp}_{A}(T). \end{aligned}$$

R is well defined on the elements of π because $\mathcal{A}$ is deterministic; in particular, it holds that $R(D_k) = D_{k+1}$ for all $k \geqslant n = |w|$. By Lemma 7.2.1, if R is well defined for two multisets $D \leqslant D'$, then $R(D) \leqslant R(D')$ (see Figure 23).

Let us now describe the halting algorithm. Given the sequence $D_n, D_{n+1}, \dots$, by Corollary 7.1.5 we can always find two integers $N \geqslant M$ such that $D_N \geqslant D_M$. Thus $R(D_N) \geqslant R(D_M)$, which translates to $D_{N+1} \geqslant D_{M+1}$. By induction, $D_{N+k} \geqslant D_{M+k}$ until k reaches $N' = N - M$, where we get $D_{N+N'} \geqslant D_{M+N'} = D_N \geqslant D_M$. This implies that any state D_k reached by the process later than D_N must belong to $\mathcal{U}_{D_{M+d}}$ for some $d \in [0, N']$: more precisely,

$$D_k \geqslant D_{M+((k-N) \mod N')} \quad \forall k \geqslant N, \quad \text{i.e.} \quad D_k \in \mathcal{U}_{D_{M+((k-N) \mod N')}}. \tag{119}$$

We call $D_k \leqslant D_{k+N'} \leqslant \dots$ the *chain* obtained from D_k as per Equation 119. The existence of such chains guarantees that the algorithm only needs to inspect a finite number of states to decide whether the DCPRA halts. Indeed, there are the following two cases. (1) If $F \notin \mathcal{U}_{D_k}$ (i.e. $F \not\geqslant D_k$) for any $k \in \{M, \dots, N-1\}$, then $\pi \notin \mathrm{AIP}_{mp}(\mathcal{A}, w)$ and we can immediately halt the algorithm. Otherwise, (2) there exists $k \in \{M, \dots, N-1\}$ such that $F \in \mathcal{U}_{D_k}$ (i.e. $F \geqslant D_k$): let $I = \{M_1, \dots, M_j\} \subseteq \{M, \dots, N-1\}$ be the subset of such indices, i.e. $F \geqslant D_k$ for all $k \in I$. Consider now $I_1 = \{M_1 + N', \dots, M_j + N'\}$ the set of indices of the successors in the chains starting from D_k, for all $k \in I$. If $F = D_h$ for some $h \in I_1 \cup I$, then $\pi \in \mathrm{AIP}_{mp}(\mathcal{A}, w)$, thus we can halt the algorithm. Therefore, assume $F > D_h$ for all $h \in I_1$. There are two subcases.

(i), for some index $M_k + N' \in I_1$, we find $D_{M_k+N'} = D_{M_k}$: this indicates that the process entered in a loop, thus $\pi \notin \mathrm{AIP}_{mp}(\mathcal{A}, w)$ and we can halt the algorithm. (ii), $D_{M_k+N'} > D_{M_k}$ for every $M_k \in I_1$. In this case, we iterate the procedure described for case (2) on I_2 (the successors of the chains corresponding to the indices of I_1), that

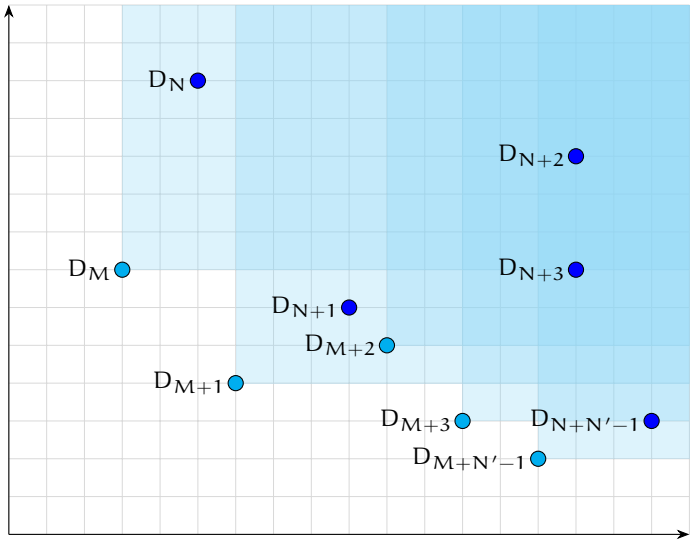


Figure 24. Example of multisets $\{D_M, D_{M+1}, \ldots, D_{M+N'-1}\}$ whose corresponding open sets bound the dynamics of the automaton.

is, we check if F is equal to any of those elements or it is found in any of the relative $\mathcal{U}$'s, we verify we have not entered a closed loop and then we proceed considering I_3. Remark that at every iteration we get closer to F, since we have strictly greater multisets at each step, and thus the maximum number of possible iterations is less than $\|F - D_{M_k}\|_1$ for all $M_k \in I_1$.

Since the DCPRA has only a finite number of final states, by applying this procedure separately for every possible final state we obtain the statement. □

Observe that Theorem 7.2.2 implies that the halting problem associated with the class of DCPRA working in mp is decidable. We deduce that this class of automata cannot be Turing complete, or equivalently, the set of languages accepted by DCPRA cannot coincide with the class of all recursively enumerable languages. We have arrived at the main result of this section.

Theorem 7.2.3. *$\mathcal{DCPRA}_{mp} \subsetneq \mathcal{RE}$ and $\mathcal{DCPRA}^{\lambda}_{mp} \subsetneq \mathcal{RE}$, i.e. the class of* DCPRA *working in* mp *manner is not Turing complete.*

7.3 NON-DETERMINISTIC CHEMICAL PURE REACTION AUTOMATA

In this section, we study the computational power of non-deterministic CPRA working in mp manner. Our main result, given in Theorem 7.3.1, is that the computational power of *pure* chemical reaction automata working in mp manner is the same as standard CRA working in the same manner, implying that CPRA are strictly more powerful than the *deterministic* CPRA studied in Section 7.2. We begin with an example of non-deterministic computations in CPRA.

Example 7.3.1. Given a background set $S = \{a, b, s_0, s_1, a'\}$, an input alphabet $\Sigma = \{a, b\}$, and a set of chemical reactions $\mathbf{A} = \{\mathbf{r}_1 = (s_0 + a, s_0 + a'), \mathbf{r}_2 = (s_0 + a' + b, s_1), \mathbf{r}_3 = (s_1 + a' + b, s_1), \mathbf{r}_4 = (a', a')\}$, let $\mathcal{A} = (S, \{a, b\}, \mathbf{A}, s_0, \{s_1\})$ be a CPRA working in mp. It is possible to show that $\mathcal{A}$ accepts the context-free language $\{a^n b^n \mid$

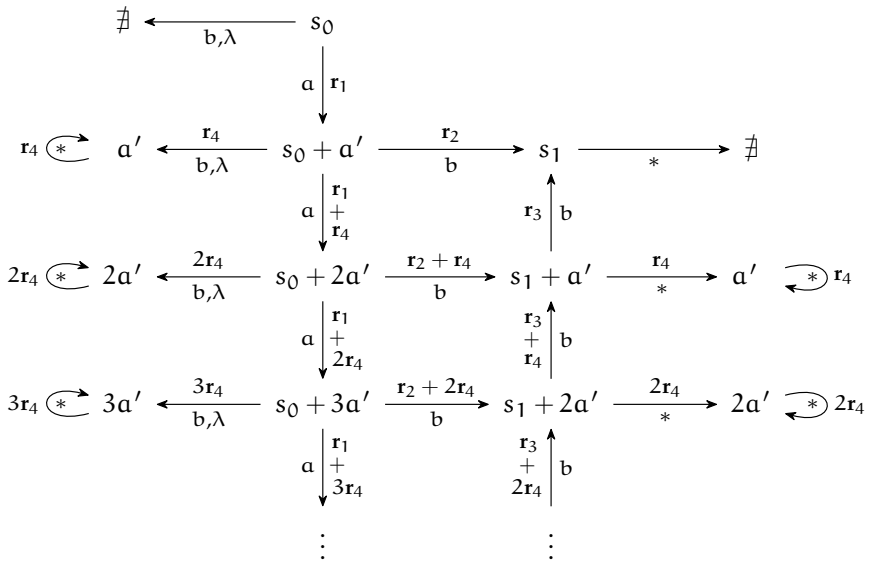


Figure 25. Schematic representation of all possible processes of the CPRA of Example 7.3.1, which accepts the language $\{a^n b^n \mid n > 0\}$. The notation $T \xrightarrow[*]{\mathbf{r}} T'$ means that, regardless of whether the input symbol is a, b or λ, the reaction enabled by T is $\mathbf{r}$.

$n > 0\}$, both in real-time and in λ-input mode, i.e. $L^{\lambda}_{mp}(\mathcal{A}) = L_{mp}(\mathcal{A}) = \{a^n b^n \mid n > 0\}$. Indeed, consider e.g. the input word $w = aabb \in \Sigma^*$. We obtain the following accepting process:

$$s_0 \xrightarrow[a]{\mathbf{r}_1} s_0 + a' \xrightarrow[a]{\mathbf{r}_1} s_0 + 2a' \xrightarrow[b]{\mathbf{r}_2+\mathbf{r}_4} s_1 + a' \xrightarrow[b]{\mathbf{r}_2} s_1.$$

Note that the non-determinism of the CPRA is relevant e.g. the third step, in which $T = s_0 + 2a'$ receives in input the symbol b: we have that $\mathrm{En}^{mp}_{\mathbf{A}}(T + b) = \{\mathbf{r}_2 + \mathbf{r}_4, 2\mathbf{r}_4\}$ thus $\widehat{\mathrm{Res}}^{mp}_{\mathbf{A}}(T + b) = \{2a', s_1 + a'\}$. Furthermore, it can be easily verified that any λ symbol added to $aabb$ would force a non-accepting process: e.g. inserting a gap after the first a would make the process loop over the state $T' = a'$, as $\mathrm{En}^{mp}_{\mathbf{A}}(s_0 + a') = \{\mathbf{r}_4\} = \mathrm{En}^{mp}_{\mathbf{A}}(a')$.

Consider now $w = aab \notin \{a^n b^n \mid n > 0\}$. It is easy to verify that no process reaches the final state s_1, e.g. w corresponds to the following two processes with real-time input:

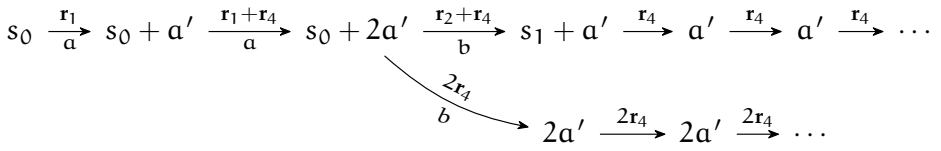


Figure 25 summarizes all possible processes in $\mathcal{A}$ for every possible input.

Using a structure similar to the proof of Theorem 6.2.1, we can prove the following results.

Theorem 7.3.1. *Given any chemical reaction automaton $\mathcal{A} = (S, \Sigma, \mathbf{A}, D_0, S_f)$ working in mp manner, there exists a chemical pure reaction automaton $\mathcal{M}$ working in mp manner such that $L^{\lambda}_{mp}(\mathcal{A}) = L^{\lambda}_{mp}(\mathcal{M})$.*

Proof. Let $\mathcal{M} = (S \cup S' \cup \{\spadesuit\}, \Sigma, \mathbf{A}', D_0, S'_f)$ be a CPRA operating in a maximally parallel manner such that:

- $S' = \{x' \mid x \in S\}$ is a set in a bijection with the elements of S (a "copy" of S). From now on, given a multiset X over S, X' will be naturally defined as the multiset consisting of the copies of the elements of X;
- Σ is the same input alphabet as $\mathcal{A}$;
- $\mathbf{A}' = \mathbf{A}_p \cup \mathbf{A}_c \cup \mathbf{A}_r \cup \{(\spadesuit, \spadesuit)\}$, where $\mathbf{A}_p = \{(x, x + x') \mid x \in S\}$, $\mathbf{A}_c = \{(R + R', P + P') \mid (R, P) \in \mathbf{A}\}$ and $\mathbf{A}_r = \{(R', \spadesuit) \mid (R, P) \in \mathbf{A}\}$;
- D_0 is the same initial state as $\mathcal{A}$;
- $S'_f = \{D + D' \mid D \in S_f\}$ is the new set of final states.

Remark that by reaction $(\spadesuit, \spadesuit)$, we know that $\spadesuit$ is preserved whenever generated, therefore, in order to reach a final state in S'_f, $\spadesuit$ should never be generated, i.e. reactions from $\langle \mathbf{A}_r \rangle$ should never be enabled. We formalize this idea in the following claim.

Claim 11. For any state W of $\mathcal{A}$, it holds $W \in \mathrm{Res}^{mp}_{\mathbf{A}}(V)$ if and only if $W + W' \in \widehat{\mathrm{Res}}^{mp}_{\mathbf{A}'}(V + V')$.

Proof. Given a reaction $\mathbf{a} \in \langle \mathbf{A} \rangle$, we denote the corresponding reaction in $\langle \mathbf{A}_c \rangle$ by $\mathbf{a}'$ and the corresponding reaction in $\langle \mathbf{A}_r \rangle$ by $\mathbf{a}_\spadesuit$. We prove the two implications.

$\Rightarrow$) If $W \in \mathrm{Res}^{mp}_{\mathbf{A}}(V)$ then $\exists \mathbf{a} = (R_\mathbf{a}, P_\mathbf{a}) \in \mathrm{En}^{mp}_{\mathbf{A}}(V)$ such that $P_\mathbf{a} + V - R_\mathbf{a} = W$. Consider $\mathbf{a}'' \coloneqq \mathbf{a}' + (V - R_\mathbf{a}, V - R_\mathbf{a} + V' - R'_\mathbf{a}) = (V + R'_\mathbf{a}, W + W') \in \langle \mathbf{A}' \rangle$. Clearly, $V + V'$ enables $\mathbf{a}''$; we want to show that it is mp enabled. First, we remark that $\mathbf{a}'' + \mathbf{c}$ is not enabled by $V + V'$ for any $\mathbf{c} \in \langle \mathbf{A}_p \cup \mathbf{A}_c \cup \{(\spadesuit, \spadesuit)\} \rangle$. If $\mathbf{a}'' + \mathbf{b}_\spadesuit$ is enabled by $V + V'$, then $R'_\mathbf{a} + R'_\mathbf{b} \leqslant V'$, thus $\mathbf{a} + \mathbf{b}$ is enabled by V, a contraddiction since $\mathbf{a}$ is mp enabled. We conclude that $\mathbf{a}'' \in \mathrm{En}^{mp}_{\mathbf{A}'}(V + V')$, hence $W + W' \in \widehat{\mathrm{Res}}^{mp}_{\mathbf{A}'}(V + V')$.

$\Leftarrow$) If $W + W' \in \widehat{\mathrm{Res}}^{mp}_{\mathbf{A}'}(V + V')$ then $\exists \mathbf{a}'' = \mathbf{a}' + (R, R + R') \in \mathrm{En}^{mp}_{\mathbf{A}'}(V + V')$, for some $\mathbf{a} \in \langle \mathbf{A} \rangle$ and $R \in S^\#$, such that $P_{\mathbf{a}''} = W + W'$. Since $\spadesuit \notin W + W'$, there can be no reaction from $\langle \mathbf{A}_r \rangle$ in the decomposition of $\mathbf{a}''$ as a sum of reactions from $\langle \mathbf{A}_p \rangle$, $\langle \mathbf{A}_c \rangle$, and $\langle \mathbf{A}_r \rangle$. We notice that $V - R_\mathbf{a} = R$, as otherwise for any $x \in V - R_\mathbf{a} - R$ the reaction $\mathbf{a}'' + (x, x + x')$ would be enabled by $V + V'$, in contradiction to the fact that $\mathbf{a}''$ is mp enabled. Remark that $\mathbf{a} \in \langle \mathbf{A} \rangle$ is enabled by V. We now prove it is also mp enabled. Suppose for a contradiction that there exists $\mathbf{b} \in \langle \mathbf{A} \rangle$ such that $\mathbf{a} + \mathbf{b}$ is enabled by V, then $\mathbf{a}'' + \mathbf{b}_\spadesuit$ would be enabled by $V + V'$, a contradiction since $\mathbf{a}''$ is mp enabled. Finally, we conclude that $\mathbf{a} \in \mathrm{En}^{mp}_{\mathbf{A}}(V)$, hence $P_\mathbf{a} + V - R_\mathbf{a} = W \in \mathrm{Res}^{mp}_{\mathbf{A}}(V)$. ∎

Claim 12. For any state W of $\mathcal{A}$ and any input letter $a \in \Sigma$, it holds $W \in \mathrm{Res}^{mp}_{\mathbf{A}}(V)$ if and only if $W + W' + a + a' \in \widehat{\mathrm{Res}}^{mp}_{\mathbf{A}'}(V + V' + a)$.

Proof. The statement follows from the previous claim and the fact that whenever $\mathbf{r} \in \mathrm{En}^{mp}_{\mathbf{A}'}(V + V')$, we have $\mathbf{r} + (a, a + a') \in \mathrm{En}^{mp}_{\mathbf{A}'}(V + V' + a)$. ∎

Consider now an input word $w = w_1 \cdots w_n \in \Sigma_\lambda^*$. Let $\pi = D_0, D_1, \ldots, D_i, \ldots \in IP^\lambda_{mp}(\mathcal{A}, w)$: the corresponding process in $\mathcal{M}$ $D_0, E_1 \ldots, E_n \in IP^\lambda_{mp}(\mathcal{M}, w)$ is obtained as follows. If $w_1 = \lambda$, then $E_1 \coloneqq D_0 + D_0' \in \widehat{Res}^{mp}_{\mathbf{A}'}(D_0)$ since the only reactions that are enabled are those in $\langle \mathbf{A_p} \rangle$. If $w_1 \neq \lambda$, then $E_1 \coloneqq D_0 + D_0' + w_1 + w_1' \in \widehat{Res}^{mp}_{\mathbf{A}'}(D_0 + w_1)$, by the second claim. Therefore in the next steps of the process we have:

$$\begin{cases} E_{k+1} = D_k + D_k' + w_{k+1} + w_{k+1}' \in \widehat{Res}^{mp}_{\mathbf{A}'}(E_k + w_{k+1}) & \text{if } w_{k+1} \neq \lambda \\ E_{k+1} = D_k + D_k' \in \widehat{Res}^{mp}_{\mathbf{A}'}(E_k) & \text{if } w_{k+1} = \lambda \text{ or } k \geqslant n. \end{cases}$$

Hence the processes of $\mathcal{M}$ mimic those of $\mathcal{A}$ with one step of delay, i.e. E_1 contains D_0, E_2 contains D_1, and so forth. If $\pi \in AIP^\lambda_{mp}(\mathcal{A}, w)$ then there exists $m \geqslant |w|$ such that $D_m \in S_f$; this holds true if and only if $E_{m+1} \in S_f'$. Therefore $\pi \in AIP^\lambda_{mp}(\mathcal{A}, w)$ if and only if $\pi' \in AIP^\lambda_{mp}(\mathcal{M}, w)$, and hence we obtain the thesis. □

The following corollary, showing the computational universality of $\mathcal{CPRA}^\lambda_{mp}$, follows directly from [209, Theorem 1] and Theorem 7.3.1.

Corollary 7.3.2. $\mathcal{CPRA}^\lambda_{mp} = \mathcal{RE}$.

7.4 CONCLUSIONS

We have investigated the computational power of both deterministic and non-deterministic chemical pure reaction automata working in mp manner as language acceptors, and proved that, while non-deterministic CPRA are Turing complete, DCPRA are not.

Several research questions remain open and are worthy of being investigated further. First of all, while we proved that DCPRA are not universal, their exact computation power is unknown. Hence, it would be interesting to see the relation between the class of languages recognised by DCPRA and the Chomsky hierarchy: does that class correspond to (or is neatly contained between) some level of the hierarchy or is it incomparable with it?

As a second question, there appears to be some relation between CPRA working in mp manner and Petri nets, similar to what happens with CPRA working in the sequential manner. Can this relation be formalised and, possibly, show that chemical (pure) reaction automata and their variations represent the "chemical version" of Petri net?

Recently, an additional working mode, the *maximally reactive* manner, was introduced in the previous chapter (Definition 6.1.2). Thus, it would be interesting to investigate the computational power given by this new manner of execution for CPRA and DCPRA. It seems likely that adapting the proof of Theorem 6.2.2 in the chemical reaction setting, we could get $\mathcal{CRA}^\lambda_{mp} \subseteq \mathcal{CPRA}^\lambda_{mr}$. In particular, exploring the effect of different ways in which the reactions are selected and the interplay with determinism can increase our understanding of *where* the computational power of chemical pure reaction automata comes from.

Finally, the proposed working modes have some analogies with the existing derivation modes for P systems. Thus, it would be an interesting research direction to see

which modes can be "ported" to CPRA and DCPRA, and their influence on computational universality. In particular, a general framework to talk about working modes across multiple natural computing models is still absent, but it would be a worthwhile addition to the field.

Part III

CRYPTHOGRAPHY AND REACTION SYSTEMS

8

BENT BOOLEAN FUNCTIONS IN DNF

In this chapter we study the Disjunctive Normal Form (DNF) of maximal nonlinear Boolean functions. The nonlinearity of a Boolean function is defined as the Hamming distance of the function from the set of all linear functions. In even dimensions, functions achieving maximal nonlinearity are called *bent* functions. We show a novel result: a bent function over n variables cannot contain any clause with less than $n/2$ variables in any of its DNF representations, see Theorem 8.2.2. See Table 8 for some examples of non-bent functions. A similar result, Theorem 8.2.5, also holds in the case of highly nonlinear *balanced* functions, where the number of ones equals the number of zeros in the truth table. These two properties, nonlinearity and balancedness, are crucial for cryptographic applications [68] and the theoretical results in this chapter will be useful later in Chapter 9 for designing a competitive evolutionary algorithm based on RSs that generates highly (balanced) nonlinear Boolean functions.

	function type
$n \geqslant 6$	$(x_i \wedge x_j) \vee \varphi(x)$
$n \geqslant 6$	$(x_i \wedge \neg x_j) \vee \varphi(x)$
$n \geqslant 6$	$(\neg x_i \wedge \neg x_j) \vee \varphi(x)$
$n \geqslant 8$	$(x_i \wedge x_j \wedge x_k) \vee \varphi(x)$
$n \geqslant 8$	$(x_i \wedge x_j \wedge \neg x_k) \vee \varphi(x)$
$n \geqslant 8$	$(x_i \wedge \neg x_j \wedge \neg x_k) \vee \varphi(x)$
$n \geqslant 8$	$(\neg x_i \wedge \neg x_j \wedge \neg x_k) \vee \varphi(x)$

Table 8. Example of non-bent functions. The function φ is any Boolean function over n variables.

Chapter organization. In Section 8.1, we introduce all the basic definitions related to Boolean functions needed in the proofs of the main results in Section 8.2. In Section 8.3, we discuss the obtained results.

This chapter is based on the following publication: R. Ascone, G. Bernardini, L. Manzoni, G. Pietropolli. A Novel Bio-Inspired Encoding for Evolving Cryptographic Boolean Functions. In: *Swarm and Evolutionary Computation* (*Submitted*).

8.1 BOOLEAN FUNCTIONS

This section covers the preliminary definitions and results related to Boolean functions used throughout this part of the thesis. The treatment is far from being exhaustive. The interested reader is referred to Carlet's book [68] for an in-depth discussion of Boolean functions and their properties relevant to cryptography.

Let $\mathbb{F}_2 = \{0, 1\}$ denote the finite field with two elements, where addition corresponds to the XOR operation of two bits $a, b \in \mathbb{F}_2$ (denoted as $a \oplus b$) and multiplication corresponds to the logical AND (denoted by the concatenation ab). For any $n \in \mathbb{N}$, the set of all n-tuples of bits is denoted by $\mathbb{F}_2^n$. The scalar product of two vectors $x, y \in \mathbb{F}_2^n$ is defined as $x \cdot y = \bigoplus_{i=1}^{n} x_i y_i$.

A *Boolean function* of n variables is a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2$. A natural way to uniquely represent such a function is by its *truth table*, which is the 2^n-bit vector Ω_f that specifies, for each possible input $x \in \mathbb{F}_2^n$ in lexicographic order, the corresponding output value $f(x)$. The Hamming weight $w_H(f)$ of f is defined as the number of ones in Ω_f.

Another unique representation of a Boolean function is the *Walsh transform*, which measures the correlations between a Boolean function f and linear functions, i.e. those that are defined as an XOR of a subset of the input variables. Formally, given $f : \mathbb{F}_2^n \to \mathbb{F}_2$, the Walsh transform is the mapping $W_f : \mathbb{F}_2^n \to \mathbb{Z}$ defined as:

$$W_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus a \cdot x} \tag{120}$$

for all $a \in \mathbb{F}_2^n$. The coefficient $W_f(a)$ determines the correlation between f and the linear function $a \cdot x$.

A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is called *balanced* if $w_H(f) = 2^{n-1}$, meaning that its truth table contains an equal number of zeros and ones. Balancedness is a fundamental cryptographic property of Boolean functions used in the combiner and filter model of stream ciphers: unbalanced functions present a statistical bias in their output that can be exploited in distinguishing attacks [68].

Another critical property of Boolean functions in cryptography is their *nonlinearity*, which measures the Hamming distance of the function from the set of all linear functions. The nonlinearity of a Boolean function can be expressed in terms of the Walsh transform:

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{a \in \mathbb{F}_2^n} |W_f(a)|. \tag{121}$$

The nonlinearity of any Boolean function with n inputs satisfies:

$$nl(f) \leqslant 2^{n-1} - 2^{\frac{n}{2}-1}. \tag{122}$$

The above inequality is also called the *covering radius bound*.

A Boolean function can be considered highly nonlinear if its nonlinearity is close to the covering radius bound in its class. Functions with high nonlinearity are crucial for resisting fast correlation attacks in stream ciphers [193]. Moreover, functions with higher correlation to linear ones are more susceptible to affine approximation

n	6	7	8	9	10
nl	26 (28)	56	116 (120)	240	492 (496)
$\#\mathcal{B}_n$	$1.8 \cdot 10^{19}$	$3.4 \cdot 10^{38}$	$1.2 \cdot 10^{77}$	$1.3 \cdot 10^{154}$	$1.8 \cdot 10^{308}$

Table 9. Best known nonlinearities for balanced Boolean functions up to $n = 10$ variables, along with the search space sizes. For even number of variables, the nonlinearity of bent functions is reported within parentheses.

attacks. In particular, the functions whose nonlinearity equals the maximal value $2^{n-1} - 2^{\frac{n}{2}-1}$ are called *bent*. Bent functions exist only for even values of n (since the Walsh transform only gives integer values), and moreover they are not balanced (since $W_f(\underline{0}) = \pm 2^{\frac{n}{2}} \neq 0$). Consequently, bent functions, being unbalanced, cannot be directly used in the design of stream ciphers, as we have remarked above.

When taking balancedness into account, the question of the maximum nonlinearity achievable by a Boolean function becomes more complicated. The optimal value is known only up to $n = 7$ variables, but beyond that it is still an open question. The problem is further exacerbated by the fact that the number of Boolean functions grows super-exponentially in n. Therefore, it is unfeasible to perform an exhaustive search of all Boolean functions of 6 or more variables.

Table 9 summarizes the best known nonlinearity values for balanced functions, from $n = 6$ up to $n = 10$ variables, along with the search space sizes. For even n, the table reports within parentheses the nonlinearity corresponding to the bent case.

Disjunctive Normal Form

We can represent Boolean functions via logical formulas, which combine input variables through the logical operators AND, OR, and NOT. A *literal* refers to either an input variable or its negation, while a *clause* is a conjunction (AND) of literals, with its *size* defined as the number of literals it contains. DNF is a particular representation of a logical formula structured as a disjunction (OR) of clauses. When working with the DNF representation, we use the following notation: $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT). Any Boolean function can be represented in DNF, unfortunately not in a unique way, i.e. different logical formulas can rise to the same Boolean functions. Nevertheless a logical formula in DNF can be easily converted in a Boolean circuit [86].

8.2 DNF OF BENT BOOLEAN FUNCTIONS

In this section, we present new results regarding the DNF representation of Boolean functions with maximal nonlinearity. These theoretical findings are reported because they form the foundation for the design of the evolutionary algorithm detailed in the next chapter.

We start by recalling a useful identity between the Walsh transform and the Hamming weight of a Boolean function. Let $\ell_a(x) = a \cdot x$, for any $a \in \mathbb{F}_2^n$, the following holds

$$W_f(a) = 2^n - 2w_H(f \oplus \ell_a). \tag{123}$$

Equation (123) tells us that the correlation between f and the linear function ℓ_a can also be expressed in terms of the Hamming distance between their truth tables. Equation (123) implies in particular the following.

Remark 8.2.1. A Boolean function f is balanced if and only if $W_f(\underline{0}) = 0$, where $\underline{0}$ is the zero vector of $\mathbb{F}_2^n$.

Furthermore, the *spectral radius* $W_{max}(f)$ of f is the maximum absolute value assumed by the Walsh transform, i.e. $\max_{a\in\mathbb{F}_2^n} |W_f(a)|$. Thus we can express the nonlinearity as $nl(f) = 2^{n-1} - \frac{1}{2}W_{max}(f)$.

Let $C_{k,i}(x) = x_1 \wedge \cdots \wedge x_i \wedge \neg x_{i+1} \wedge \cdots \wedge \neg x_k$ be a conjunction of the first k variables of x where the first i variables are not negated and the rest are negated.

Remark 8.2.2. The following hold.

1. If $i < k$ and x is such that $x_k = 0$, then $C_{k,i}(x) = C_{k-1,i}(x)$.
2. If $i = k$ and x is such that $x_k = 1$, then $C_{k,i}(x) = C_{k-1,i-1}(x)$.

We can compute the Hamming weight of f as $w_H(f) = \sum_{x\in\mathbb{F}_2^n} f(x)$, where we are interpreting f as a function from $\mathbb{F}_2^n$ to $\mathbb{N}$, thus for any $1 \leqslant k \leqslant n$, we can split the summation as

$$w_H(f) = \sum_{x_k=0} f|_{x_k=0}(x) + \sum_{x_k=1} f|_{x_k=1}(x) \tag{124}$$

where $\sum_{x_k=1}$ (resp. $\sum_{x_k=0}$) denotes the sum over all $x \in \mathbb{F}_2^n$ s.t. $x_k = 1$ (resp. $x_k = 0$). From now on, we assume, like in Equation (124), that each summation is over integer numbers, thus boolean inputs are identified with 0 and 1 in $\mathbb{Z}$. Given a vector $a \in \mathbb{F}_2^n$ and any $1 \leqslant i \leqslant n$, we define $p_i(a) = \oplus_{j=1}^{i} a_j$, i.e. $p_i(a)$ is the parity of the number of ones in the vector $(a_1, \ldots, a_i) \in \mathbb{F}_2^i$.

The following lemma is at the heart of all the next proofs, where we prove that adding a clause to a given function impose a restriction on the Walsh transform, i.e. a certain linear combination (with coefficients $+1$ and -1) of some values of the Walsh transform must be equal to -2^n.

Lemma 8.2.1. *Given* $\varphi : \mathbb{F}_2^n \to \mathbb{F}_2$ *and any* $1 \leqslant k \leqslant n$, $0 \leqslant i \leqslant k$, *let* $f : \mathbb{F}_2^n \to \mathbb{F}_2$ *be defined as* $f(x) = \varphi(x) \vee C_{k,i}(x)$. *Then*

$$\sum_{a=(a_1,\ldots,a_k,0,\ldots,0)\in\mathbb{F}_2^n} (-1)^{p_i(a)} W_f(a) = -2^n. \tag{125}$$

Proof. We prove Equation (125) by full induction on k, that is, we prove that the statement being true for some k and all n implies that it is true for $k+1$ and all n.

BASE CASE: $k = 1$. We consider the case where $i = 0$, i.e. $f(x) = \varphi(x) \vee (\neg x_1)$. We first notice that

$$w_H(f) = \sum_{x_1=0} 1 + \sum_{x_1=1} \varphi(x) = 2^{n-1} + \sum_{x_1=1} \varphi(x). \tag{126}$$

Let $e_1 = (1, 0, \ldots, 0) \in \mathbb{F}_2^n$. Notice that in this case, $\ell_{e_1}(x) = 1$ if and only if $x_1 = 1$, implying in particular that $f(x) \oplus \ell_{e_1}(x) = 1$ if either $x_1 = 0$, or $x_1 = 1$

and $\varphi(x) = 0$. Furthermore, note that for any two boolean values a and b, it holds that $a \vee b = a \oplus b \oplus ab$. We thus have

$$\begin{aligned} f(x) \oplus \ell_{e_1}(x) &= (\varphi(x) \vee (1 \oplus x_1)) \oplus x_1 \\ &= (\varphi(x) \oplus 1 \oplus x_1 \oplus (1 \oplus x_1)\varphi(x)) \oplus x_1 = 1 \oplus x_1\varphi(x). \end{aligned}$$

Thus, $w_H(f \oplus \ell_{e_1}) = \sum_{x_1=0} 1 + \sum_{x_1=1} (1 \oplus \varphi(x))$. Combining this with Equation (126), we obtain that

$$\begin{aligned} w_H(f) + w_H(f \oplus \ell_{e_1}) &= 2 \cdot 2^{n-1} + \sum_{x_1=1} (1 \oplus \varphi(x)) + \sum_{x_1=1} \varphi(x) \\ &= 2 \cdot 2^{n-1} + 2^{n-1} = 3 \cdot 2^{n-1} \end{aligned}$$

since $\sum_{x_1=1} (1 \oplus \varphi(x))$ and $\sum_{x_1=1} \varphi(x)$ together sum up to 2^{n-1}. Finally, by applying Equation (123) we get:

$$\begin{aligned} \sum_{a=(a_1,0,\ldots,0)\in\mathbb{F}_2^n} (-1)^0 W_f(a) &= W_f(\underline{0}) + W_f(e_1) \\ &\overset{\text{(Eq. 123)}}{=} 2 \cdot 2^n - 2(w_H(f) + w_H(f \oplus \ell_{e_1})) \\ &= 2 \cdot 2^n - 2 \cdot 3 \cdot 2^{n-1} = -2^n. \end{aligned}$$

The case $i = 1$ follows similarly.

INDUCTIVE CASE: $k - 1 \Rightarrow k$. Consider the case $i < k$ (the case $i = k$ follows similarly). We define $A = \{a = (a_1, \ldots, a_k, 0, \ldots, 0) \in \mathbb{F}_2^n\}$ and we want to evaluate $\sum_{a\in A} (-1)^{p_i(a)} w_H(f \oplus \ell_a)$. By applying Equation (124) and Remark 8.2.2.1, and since $x_k = 1$ implies that $C_{k,i}(x) = 0$, we have that

$$\begin{aligned} w_H(f \oplus \ell_a) &= \sum_{x\in\mathbb{F}_2^n} f(x) \oplus a \cdot x \\ &\overset{\text{(Eq. 124)}}{=} \sum_{x_k=0} (\varphi(x) \vee C_{k,i}(x) \oplus a \cdot x) + \sum_{x_k=1} \varphi(x) \oplus a \cdot x \\ &\overset{\text{(Rk. 8.2.2.1)}}{=} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a \cdot x) + \sum_{x_k=1} \varphi(x) \oplus a \cdot x. \end{aligned}$$

Given any $a \in A$, we denote $a^0 = (a_1, \ldots, a_{k-1}, 0, \ldots, 0) \in A$ and $a^1 = (a_1, \ldots, a_{k-1}, 1, 0 \ldots, 0) \in A$, and we split A in the two sets $A_0 = \{a^0 : a \in A\}$ and $A_1 = \{a^1 : a \in A\}$: note that $A_0 \cup A_1 = A$. Then the sum of $w_H(f \oplus \ell_a)$ over A_0 is:

$$\begin{aligned} &\sum_{a^0\in A_0} (-1)^{p_i(a)} w_H(f \oplus \ell_{a^0}) = \\ &= \sum_{a^0\in A_0} (-1)^{p_i(a)} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^0 \cdot x) + \\ &\quad + \sum_{a^0\in A_0} (-1)^{p_i(a)} \sum_{x_k=1} \varphi(x) \oplus a^0 \cdot x. \end{aligned}$$

Whereas, the sum of $w_H(f \oplus \ell_a)$ over A_1 is:

$$\begin{aligned}
&\sum_{a^1 \in A_1} (-1)^{p_i(a)} w_H(f \oplus \ell_{a^1}) = \\
&= \sum_{a^1 \in A_1} (-1)^{p_i(a)} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^1 \cdot x) + \\
&\quad + \sum_{a^1 \in A_1} (-1)^{p_i(a)} \sum_{x_k=1} \varphi(x) \oplus a^1 \cdot x \\
&= \sum_{a^0 \in A_0} (-1)^{p_i(a)} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^0 \cdot x) + \\
&\quad + \sum_{a^0 \in A_0} (-1)^{p_i(a)} \sum_{x_k=1} \varphi(x) \oplus a^0 \cdot x \oplus 1.
\end{aligned}$$

The last equality holds since when $x_k = 0$ then $a^0 \cdot x = a^1 \cdot x$, and when $x_k = 1$ then $a^1 \cdot x = a^0 \cdot x \oplus 1$. Since

$$\begin{aligned}
&\sum_{x_k=1} ((\varphi(x) \oplus a^0 \cdot x \oplus 1) + (\varphi(x) \oplus a^0 \cdot x)) = \qquad (127) \\
&= \sum_{x_k=1} \neg(\varphi(x) \oplus a^0 \cdot x) + (\varphi(x) \oplus a^0 \cdot x) = \sum_{x_k=1} 1 = 2^{n-1},
\end{aligned}$$

we obtain that

$$\begin{aligned}
&\sum_{a \in A} (-1)^{p_i(a)} w_H(f \oplus \ell_a) = \qquad (128) \\
&\overset{(\text{Eq. } 127)}{=} 2 \sum_{a^0 \in A_0} (-1)^{p_i(a)} \sum_{x_k=0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^0 \cdot x) + \qquad (129) \\
&\qquad + \sum_{a^0 \in A_0} (-1)^{p_i(a)} 2^{n-1}.
\end{aligned}$$

Let $\pi_k : \mathbb{F}_2^n \to \mathbb{F}_2^{n-1}$, $\pi_k(x) = (x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n)$ and $\iota_k : \mathbb{F}_2^{n-1} \to \mathbb{F}_2^n$, $\iota_k(x) = (x_1, \ldots, x_{k-1}, 0, x_k, \ldots, x_{n-1})$. Remark that $\pi_k(\iota_k(x)) = x$ for any $x \in \mathbb{F}_2^{n-1}$. Consider the function $f' : \mathbb{F}_2^{n-1} \to \mathbb{F}_2$, $f'(x) = \varphi(\iota_k(x)) \vee C_{k-1,i}(\iota_k(x))$. Since the form of f' is the same form as f but with $C_{k-1,i}$ instead of $C_{k,i}$, the inductive hypothesis applies. Given any $b \in \mathbb{F}_2^{n-1}$, we have that

$$\begin{aligned}
w_H(f' \oplus \ell_b) &= \sum_{x \in \mathbb{F}_2^{n-1}} f'(x) \oplus b \cdot x \\
&= \sum_{x \in \mathbb{F}_2^{n-1}} \varphi(\iota_k(x)) \vee C_{k-1,i}(\iota_k(x)) \oplus b \cdot x \\
&= \sum_{x \in \mathbb{F}_2^n : x_k = 0} (\varphi(x) \vee C_{k-1,i}(x) \oplus \iota_k(b) \cdot x).
\end{aligned}$$

Therefore, since $\iota_k(\pi_k(a^0)) = a^0$ for any $a^0 \in A_0$, we always have that

$$\sum_{x \in \mathbb{F}_2^n : x_k = 0} (\varphi(x) \vee C_{k-1,i}(x) \oplus a^0 \cdot x) = w_H(f' \oplus \ell_{\pi_k(a^0)}). \qquad (130)$$

Furthermore, since f' is over $n-1$ variables, for any $b \in \mathbb{F}_2^{n-1}$ Equation (123) gives us

$$2w_H(f' \oplus \ell_b) = 2^{n-1} - W_{f'}(b). \tag{131}$$

By applying Equations (128), (130), and (131), we get:

$$\begin{aligned}
&\sum_{a \in A} (-1)^{p_i(a)} w_H(f \oplus \ell_a) = \\
&\overset{\text{(Eq. 128,130)}}{=} \sum_{a^0 \in A_0} (-1)^{p_i(a)} 2w_H(f' \oplus \ell_{\pi_k(a^0)}) + \sum_{a^0 \in A_0} (-1)^{p_i(a)} 2^{n-1} \\
&\overset{\text{(Eq. 131)}}{=} \sum_{a^0 \in A_0} (-1)^{p_i(a)} (2^{n-1} - W_{f'}(\pi_k(a^0))) + \sum_{a^0 \in A_0} (-1)^{p_i(a)} 2^{n-1} \\
&= \sum_{a^0 \in A_0} (-1)^{p_i(a)} 2^n - \sum_{a^0 \in A_0} (-1)^{p_i(a)} W_{f'}(\pi_k(a^0)) \\
&= \sum_{a^0 \in A_0} (-1)^{p_i(a)} 2^n - \sum_{b \in \pi_k(A_0)} (-1)^{p_i(b)} W_{f'}(b) \\
&= \sum_{a^0 \in A_0} (-1)^{p_i(a)} 2^n - \sum_{b=(b_1,\ldots,b_{k-1},0,\ldots,0) \in \mathbb{F}_2^{n-1}} (-1)^{p_i(b)} W_{f'}(b) \\
&= 2^n \sum_{a^0 \in A_0} (-1)^{p_i(a)} - (-2^{n-1}),
\end{aligned}$$

where the last equality holds by the inductive hypothesis. Finally, we obtain Equation (125):

$$\begin{aligned}
&\sum_{a \in A} (-1)^{p_i(a)} W_f(a) = \\
&= \sum_{a \in A} (-1)^{p_i(a)} 2^n - 2 \sum_{a \in A} (-1)^{p_i(a)} w_H(f \oplus \ell_a) \\
&= 2^n \sum_{a \in A} (-1)^{p_i(a)} - 2 \left(\sum_{a^0 \in A_0} (-1)^{p_i(a)} 2^n + 2^{n-1} \right) \\
&= 2^n \left(2 \sum_{a^0 \in A_0} (-1)^{p_i(a)} \right) - 2^n \left(2 \sum_{a^0 \in A_0} (-1)^{p_i(a)} \right) - 2^n \\
&= -2^n,
\end{aligned}$$

where $\sum_{a \in A} (-1)^{p_i(a)} = 2 \sum_{a_0 \in A_0} (-1)^{p_i(a)}$ since $i < k$ and thus the value of a_k does not influence the result of $(-1)^{p_i(a)}$. □

We now focus on bent functions, where the number of variables n is even. The following results establish a lower bound on the size of the clauses in the DNF representation of a Boolean function for it to be bent:

Theorem 8.2.2. *Given any $\varphi : \mathbb{F}_2^n \to \mathbb{F}_2$, n even, and given $C(x)$ a conjunction of at most $k < \frac{n}{2}$ literals, the function $f(x) = C(x) \vee \varphi(x)$ is never bent.*

Proof. Up to renaming the variables, we can assume w.l.o.g. that $C(x) = C_{k,i}(x)$ for some $0 \leqslant i \leqslant k$. By Lemma 8.2.1 and the triangle inequality, we deduce that

$$\sum_{a=(a_1,\ldots,a_k,0,\ldots,0)\in\mathbb{F}_2^n} |W_f(a)| \geqslant 2^n.$$

Recall that $W_{max}(f) = \max_{a\in\mathbb{F}_2^n} |W_f(a)|$, therefore

$$2^n \leqslant \sum_{a=(a_1,\ldots,a_k,0,\ldots,0)\in\mathbb{F}_2^n} |W_f(a)| \leqslant 2^k W_{max}(f)$$

thus $W_{max}(f) \geqslant 2^{n-k}$. Therefore, by the definition of nonlinearity:

$$nl(f) = 2^{n-1} - \frac{1}{2}W_{max}(f) \leqslant 2^{n-1} - 2^{n-k-1} < 2^{n-1} - 2^{\frac{n}{2}-1}.$$ □

Corollary 8.2.3. *In any DNF representation of a bent function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, all clauses have a size of at least $\frac{n}{2}$.*

In the next statement, we demonstrate that the bound of Corollary 8.2.3 is sharp. Specifically, we show the existence of bent functions with clauses of size $\frac{n}{2}$ in their DNF representation.

Proposition 8.2.4. *For each even $n \in \mathbb{N}$, there exist bent functions $f : \mathbb{F}_2^n \to \mathbb{F}_2$ with at least a clause of size $\frac{n}{2}$ in their DNF representation.*

Proof. Given an even integer n, we define inductively $q_n(x) = q_{n-2}(x) \oplus x_{n-1}x_n$ where $q_2(x) = x_1x_2$. The function q_n is a quadratic bent function, thus also $f_n(x) = \neg q_n(x) = q_n(x) \oplus 1$ is a quadratic bent function: we prove by induction that f_n has a clause of size $\frac{n}{2}$ in its DNF representation. Remark that $f_n(x) = q_n(x) \oplus 1 = q_{n-2} \oplus x_{n-1}x_n \oplus 1 = f_{n-2} \oplus x_{n-1}x_n$ and $a \oplus b = (a \wedge \neg b) \vee (\neg a \wedge b)$.

BASE CASE: $n = 2$. Since $\neg q_2(x) = x_1x_2 \oplus 1 = \neg x_1 \vee \neg x_2$, in the base case the statement holds.

INDUCTIVE CASE: $n - 2 \Rightarrow n$. By the definition of f_n, we have

$$\begin{aligned} f_n(x) &= f_{n-2}(x) \oplus x_{n-1}x_n \\ &= (f_{n-2}(x) \wedge \neg(x_{n-1} \wedge x_n)) \vee (\neg f_{n-2}(x) \wedge x_{n-1} \wedge x_n) \\ &= (f_{n-2}(x) \wedge \neg x_{n-1}) \vee (f_{n-2}(x) \wedge \neg x_n) \\ &\qquad \vee (\neg f_{n-2}(x) \wedge x_{n-1} \wedge x_n) \\ &= g_1(x) \vee g_2(x) \vee g_3(x). \end{aligned}$$

By the inductive hypothesis we know that f_{n-2} has a DNF representation $f_{n-2}(x) = C_1 \vee \cdots \vee C_m$ with at least a clause C_i of size $\frac{n-2}{2}$. Starting from this DNF representation, we can build a DNF representation for f_n containing a clause of size $\frac{n}{2}$. Indeed, $g_1(x) = f_{n-2}(x) \wedge \neg x_{n-1} = (C_1 \wedge \neg x_{n-1}) \vee \cdots \vee (C_m \wedge \neg x_{n-1})$ is a DNF representation of g_1 with at least a clause of size $\frac{n-2}{2} + 1 = \frac{n}{2}$. The same argument applies to g_2. We can finally choose any DNF decomposition of g_3 to get a DNF representation of f_n where there exists a clause of size $\frac{n}{2}$. □

Finally, we show that a similar result as Theorem 8.2.2 applies to balanced Boolean functions with maximal nonlinearity, for all $n \leqslant 15$ (odds n included).

Theorem 8.2.5. *Given any $\varphi : \mathbb{F}_2^n \to \mathbb{F}_2$ and $n \leqslant 15$, let $C(x)$ be a conjunction of at most $k < \lceil \frac{n}{2} \rceil$ literals such that the function $f(x) = C(x) \vee \varphi(x)$ is balanced. Then, f does not have the maximal known nonlinearity value.*

Proof. Up to renaming the variables, we can assume w.l.o.g. that $C(x) = C_{k,i}(x)$ for some $0 \leqslant i \leqslant k$. Recall that by Remark 8.2.1 f is balanced if and only if $W_f(\underline{0}) = 0$. Using Lemma 8.2.1, we get that

$$2^n \leqslant \sum_{a=(a_1,\ldots,a_k,0,\ldots,0)\in\mathbb{F}_2^n} |W_f(a)| \leqslant (2^k - 1)W_{max}(f)$$

thus $W_{max}(f) \geqslant \frac{2^n}{2^k-1}$. Therefore, since $k \leqslant \lceil \frac{n}{2} \rceil - 1$, we get

$$nl(f) = 2^{n-1} - \frac{1}{2}W_{max}(f) \leqslant 2^{n-1} - \frac{2^{n-1}}{2^k - 1} \leqslant 2^{n-1} - \frac{2^{n-1}}{2^{\lceil \frac{n}{2} \rceil - 1} - 1}.$$

To conclude, when we evaluate the right-hand side for $n \leqslant 15$, we always obtain a smaller value than the best-known nonlinearity value (see [68, Table 3.1]). □

Corollary 8.2.6. *In any DNF representation of a maximally nonlinear balanced function $f : \mathbb{F}_2^n \to \mathbb{F}_2$, $n \leqslant 15$, we can not have a clause of size less than $\lceil \frac{n}{2} \rceil$.*

8.3 CONCLUSIONS

In this chapter, we studied the DNF of bent and highly nonlinear balanced Boolean functions. The DNF is not the standard representation used for cryptographic functions; in the literature, the most used representation is Algebraic Normal Form (ANF). Any function in ANF is represented by a polynomial over $\mathbb{F}_2$, using $\oplus$ and $\wedge$ as product and sum, e.g. $(x_1 \wedge x_2 \wedge x_3) \oplus (x_2 \wedge x_3)$. The ANF is always unique, and the degree of the maximal monomial is called *algebraic degree* of the function. Corollary 8.2.3 seems to be dual to the fact that the algebraic degree of bent functions is at most $\frac{n}{2}$ [68, Section 6.1.8]. A future research topic would be to understand what happens for other representations, for instance, CNF. Finally, all the results in this chapter rely on the technical Lemma 8.2.1, which could be used to prove other interesting questions on Boolean functions.

9

EVOLUTIONARY BOOLEAN REACTION SYSTEMS

In this chapter, we use the results from Chapter 8 to solve the complex optimization problem of discovering Boolean functions that are highly nonlinear, also under the constraint of balancedness. The difficulty of this problem lies in the search space growing super-exponentially in the number of variables (see Table 9). Evolutionary approaches, including Genetic Algorithms (GA) and Genetic Programming (GP) have been successfully applied to overcome this difficulty. The major drawback of these methods is that they evolve functions through encodings that are either exponential in the input size or hard to interpret. To address this problem, we propose Evolutionary Boolean Reaction Systems (EvoBRS): an optimization method based on RSs. One best quality of EvoBRS is that, unlike the existing GA and GP approaches, it uses a compact and easily interpretable function representation: Disjunctive Normal Form (DNF), in turn encoded as a special type of RSs (see Figure 26).

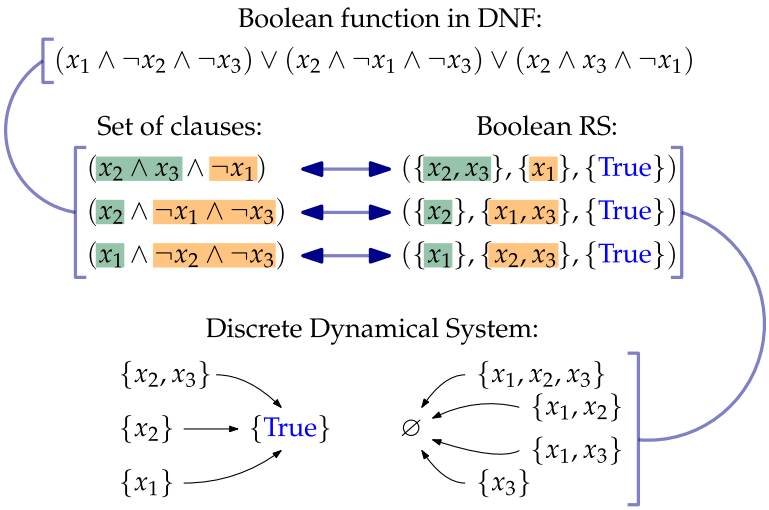


Figure 26. A representation of the relation between a Boolean formula in DNF and RSs.

Chapter organization. In Section 9.1 we introduce the problem we will study. In Sections 9.2 and 9.3 we formally define Boolean RSs and the evolutionary algorithm (EvoBRS) relying on them. In Sections 9.4 and 9.5 we present the experiments conducted and the results obtained. Finally, in Section 9.6 we discuss the results and future directions.

This chapter is based on the following publication: R. Ascone, L. Mariot, L. Manzoni, G. Pietropolli. Evolving Cryptographic Boolean Functions with Reaction Systems. In: *The Genetic and Evolutionary Computation Conference* (short paper at **GECCO 2025**) [24] and R. Ascone, G. Bernardini, L. Manzoni, G. Pietropolli. A Novel Bio-Inspired Encoding for Evolving Cryptographic Boolean Functions (*Submitted*).

9.1 INTRODUCTION

Designing Boolean functions satisfying strong – and often conflicting – cryptographic properties like nonlinearity, balancedness, and correlation immunity is a challenging task [68, 206]. Functions enjoying properties of this kind serve as key components in block and stream ciphers to resist attacks and ensure secure encryption; their search is a complex combinatorial optimization problem, as the number of candidates increases super-exponentially in the number of variables. Finding new ways to generate cryptographic Boolean functions is thus an active research area, with applications spanning several domains [43, 165, 240, 262]. Three main approaches are used in the literature to generate Boolean functions for cryptographic applications: algebraic constructions [68, Section 6.1.15], random generation, and (meta-) heuristic methods [102]. Algebraic constructions have the merit of working for many different function sizes; however, they produce the same Boolean functions when using the same starting conditions, and finding new constructions can be far from trivial [102]. Random generation is impractical due to the prohibitive size of the search space [197]. Among metaheuristic strategies, EC has demonstrated strong potential [223], the two main paradigms being GA [169] and GP [171].

The major drawback of existing EC approaches is represented by the function encodings they rely upon: GA methods use a bitstring encoding of the function's truth table, whose size is exponential in the number of variables, while GP methods use complex, hard-to-interpret trees of operators (see Figure 27 for an example).

The main goal of this chapter is to address these limitations by proposing a novel encoding to evolve Boolean functions that is at the same time polynomial in size and easily interpretable. Our encoding consists of the direct translation of the DNF of a special type into RSs. The main advantage of RS-based strategies is that they do not require the definition of a problem-specific set of functional or terminal symbols, and individuals can be directly read and understood by humans.

The evolution of general RS has been first explored by Manzoni et al. [180], who proposed the paradigm of Evolutionary Reaction Systems (EvoRS) and reported promising results for several real-world binary classification problems. Inspired by this work, we introduce EvoBRS, an original evolutionary framework specifically designed to evolve individuals represented through the newly defined class of Boolean RS, that encode Boolean functions in DNF. To suitably design the key components of EvoBRS, we apply the comprehensive investigation of the theoretical properties of the DNF representation of cryptographic Boolean functions presented in Chapter 8, and use these insights to design the EvoBRS operators.

We apply our paradigm to two well-established problems related to the design of cryptographic Boolean functions: generating highly nonlinear balanced Boolean functions and bent Boolean functions, called the BAL and BENT problems, respectively, formally defined as follows:

BALANCED HIGHLY NON-LINEAR BOOLEAN FUNCTIONS (BAL)
Input: $n \in \mathbb{N}$
Output: a n-variable Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ such that f is balanced and has maximum nonlinearity.

BENT BOOLEAN FUNCTIONS (BENT)
Input: $n \in \mathbb{N}$
Output: a n-variable Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ such that f has maximum nonlinearity, i.e. f is bent.

To validate our approach, we conducted an extensive set of experiments across multiple problem instances. We compare our results with a recent study [182], which investigates multiple GA variations and introduces improved evolutionary operators. Beyond performance measures like nonlinearity and balancedness, we analyse the structural properties of the evolved solutions and show that EvoBRS not only generates Boolean functions with strong cryptographic properties but also produces compact and diverse solutions, highlighting the benefits of RS-based representations.

We remark that the main goal of our work is to provide a compact, easy-to-interpret model (as well as to unveil nontrivial connections between the seemingly unrelated domains of RS and Boolean functions), rather than maximizing the success rate of our method. We thus refrain from comparing our results with GP-based methods, which by design trade interpretability for a higher success rate (to the best of our knowledge, not yet matched by any GA-based methods). Nevertheless, we highlight that EvoBRS outperforms the state-of-the-art GA-based methods also in terms of success rate, while offering a compact, polynomially bounded output encoding. We provide a user-friendly C++ implementation of the algorithm, which allows a faster execution and a larger fitness evaluation budget.

9.1.1 *Related Work*

We begin by reviewing the main EC paradigms employed to generate Boolean functions: for a more comprehensive analysis, we refer the reader to the survey [102]. Various representations for Boolean functions have been proposed in the literature, and, building upon them, different metaheuristic approaches have been developed to explore the corresponding search spaces.

Bitstring encoding is the most straightforward and common approach, where a candidate Boolean function over n variables is encoded as a string of bits of length 2^n, corresponding directly to its truth table. This encoding is particularly suitable for many metaheuristics, especially GA, which can operate directly on bitstrings. The first effort in this direction employed a GA in 1997 [195], followed by progressively advanced GA-based algorithms [3, 25, 44, 78, 188, 196]. Other EC paradigms, including Particle Swarm Optimization [185], have also been explored in this context [187] relying on bitstring encoding. However, direct mappings to truth tables suffer from the curse of dimensionality as the number of variables increases, making this representation hardly scalable for large n.

Symbolic representations partially mitigate the scalability issues associated with bitstring encoding, although at the cost of reduced interpretability. Evolutionary Algorithms (EA)s that employ symbolic representations, such as GP, are commonly applied in this context. A GP individual representing a Boolean function typically takes the form of a syntax tree, where terminal nodes (leaves) correspond to the n Boolean variables, and internal nodes represent binary operators such as AND, OR, NOR, XOR, XNOR, NAND, or other more complex operators. GP and its variants,

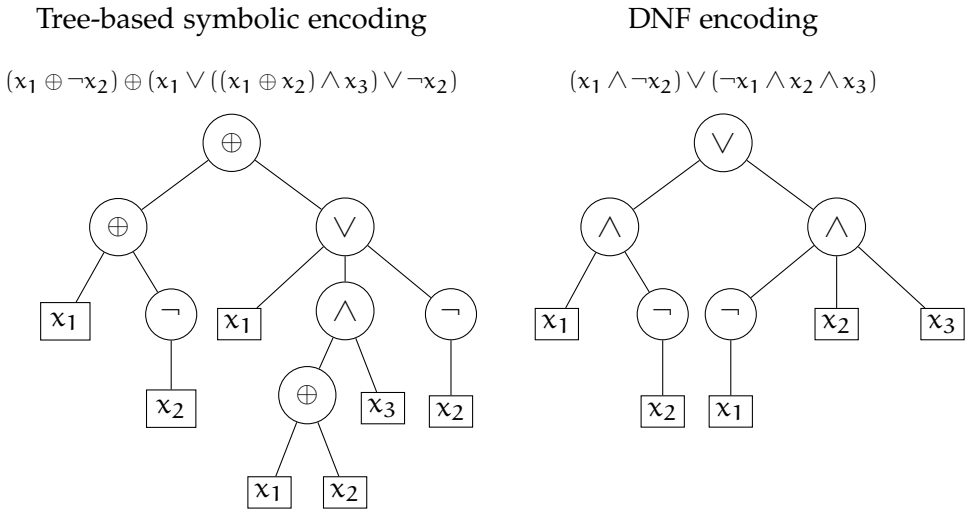


Figure 27. Three different encodings of the 3-variable function $f(x) = (x_1 \oplus \neg x_2) \oplus (x_1 \vee ((x_1 \oplus x_2) \wedge x_3) \vee \neg x_2)$. In DNF, f can be written as $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2 \wedge x_3)$; 00011100 is the bitstring (of length 2^3) representing its truth table. The DNF of any function f can also be represented via a tree with the root labeled by $\vee$, and its sons always labeled by $\wedge$. Observe that the height of a DNF tree is always bounded by 3; in contrast, the height of a general tree encoding using more operators (like those used in GP methods) is unbounded, and can be particularly large due to phenomena like bloating [102]. The uniformity of DNF trees ensures human-readable solutions without limiting the number of possible Boolean functions they can represent.

including Cartesian GP [199] and linear GP [57], have shown good performances in evolving Boolean functions [70, 149, 150, 151, 152, 198, 224, 237]. Another line of research has focused on evolving secondary Boolean constructions rather than designing entirely novel ones [69, 71, 190, 191, 222]. Nonetheless, it remains uncertain whether these evolved constructions are genuinely novel, as metaheuristics often produce numerous candidate solutions requiring exhaustive validation to assess their novelty [102]. Additionally, the size of these constructions is often unbounded, leading to non-interpretable results and a considerable bloat in many cases [71, 102].

Other representations of Boolean functions proposed in the literature include integer encoding [226] and floating-point encoding [225], though these are rarely used and suffer from similar scalability problems as bitstring encoding. Figure 27 shows an example of a Boolean function represented through bitstring and symbolic encoding, together with an example of the DNF encoding proposed in this work.

9.2 BOOLEAN REACTION SYSTEMS

RSs can be used to represent Boolean functions in DNF form [136]. A RS $\mathcal{A}$ univocally represents the DNF of a function f if it is of the following form. The background set

of $\mathcal{A}$ consists of the set of literals $X = \{x_1, \ldots, x_n\}$ appearing in the DNF plus an extra entity denoted 'True'. Each reaction univocally corresponds to a clause C of the DNF and is of the form $(R, I, \{\text{True}\})$ with $R, I \subseteq X$, i.e. the product set always consists of the single entity True, which in contrast, is never part of the reactants nor the inhibitors. The set of reactants R contains all and only the literals that appear non-negated in C; the set of inhibitors I contains all and only the negated literals of C. We define the *size* of a reaction $(R, I, \{\text{True}\})$ as $|R \cup I|$. We call RSs of this form BRS.

The states of a BRS not including the entity True are into a one-to-one correspondence with all possible assignments to the literals in X: in particular, a state $T \subseteq X$ uniquely encodes the assignment such that $x_i = 1$ if and only if $x_i \in T$, for all $i = 1, \ldots, n$. This encoding implies that a reaction is enabled by a state T if and only if the corresponding clause is satisfied by the assignment encoded with T: see also Example 9.2.1 and Figure 26. Thus, the production of the entity True corresponds to the output of the Boolean function being 1: in other words, f is 1 for an assignment encoded with a state T iff $\text{True} \in \text{res}_{\mathcal{A}}(T)$.

Note that the encoding described above is unique and, as such, the reduction can be reversed so that a Boolean function in DNF can univocally be reconstructed from any BRS. In particular, the usual requirement $R \cap I = \varnothing$ in RSs corresponds to requiring that no clause of the DNF contains $x_i \wedge \neg x_i$ for any $i = 1, \ldots, n$. We will call *balanced* any BRS representing a balanced function; and similarly, a BRS is *bent* if it encodes a bent function.

Example 9.2.1. Consider the Boolean function $f : \mathbb{F}_2^3 \to \mathbb{F}_2$ in DNF:

$$\begin{aligned} f(x) &= C_1(x) \vee C_2(x) \vee C_3(x) \\ &= (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_2 \wedge \neg x_1 \wedge \neg x_3) \vee (x_2 \wedge x_3 \wedge \neg x_1). \end{aligned}$$

Function f is univocally encoded by the BRS $\mathcal{A} = (S, \{r_1, r_2, r_3\})$, where $S = \{x_1, x_2, x_3, \text{True}\}$ and the reactions are

$$\begin{aligned} r_1 &= (\{x_1\}, \{x_2, x_3\}, \{\text{True}\}) \\ r_2 &= (\{x_2\}, \{x_1, x_3\}, \{\text{True}\}) \\ r_3 &= (\{x_2, x_3\}, \{x_1\}, \{\text{True}\}). \end{aligned}$$

In this way, any state $T \subseteq \{x_1, x_2, x_3\}$ encodes an input for f where $x_i = 1$ iff $x_i \in T$, e.g., the state $T = \{x_1, x_2\}$ corresponds to the assignment $x_1 = 1, x_2 = 1, x_3 = 0$. The symbol True represents that the output of f is 1. For instance, the reaction $r_1 = (\{x_1\}, \{x_2, x_3\}, \{\text{True}\})$ is not enabled by the state $T = \{x_1, x_2\}$, since $C_1 = (x_1 \wedge \neg x_2 \wedge \neg x_3)$ is not satisfied by the assignment $x_1 = 1, x_2 = 1, x_3 = 0$. In general, each reaction r_i corresponds to the clause C_i, and r_i is enabled by a state T iff C_i is satisfied by the assignment associated with T: in particular, f is 1 with input encoded by state T iff $\text{True} \in \text{res}_{\mathcal{A}}(T)$. See Figure 26 for a visual interpretation.

9.3 EVOLUTIONARY BOOLEAN REACTION SYSTEMS

In this section, we propose a new algorithm, *Evolutionary Boolean Reaction Systems* (EvoBRS), specifically designed for the representation and evolution of Boolean functions through BRS.

	function type	associated reaction
$n \geqslant 6$	$(x_i \wedge x_j) \vee \varphi(x)$	$(\{x_i, x_j\}, \varnothing, \{\text{True}\})$
$n \geqslant 6$	$(x_i \wedge \neg x_j) \vee \varphi(x)$	$(\{x_i\}, \{x_j\}, \{\text{True}\})$
$n \geqslant 6$	$(\neg x_i \wedge \neg x_j) \vee \varphi(x)$	$(\varnothing, \{x_i, x_j\}, \{\text{True}\})$
$n \geqslant 8$	$(x_i \wedge x_j \wedge x_k) \vee \varphi(x)$	$(\{x_i, x_j, x_k\}, \varnothing, \{\text{True}\})$
$n \geqslant 8$	$(x_i \wedge x_j \wedge \neg x_k) \vee \varphi(x)$	$(\{x_i, x_j\}, \{x_k\}, \{\text{True}\})$
$n \geqslant 8$	$(x_i \wedge \neg x_j \wedge \neg x_k) \vee \varphi(x)$	$(\{x_i\}, \{x_j, x_k\}, \{\text{True}\})$
$n \geqslant 8$	$(\neg x_i \wedge \neg x_j \wedge \neg x_k) \vee \varphi(x)$	$(\varnothing, \{x_i, x_j, x_k\}, \{\text{True}\})$

Table 10. Example of non-bent functions in Table 8 where the last column highlights the reactions that must be avoided to construct a BRS capable of representing a bent function.

Recall that a framework for evolving general RSs was originally proposed in [180]; our method, however, is fundamentally different, as the individuals we evolve are constrained to be BRS (see Section 9.2). To maintain this constraint, we define specialized operators, as well as novel fitness measures which are evaluated on the Boolean functions represented by the BRS. Our algorithm tailors the evolution to the requirements of the BENT and BAL problems and incorporates the theoretical results of Chapter 8. Indeed, in light of the correspondence between BRS and the DNF representation of Boolean functions, Theorem 8.2.2 immediately gives a lower bound on the size of reactions required for a BRS to represent a bent function. To increase the likelihood of generating individuals that are bent or close to being bent, we thus impose that the size of reactions during the initialization phase is at least $\frac{n}{2}$. For instance, for $n \geqslant 6$, a function such as $(x_i \wedge x_j) \vee \varphi(x)$ cannot be bent for any i and j. The results for small values of n are summarized in Table 10. Similarly, we can use Theorem 8.2.5 to refine the generation process for individuals in the BAL problem. For example, Corollary 8.2.6 implies that, for $n = 7$, no balanced BRS containing a reaction of size 3 can have maximal nonlinearity.

In the following subsections, we provide all the details of this new evolutionary paradigm.

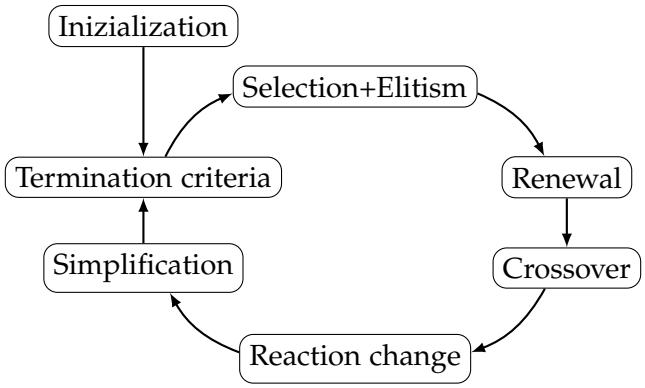


Figure 28. The execution cycle of EvoBRS.

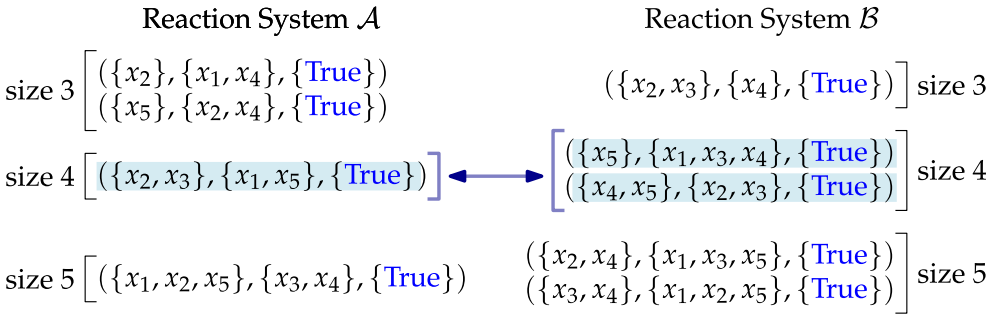


Figure 29. Example of two balanced BRS $\mathcal{A}$ and $\mathcal{B}$ over $S = \{x_1, \ldots, x_5, \text{True}\}$. The reaction system $\mathcal{A}_4$ whose reactions are $A_3 \cup B_4 \cup A_5$, is balanced.

Initialization

To define the initial population for EvoBRS, each BRS individual requires both a background set and a set of reactions. For a problem of dimension n (i.e. to generate n-variable functions), the background set of each BRS is defined as $S = \{x_1, \ldots, x_n, \text{True}\}$. Each reaction is represented as a triplet of sets of the form $(R, I, \{\text{True}\})$. To initialize the reactions, we rely on the two parameters `init_size_min` and `init_size_max`, which set the lower and upper bounds for the number of reactions allowed in each BRS. Furthermore, we randomly generate reactions of size at least $\lceil \frac{n}{2} \rceil$: this additional constraint is motivated by Theorems 8.2.2 and 8.2.5 as explained earlier.

Crossover

We define a new crossover operator, which is applied with probability p_c. Given two BRS, $\mathcal{A} = (S, A)$ and $\mathcal{B} = (S, B)$, let A_i (resp. B_i) be the set of all reactions of size i of $\mathcal{A}$ (resp. $\mathcal{B}$). For each possible size i, it generates an offspring of two BRS: $\mathcal{A}_i = (S, (A \setminus A_i) \cup B_i)$ and $\mathcal{B}_i = (S, (B \setminus B_i) \cup A_i)$, i.e. it swaps all the reactions of size i of $\mathcal{A}$ and $\mathcal{B}$. An example of this crossover (with $i = 4$) is shown in Figure 29.

This crossover is particularly effective for the Bal problem, as balanced parents are more likely to produce balanced offspring when only blocks of reactions of the same size are exchanged, as we observed from preliminary tests. Moreover, since we initialize individuals with reactions of size at least $\lceil \frac{n}{2} \rceil$, the amount of possible offspring is bounded by n, and all offspring have reactions of size at least $\lceil \frac{n}{2} \rceil$. Thus, the offspring does not violate Theorem 8.2.2 or Theorem 8.2.5.

Mutations

We define two types of mutations that operate on BRS. The first, called *renewal*, replaces an existing BRS with a completely new one, generated in the same way as the individuals during the initialization process. Each individual has a probability p_{ren} of undergoing this mutation. Renewal is a strong mutation that introduces new genetic material in each cycle to increase diversity and help explore the search space, especially in the early stages of evolution. To balance this, the second mutation is less disruptive, changing fewer reactions to allow for a more gradual evolution.

The second mutation, called *random reaction change*, modifies an existing BRS $\mathcal{A} = (S, A)$ by replacing up to one third of its reactions with new reactions of the same size. The number of reactions to be replaced is chosen uniformly at random. Each individual has a probability p_m of being affected by this mutation. As with the crossover, this mutation ensures that the resulting BRS remains consistent with Theorems 8.2.5 and 8.2.2.

Simplification

This operator reduces the number of reactions in BRS by removing some reactions from each system. The simplification operator is always applied to the new individuals generated in the evolution cycle. We say that a reaction a *covers* a reaction b when all states T enabling b enable a as well (see [108]). In the context of BRS, it can be proven that $a = (R_a, I_a, \{\text{True}\})$ covers $b = (R_b, I_b, \{\text{True}\})$ if and only if $R_a \subseteq R_b$ and $I_a \subseteq I_b$. Thus if a covers b, then the size of a is smaller than the size of b, and in particular, the products of b are always consistent with the products of a, implying that the presence of a in a RS makes b redundant. To eliminate this redundancy, the simplification operator transforms each BRS $\mathcal{A} = (S, A)$ into a new BRS $\mathcal{B} = (S, B)$, where $B \subseteq A$ is obtained from A by removing the *covering* reactions. Although it would perhaps seem more natural to remove the *covered* reactions instead of the covering ones, we notice that this choice would actually work against the evolutionary process. Instead, by removing covering reactions, we can release space for reactions of a bigger size, thus keeping more diversity in the reactions. For example, fixed $n = 6$ and a reaction $a = (\{x_1, x_2, x_3\}, \{x_4\}, \{\text{True}\})$ of size $k = 4$, a covers $(\{x_1, x_2, x_3, x_5\}, \{x_4\}, \{\text{True}\})$ or $(\{x_1, x_2, x_3\}, \{x_4, x_5\}, \{\text{True}\})$ (and many more). If we kept a instead, all of the covered reactions would never be used.

Fitness

We define two distinct fitness functions, tailored to the specific problem we are investigating. The fitness values are evaluated on the function represented by the BRS individuals. The goal for evolving bent BRS is to maximize nonlinearity. Thus, we define the fitness function as:

$$\text{fit}_{\text{bent}}(f) = \text{nl}(f) + \frac{2^n - \#\text{maxvalues}}{2^n}$$

where $\text{nl}(f)$ represents the nonlinearity of the function f, and the term $1 - \frac{\#\text{maxvalues}}{2^n}$ represents the normalized number of instances in which the Walsh transform (see Eq. 120) of f attains its largest value. A small number of instances attaining this maximum value makes it easier for the algorithm to progress to the next higher nonlinearity value.

For highly nonlinear balanced functions, we use a more complex fitness function [71]:

$$\begin{aligned}\text{fit}_{\text{bal}}(f) &= -\text{unbal} + \delta_{\text{unbal}} \left(\text{nl}(f) + \frac{2^n - \#\text{maxvalues}}{2^n} \right) \\ &= -\text{unbal} + \delta_{\text{unbal}}\, \text{fit}_{\text{bent}}(f),\end{aligned}$$

where $\mathtt{unbal}$ is the *unbalancedness*, defined as the smallest number of bits in the truth table that need to be flipped to make f balanced. The negative sign penalizes unbalancedness, ensuring that balancedness is prioritized during the optimization process. The delta function $\delta_{\mathtt{unbal}}$ takes the value one when $\mathtt{unbal} = 0$ (i.e. f is balanced) and zero otherwise. This mechanism ensures the fitness function initially focuses on creating a balanced population before shifting its focus to maximizing nonlinearity.

EvoBRS Cycle

At the beginning, the initial population is generated. Then, the evolutionary cycle (illustrated in Figure 28) begins. The process starts with selection, where elitism preserves the best individuals to maintain high-quality solutions in the population. After selection, the renewal mutation is applied. Next, parents are chosen via tournament selection for crossover, followed by the application of the random reaction change mutation. The cycle ends with simplification. Then, syntactic duplicates (BRS with identical reaction sets) are eliminated to prevent equivalent solutions from persisting in the population. At the end of each generation, we check whether the stopping criteria have been met, either reaching the maximum number of fitness evaluations or fitness achieving the maximal nonlinearity thresholds (reported in Table 9). If not, the process repeats.

9.4 EXPERIMENTAL METHODOLOGY

In this section, we present the experimental settings for running the EvoBRS algorithm, as well as the algorithms we use for comparison. Our code is implemented in C++ to ensure computational efficiency and is publicly available at `https://github.com/roccoasc1/evobrs`.

Algorithms Selected for Comparison

In [182], the authors conduct a comprehensive evaluation and comparison of different GA methods for solving the BAL and BENT problems. Their study includes a detailed statistical analysis and introduces new balanced crossover operators that improve results for the BAL problem. To assess the performance of EvoBRS, we focus on the best-performing GA algorithms identified in that work. Specifically, we consider three GA algorithms, i.e. OP, CB, and MoO, named after their crossover strategies: one-point, counter-based, and map-of-ones crossover, respectively. While OP employs a standard crossover commonly used in Boolean frameworks, CB and MoO use balanced crossovers designed to evolve balanced functions.

For our comparison, we adopt the same framework used in the original paper [182], including the evolution cycle, evolutionary operators, fitness functions, and hyperparameters, which were already fine-tuned by the authors.

Experimental Settings

Table 11 summarizes the problems evaluated in our experiments. The second column specifies the problem instances for each category, identified by the number of vari-

Problem	**Instance**	`init_size_min`	`init_size_max`	Fit evaluations
BAL	$n = 6$	80	100	10^5
	$n = 7$	120	160	$5 \cdot 10^5$
	$n = 8$	240	320	$5 \cdot 10^6$
BENT	$n = 6$	80	100	10^5
	$n = 8$	240	320	10^7
	$n = 10$	320	480	$5 \cdot 10^7$

Table 11. Problem instances, initialization sizes and number of fitness evaluations.

ables n. The third and fourth columns provide the minimum and maximum initial sizes of the BRS individuals (`init_size_min` and `init_size_max`), which are the only parameters of the algorithms that change depending on the input dimension.

In all experiments, we used a population size of $n_{pop} = 100$ individuals. We use a tournament size of $n_t = 4$. We scaled the number of fitness evaluations depending on the input n, since the search space grows super-exponentially (see Table 9). Fitness evaluations are used as a stopping criterion to ensure a fair comparison, with all methods terminated after the same number of evaluations. A random reaction change mutation was applied to each individual with a probability of 0.8, a renewal mutation with a probability of 0.1, and crossover with a probability of 0.2. Elitism preserved the best five individuals for the next generations.

We conducted 30 independent runs for each problem instance to ensure statistically significant results. For comparisons, we used the Wilcoxon rank-sum test to assess statistical significance, with the null hypothesis of equality.

9.5 EXPERIMENTAL RESULTS

This section presents the results of the experimental campaign. The first part (Section 9.5.1) focuses on performance, while the second (Section 9.5.2) addresses the diversity and interpretability of the solutions generated.

9.5.1 *Performances and comparison with other algorithms*

Figures 30 and 31 summarize the results for the BAL and BENT problems, respectively. In both figures, different instances are shown in separate rows. The left panels report the progression of the average nonlinearity of the best individuals over fitness evaluations, while the right panels show the final nonlinearity distribution, both based on 30 runs. The boxplot graph for the BAL 6 instance also shows p-values from pairwise statistical tests to highlight the significance of differences. The bold p-value (with a value of 0.046) highlights the only case in which EvoBRS significantly outperformed the opponent for this instance. For the other problem, p-values (all on the order of 10^{-11}) are omitted, as they consistently confirm that EvoBRS significantly outperforms the competitors.

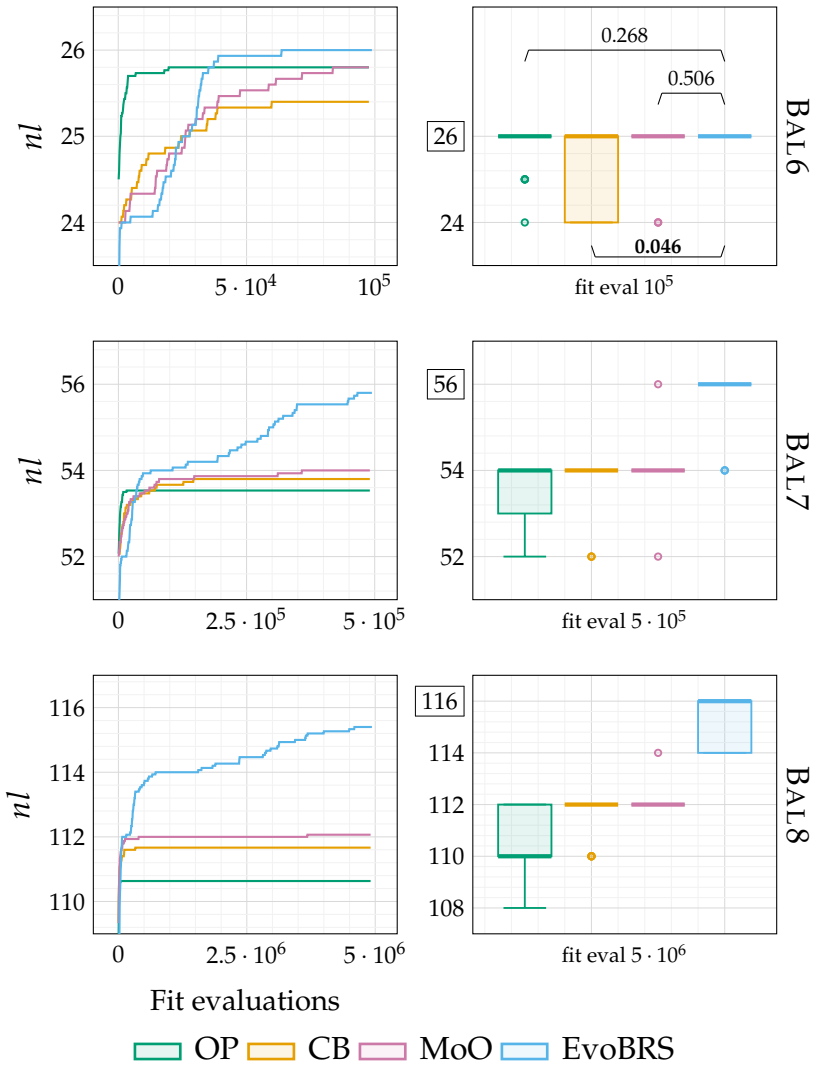


Figure 30. Progression of the average nonlinearity achieved by the best individual in the population for an increasing number of fitness evaluations (left) and the final distribution (right) represented as a boxplot across 30 independent runs for the BAL problem for $n = 6, 7, 8$. Since in the BAL6 instance the four distributions look similar, we ran a pairwise Wilcoxon rank-sum test with equality as null hypothesis between EvoBRS and each of the other methods, and decorated the boxplot with the resulting p-values. The same tests run for BAL7 and BAL8 always returned p-values on the order of 10^{-11}; we omitted these values from the figure for clarity. The maximum possible nonlinearity value for each of the instances is boxed: EvoBRS reached the maximum value in every single run on BAL6, in 27 out of 30 runs on BAL7, and in 21 out of 30 runs on BAL8.

For all BAL instances, our method reaches the known theoretical maximum nonlinearity (26, 54, and 116 for $n = 6$, 7, and 8, respectively). This occurs consistently (30 out of 30 runs) for $n = 6$, whereas in the other two cases, such consistency would likely be achieved with additional fitness evaluations. The average nonlinearity is higher than that of all the other methods in the comparison. While for $n = 6$ other approaches can also reach the maximum nonlinearity, for the other two instances all GA-based approaches fail to do so (with one exception for $n = 7$: see Figure 30).

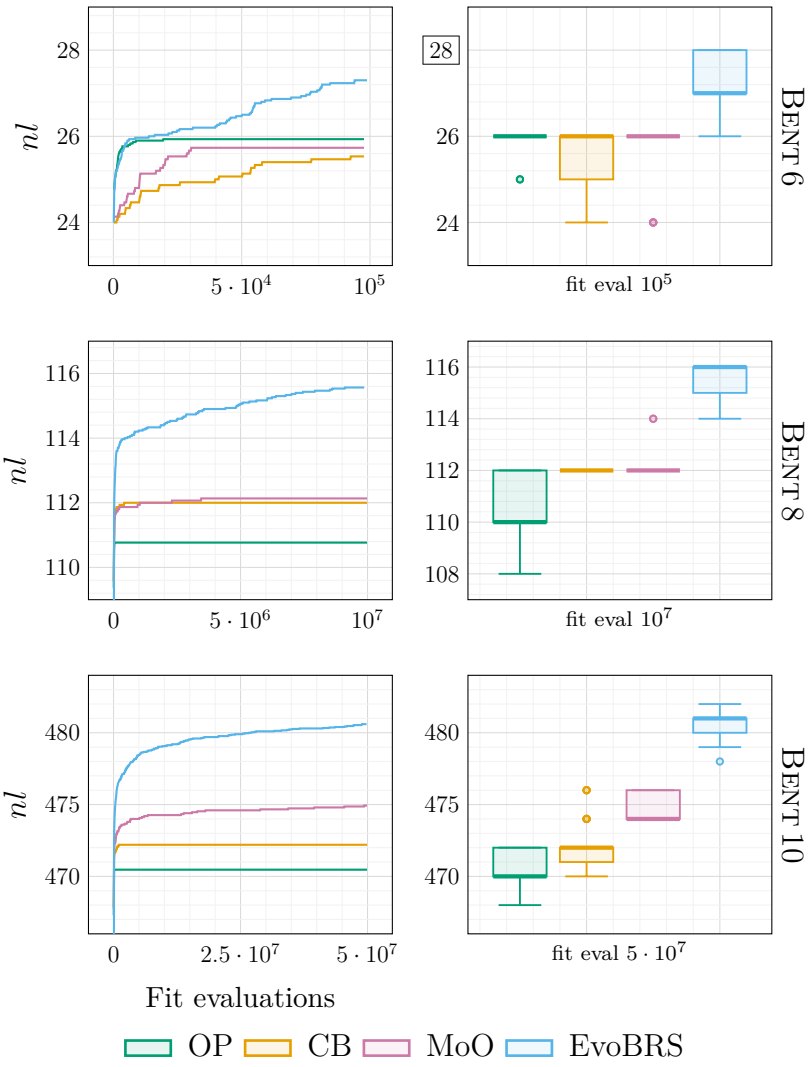


Figure 31. Average nonlinearity of the best individual per iteration (left) and final distributions across 30 runs (right) for the Bent problem with $n = 6, 8, 10$. We ran a pairwise Wilcoxon rank-sum test with quality as the null hypothesis between EvoBRS and each of the other methods: the resulting p-values in all cases were on the order of 10^{-11}, thus indicating statistically significant differences in the performance of EvoBRS and all the other methods. The maximum possible nonlinearity value for BENT6 is boxed: EvoBRS reached the maximum value in 12 out of 30 runs on this instance, while none of the other methods could reach it. The maximum value for the other instances is not shown as it was never reached by any of the methods, including EvoBRS.

Moreover, for all instances, EvoBRS starts with a lower initial nonlinearity than the other methods. The higher initial values of the GA algorithms result from their initialization: they generate balanced individuals, a straightforward task in a GA representation (i.e. creating a string with equal numbers of 0 and 1 [182]) but more challenging with a BRS representation and, in general, with compact encodings.

Focusing on the BENT problem, for $n = 6$ EvoBRS reaches the maximum theoretical nonlinearity of 28, which is not achieved by any of the competitors. For the other instances, although EvoBRS achieves high nonlinearity, it does not reach the

n	BAL	n	BENT
6	20.41 ± 2.14	6	19.86 ± 2.37
7	44.29 ± 2.43	8	94.89 ± 3.10
8	95.37 ± 2.74	10	426.09 ± 4.65

Table 12. Mean ($\pm$ standard deviation) of the semantic distances for the best individuals generated by EvoBRS, across all problems and instances considered.

maximum bound (120 for $n = 8$ and 496 for $n = 10$). However, the left panels of the plots show that the fitness of EvoBRS continues to increase throughout the evolution, suggesting that higher fitness evaluation budgets could further improve the results. In all the instances considered, EvoBRS outperforms the baselines from the early stages of the evolution, and maintains consistent fitness growth, whereas other methods tend to stagnate in local optima: see Figure 31.

Overall, EvoBRS performs better for the BAL problem than for the BENT problem. For the BAL problem, it reaches the maximum nonlinearity for all instances, unlike for the BENT problem, despite the latter using more fitness evaluations. The cause likely lies in the relative sizes of the optimal solution spaces. In particular, for a given number of variables, there are more balanced functions than functions with the weight of a bent function (Section 8.1). This gives an advantage to the evolutionary strategy, as the landscape of balanced functions might be better connected than the space of functions with a bent weight.

9.5.2 *EvoBRS solutions interpretability and diversity*

We now closely examine the functions generated by EvoBRS, focusing on interpretability and diversity. Interpretability is crucial to understand and validate the solutions, while diversity ensures that different runs produce alternative functions, offering multiple equally valid solutions to the same problem.

We assess interpretability in terms of structural simplicity and semantic transparency. A Boolean function in our BRS representation is expressed as a disjunction of reactions, each corresponding to a set of input assignments for which the output is True. This clause-level representation allows direct, human-readable mapping between the formula and the conditions under which it evaluates to True, without the need to traverse nested logical operators. For example, in the BAL problem with $n = 6$, one evolved solution is:

$$
\begin{array}{ll}
(\{x_5, x_6\}, \{x_2, x_3\}, \{\text{True}\}) & (\{x_4, x_5\}, \{x_1, x_6, x_3\}, \{\text{True}\}) \\
(\varnothing, \{x_2, x_4, x_5, x_6\}, \{\text{True}\}) & (\{x_2, x_4\}, \{x_1, x_3, x_5\}, \{\text{True}\}) \\
(\{x_1, x_2\}, \{x_5, x_6\}, \{\text{True}\}) & (\{x_1, x_6, x_3, x_4\}, \{x_2\}, \{\text{True}\}) \\
(\{x_5, x_2, x_3\}, \{x_1\}, \{\text{True}\}) & (\{x_1, x_2\}, \{x_6, x_3, x_4\}, \{\text{True}\}) \\
(\{x_1, x_3, x_5\}, \{x_4\}, \{\text{True}\}) & (\{x_5, x_1, x_3, x_4\}, \{x_6\}, \{\text{True}\}) \\
(\{x_2, x_3\}, \{x_4, x_5\}, \{\text{True}\}) &
\end{array}
$$

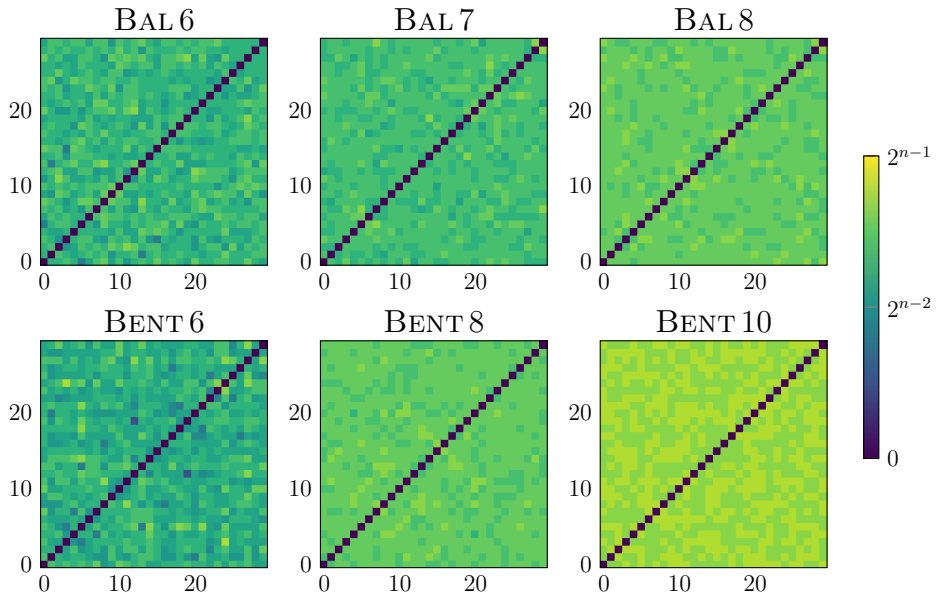


Figure 32. Heatmaps showing the permutation distances between the best EvoBRS individuals from each experiment for the BAL problem (up) and the BENT problem (bottom). The lighter the color, the more diversity in the solutions.

This function contains only 11 reactions, each involving 4–5 variables. For instance, the clause $(\{x_5, x_6\}, \{x_2, x_3\}, \{\text{True}\})$ corresponds to all the assignments where $x_2 = x_3 = 0$, $x_5 = x_6 = 1$ and x_1 and x_4 could take any value. In this way, given any assignment of the variables, it is straightforward to check whether it satisfies any of the reactions above. For example the assignment $x_1 = x_2 = 1$ and $x_3 = x_4 = x_5 = x_6 = 0$ satisfies both $(\{x_1, x_2\}, \{x_5, x_6\}, \{\text{True}\})$ and $(\{x_1, x_2\}, \{x_6, x_3, x_4\}, \{\text{True}\})$. This kind of reasoning is not possible when the function is written using more nested logic operators, as usually happens for the solutions obtained via GP (see Figure 27).

Other studies employing GP report that even after logic minimization, expressions for $n = 6$ remain too complex for manual interpretation [71]. On the other hand, when using GA methods, finding the minimal DNF representation from a truth table is an NP-hard problem [9, 255]. Our approach circumvents this issue by directly generating compact and easily implementable solutions (thanks to the simplification operator), thus eliminating the need for post-processing. Indeed, the circuit complexity required to physically implement our Boolean functions remains low. Specifically, the circuit depth, defined as the longest path from input to output, is limited to 3, while the circuit size, measured as the number of gates, is at most $\mathcal{O}(n \cdot \texttt{init\_size\_max})$. For a detailed discussion on circuit complexity, we refer to [257].

We also crucially investigate whether EvoBRS produces multiple alternative, equally valid Boolean functions or consistently converges to similar ones, as discovering new constructions is a relevant task in cryptography [102]. To do so, we measure semantic similarity between evolved functions using Hamming distance. Recall that given two Boolean functions f and g, their Hamming distance $d_H(f, g)$ is $w_H(f \oplus g)$, i.e. the number of input $x \in \mathbb{F}_2^n$ such that $f(x) \neq g(x)$ [68]. We first perform pairwise

comparisons among all solutions in each experiment and verify that none of them is identical. To quantify the magnitude of this pairwise difference, we report the mean ($\pm$ standard deviation) for both problems and all instances in Table 12. In all cases, the mean is close to 2^{n-1}, indicating that the functions differ by about half of their total length (which is 2^n).

While the Hamming distance captures semantic diversity, it does not account for the equivalence of functions under input variable permutations. To address this, we compute the *permutation distance* between each pair of functions f and g. The permutation distance is defined as $\min_{\sigma \in S_n} d_H(f, g \circ \sigma)$, where S_n is the group of permutations of n. Results are shown in Figure 32, with the BAL problem on the left and the BENT problem on the right. Each heatmap can be seen as the adjacency matrix of a weighted undirected graph with 30 nodes, each representing the best solution for one of the 30 experiments. The weight on an edge between two nodes is the permutation distance between the two corresponding solutions: the distance is zero only if the BRS are semantically equivalent under any input permutation. These results confirm that none of the solutions generated by EvoBRS are identical and that they remain highly diverse, even when evaluated using this stricter permutation distance.

9.6 CONCLUSIONS AND FUTURE DIRECTIONS

We presented EvoBRS, an evolutionary framework based on Boolean reaction systems for the design of cryptographic Boolean functions. Unlike traditional bitstring or tree-based representations, EvoBRS directly encodes Boolean functions in Disjunctive Normal Form, providing compact human-readable solutions without requiring post-processing or logic minimization.

Our experimental analysis on two canonical problems, BAL and BENT Boolean function generation, shows that EvoBRS consistently reaches the theoretical maximum nonlinearity in the BAL case, and produces highly nonlinear solutions in the BENT case, outperforming all the state-of-the-art GA methods we compared against. Furthermore, our diversity analysis confirmed that EvoBRS consistently finds semantically distinct functions across different runs, even when accounting for input variable permutations.

This work provides the first extensive analysis demonstrating the potential of compact and interpretable representations for Boolean function evolution. Considering the promising results, our work opens the door to multiple future developments such as the exploration of higher-dimensional cases. An interesting direction is also extending RS-based representations to encode Boolean functions from $\mathbb{F}_2^n \to \mathbb{F}_2^m$. Given the good results presented in this paper for $m = 1$, this extension could provide a novel approach for designing Substitution-boxes (S-boxes), which are key components in modern symmetric-key ciphers [205].

Part IV

FINAL REMARKS

10

CONCLUSIONS AND FUTURE WORK

In this final chapter, we present some relations between RSs and other natural computing models, and propose some main open problems that arise from the study conducted in this thesis.

10.1 PART I AND AUTOMATA NETWORKS

As shown by Formenti et al. [123], there is a close relation between RSs and *Boolean Automata Networks* (BANs) with fully synchronous update mode. Indeed, given a RS, we construct one automaton and one Boolean variable for each entity. By the definition of result function, an entity t is produced if and only if there exists a reaction enabled with t in the products. Furthermore, the enabling of a reaction can be encoded via a conjunction of literals (positive literals correspond to reactants, negative ones to inhibitors). Thus, we can build a BAN where each automaton is defined through a DNF formula and whose dynamics (with the fully synchronous update mode) is isomorphic to the dynamics of the RS we are reducing from. An example of this correspondence is shown in Figure 33.

$(\varnothing, \{\spadesuit, \heartsuit\}, \{\heartsuit\})$ $\qquad$ $f_\heartsuit(x) = (\neg x_\heartsuit \wedge \neg x_\spadesuit) \vee (x_\heartsuit \wedge \neg x_\spadesuit)$

$(\{\heartsuit\}, \{\spadesuit\}, \{\spadesuit, \heartsuit\})$ $\qquad$ $f_\spadesuit(x) = x_\heartsuit \wedge \neg x_\spadesuit$

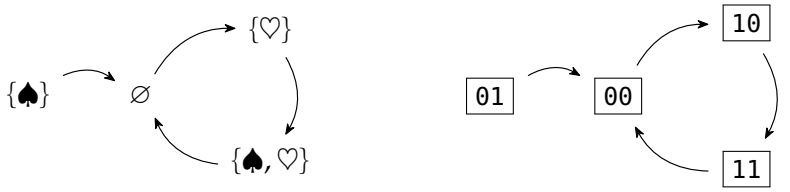


Figure 33. Representation of a RS and the associated BAN. The enabling of the reaction $(\varnothing, \{\spadesuit, \heartsuit\}, \{\heartsuit\})$ (resp. $(\{\heartsuit\}, \{\spadesuit\}, \{\spadesuit, \heartsuit\})$) is in a one-to-one correspondence to the satisfiability of $\neg x_\heartsuit \wedge \neg x_\spadesuit$ (resp. $x_\heartsuit \wedge \neg x_\spadesuit$).

In [133], the authors proved the following interesting result on Automata Networks:

Metatheorem [133]. *Any nontrivial property of the function computed by an automata network has high computational complexity.*

There, "high computational complexity" means **NP**-hard, **coNP**-hard or even **PSPACE**-hard. A similar result was proved for limit sets for Cellular Automata by Kari [164]. These kinds of results are called Rice-Like, since they semantically look similar to the famous Rice's theorem [233]. Looking at the column regarding the class $\mathcal{RS}(\infty, \infty)$ of general RSs in Table 1, we can notice that every problem is either **NP**-hard or **coNP**-hard (assuming no collapses in the polynomial hierarchy). A natural question arises:

RICE-LIKE THEOREM FOR REACTION SYSTEMS
Open problem 1: Does a Rice-Like theorem for the dynamical properties of Reaction Systems hold?

The answer is negative for the classes of resource-bounded RSs studied in Part I (see the other columns in Table 1), but it would be interesting to understand if there exists a general rule to detect the hardness of the problems in a Rice-like flavour. Furthermore, it would be interesting to complete the picture in Table 1 by solving the few cases left open.

10.2 PART II AND P AUTOMATA

A well-studied model that resembles RA is P automata (see, e.g., [89, 91, 92, 128]), the automata-like version of P systems [216, 217], where multisets of *objects* are subdivided into multiple regions (or *membranes*). Those objects evolve according to a series of rules that move the objects between the different regions and rewrite them. These rules are applied according to a given computational model, like sequential or maximally parallel (where conflicts are possible). One important difference w.r.t. RA is the subdivision of the space into cell membranes that actively participate in the computation and are essential to expand the computational power of the models [247]. As an example, consider the membrane nesting depth for P systems with active membranes with charges and its influence in the complexity classes that different depths characterise [173, 174, 246]. RA can be seen as a "degenerate" case of P automata, where only one region is present and only rules of a particular kind are admitted. While investigating different derivation modes for the selection of reactions (or rules) to be applied is a new research direction for models related to RSs, it is a well-developed field of study for P systems. In particular, asynchronous and sequential derivations already exist for P systems in addition to the most common maximal parallelism criterion, which itself can be declined in different ways. Of particular interest is the case of maximal parallelism when the P system is used as a generator for numbers of Parikh sets [6], which has some similarity with the derivation modes for PRA studied in Part II. For more details, we refer the reader to the ample literature on the topic of derivation modes for P systems [6, 7, 126, 127]. Furthermore, non-permanence in P systems has been studied in [218].

It would be interesting to characterize PRA for each possible known derivation mode, even without the constraint of chemical reactions. Pursuing this research direction could also benefit the P automata community, while clearly revealing the sources of computational power in PRA.

COMPUTATIONAL CAPABILITY OF PURE REACTION AUTOMATA
Open problem 2: Characterize the languages recognized by all variants of PRA under different update rules.

10.3 PART III AND CELLULAR AUTOMATA

Cellular Automata (CA) constitute a natural computing model introduced by Ulam and Von Neumann in [203]. For sure, it is the most famous and oldest among the computationally bioinspired models. Among the many applications of CA, cryptography has represented a significant share in recent years [161, 186], with examples ranging from CA-based pseudorandom number generators to stream ciphers, and from hash functions to public-key encryption schemes and secret sharing schemes [184]. A different direction w.r.t. combinatorial design [189] is the study of CA-based evolutionary algorithms to generate cryptographic functions [192]. In particular, CA have been used to solve even more specialized versions of the problem considered in Part III, i.e. problems with more constraints on the function's properties. It would be worthwhile to try to solve those problems with the EvoBRS framework.

CRYPTOGRAPHY AND REACTION SYSTEMS
Open problem 3: Study further applications of RS in cryptography.

10.4 NATURAL COMPUTING AND FINE-GRAINED COMPLEXITY

In Part V, we will present some applications of Fine-Grained-Complexity. To the best of the author's knowledge, this theory has never been applied in a natural computing framework. It would be of interest to apply this technique in the context of natural computing, in any of the paradigms discussed in this final section.

FINE-GRAINED COMPLEXITY IN NATURAL COMPUTING
Open problem 4: Apply Fine-Grained Complexity Theory to problems in natural computing models.

Part V

APPENDIX

A

PATTERN MATCHING WITH ELASTIC-DEGENERATE STRINGS AND ELASTIC-FOUNDER GRAPHS

In this chapter we study Elastic Degenerate (*ED*) strings and Elastic Founder (*EF*) graphs, here collectively named *variable* strings. Those objects are two representations of acyclic components of pangenomes which extend the well-known notion of indeterminate string. Recent studies have focused extensively on algorithmic tasks involving these structures and other forms of variable strings that they generalize. Among such tasks, the basic operation of matching a pattern into a text, a fundamental toolkit for pangenomic data analysis, deserves special attention. In this chapter, (1) we establish a clear taxonomy across *ED* strings and *EF* graphs, categorizing types of variable strings from the simplest linear (*solid*) string to the most complex general cases (see Figure 34); (2) we consider the problem Match(X,Y) of matching a solid or variable pattern of type X into a variable text of type Y, and investigate its time complexity when X and Y are chosen from types of variable strings in the taxonomy of (1). For all possible X and Y, we either provide a non-trivial, often sub-quadratic, upper bound for Match(X,Y), or we prove a quadratic conditional lower bound, taking as a reference the existing quadratic conditional lower bounds for Match(solid,*ED*) and Match(solid,*EF*).

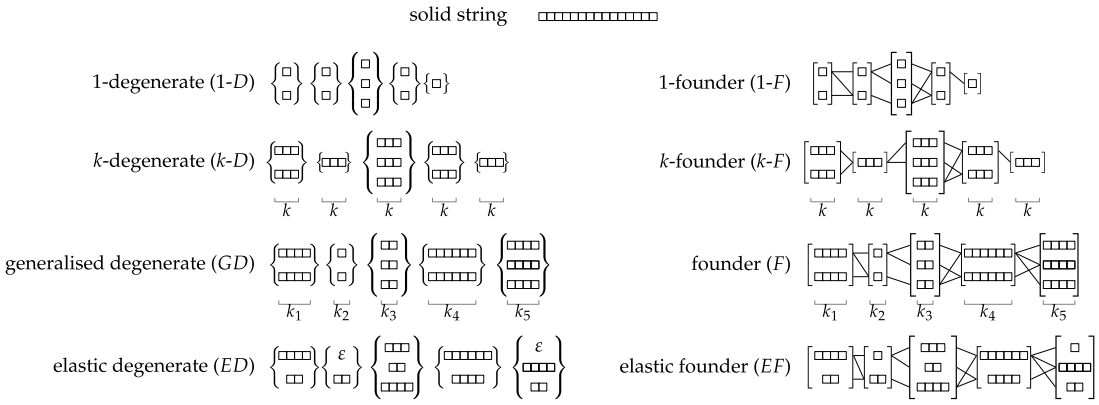


Figure 34. The landscape of variable strings considered in this chapter. The leftmost column depicts degenerate strings in increasing order of generality; the rightmost column shows an analogous progression for founder graphs. Squares represent characters, ε denotes the empty string.

This chapter is based on the following publications: R. Ascone, G. Bernardini, A. Conte, M. Equi, E. Gabory, R. Grossi, N. Pisanti. Pattern matching with Elastic-Degenerate strings and Elastic-Founder graphs. In: *Algorithms for Molecular Biology* (accepted). Extended version of [17].

A.1 INTRODUCTION

Analysing an ever-increasing number of genome sequences and determining which genome to select as a reference are two major challenges in genomic data analysis. Recently, these challenges have converged into a powerful approach: using a *pangenome* – rather than a single genome – as a reference. According to [83], a pangenome is indeed "any collection of genomic sequences to be analyzed jointly or to be used as a reference". The new -omics science *pangenomics* thus imposed a paradigm shift: in several analysis tasks, and in particular for species like humans that enjoy a widespread availability of sequencing data as well as a growing awareness of genomic variants, the simple linear genomic sequence is being replaced by more complex graph-like structures [118, 215]. Unlike a linear reference, a pangenome reference allows a simultaneous and compact representation of variations and commonalities among the underlying sequences. The most general pangenome representations are edge- and/or node-labeled directed graphs [29], such as the *variation graphs* [75, 112, 134] (with their haplotype aware version [244]) and *sequence graphs* [232]. Simpler *acyclic* alternatives to these representations are *Elastic-Degenerate strings* [143, 155] (ED strings) and the slightly more haplotype-aware *Elastic Founder graphs* [236] (EF graphs), as well as their non-elastic versions *Generalised Degenerate strings* [10, 11, 201] (GD strings) and *Founder graphs* [179] (F graphs).

Figure 34 shows the taxonomy of variable strings, from the simplest solid string (top) to the most general variable strings (bottom-left and bottom-right). 1-D strings are also known in the Bioinformatics literature as *indeterminate* strings. They have been extensively employed and investigated [2, 5, 13, 14, 55, 88, 95, 97, 103, 104, 135, 147, 153, 154, 178, 227, 228, 238, 239, 245] as they naturally represent the IUPAC encoding [159] of DNA and RNA nucleotides subsets and can be used to highlight SNPs (Single Nucleotide Polymorphisms, i.e. substitutions of single letters in the DNA sequence). A wide literature exists also for strings with wildcards [40, 41, 42, 79, 121, 141, 142] that can be seen as a special case of 1-D strings. The class k-D generalises 1-D in that all variants have length $k \geqslant 1$, while the class GD only requires the variants at any locus to be of the same length, without requiring them to have the same length all over. GD strings thus represent gapless multiple sequence alignments of fixed width, e.g. high-scoring local alignments of multiple sequences. Finally, ED strings allow variants of any size at any locus, including the empty string, which allows explicitly representing short INDELs (insertions and deletions of letters in the genomic sequence). ED strings correspond to the VCF file format [93] for genomic variants. The same taxonomy holds for founder graphs with the addition of edges that connect variants of adjacent segments according to haplotype information and the further difference that EF graphs cannot contain the empty string. The term *variable string* collectively refers to ED strings, EF graphs, and any of their above-mentioned special cases.

In contrast with more general representations like variation and string graphs, ED strings support both theoretically [15, 46, 49] and practically [67, 76, 143, 230, 231] fast *on-line* exact and approximate pattern matching [45, 47, 48] when the pattern is a solid (i.e. non-variable) string. Nevertheless, conditional lower bounds for this problem [31, 49] and the problem of indexing ED strings [137] rule out the ex-

istence of even faster methods. Efficient algorithms also exist to decide whether the intersection of the sets of sequences represented by two ED strings is non-empty [129, 130, 131] (*intersection query*). ED strings cannot be efficiently indexed, but filtering methods [219, 220, 221] could be applied to their k-mers, that can be efficiently indexed [172]. Furthermore, GD strings support fast dynamic-programming-based alignment [140, 201, 200] for approximate matching with a solid string and linear-time answer to intersection queries [10, 11].

For EF graphs, efficient off-line pattern matching algorithms are known under specific conditions which can be ensured with a linear-time construction from a gapless multiple sequence alignment (for F graphs), or a near-linear time construction from a general multiple sequence alignment (for EF graphs) [114, 117, 179, 234, 235, 254]. Here, "near-linear time" means that the time complexity is linear when parametrised in the maximum number of variations appearing in a single locus of the EF graph.

While the main advantage of variable strings is their supporting provably efficient and accurate methods, their limitation is in their expressivity: both ED strings and EF graphs are acyclic structures that can only express mismatches and INDELs, and cannot adequately represent structural genomic variation such as repetitions, translocations, or inversions. A main challenge in computational pangenomics is to balance the pangenome representations' accuracy and expressivity and the efficiency of the methods for their construction and analysis: variable strings are a trade-off between expressive power and efficiency.

This chapter aims to provide a clear taxonomy and a time-complexity landscape of matching problems in variable strings. More formally, we analyse the complexity of the problem, denoted MATCH(X, Y), of matching a solid or variable pattern of type X in a variable text of type Y: the types of variable strings we consider for the pattern and text are in Figure 34. For instance, MATCH(1-D,k-F) is the problem of finding occurrences of a 1-D pattern within a k-F graph.

OUR RESULTS For the problem of matching a solid pattern of size M into an ED or EF text of size N, it is known from the literature that an algorithm with complexity $\mathcal{O}(M^{1-\varepsilon}N)$ or $\mathcal{O}(MN^{1-\epsilon})$ with $\epsilon > 0$ would contradict the Strong Exponential Time Hypothesis (SETH) [31, 46, 49, 114, 138], and the contradiction holds even if such complexity is achieved at query time after a polynomial-time indexing step [113, 115, 137]. Quadratic-time algorithms are known [49, 76]. Moreover, strongly sub-quadratic algorithms are known for MATCH(SOLID,1-D) [81] and for MATCH(1-D ,1-D) [153], in the latter case restricted to constant-size alphabets.

As a consequence, our reference bound is quadratic: given two types X and Y of strings (variable or solid), either this is proved as a lower bound for MATCH(X, Y), or a better algorithm – an upper bound – should be exhibited. We aim to paint a complete picture of the complexity of this task for all types X of patterns and all types Y of variable texts, investigating both lower and upper bounds. All our results are anticipated in Section A.2 and summarised in Tables 13 and 14, where columns correspond to the pattern and rows correspond to the text. Note that, due to space constraints, in the tables we write $\Omega((MN)^{1-\epsilon})$, for every $\epsilon > 0$, to denote the quadratic lower bound, but in fact, all our lower bound results prove both bounds $\Omega(M^{1-\epsilon}N)$ and $\Omega(MN^{1-\epsilon})$ (for every $\epsilon > 0$).

Pattern \ Text	1-D	k-D	GD	ED
SOLID	[81] $\mathcal{O}(n + n\log^2 m)$	**Thm. A.2.1** $\mathcal{O}(N + N\log^2 m)$	**Thm. A.2.3** $\mathcal{O}(mn + N)$	**Thm. A.2.11** $\Omega((mN)^{1-\epsilon})$

(a) Complexity of matching a solid pattern in several types of degenerate strings, in increasing order of generality.

Pattern \ Text	1-F	k-F	F	EF
SOLID	**Thm. A.2.2** $\mathcal{O}(\sqrt{m}(\lvert E\rvert + N + n\log^2 m))$		**Thm. A.2.4** $\mathcal{O}(mn + N + \lvert E\rvert)$	[114] $\Omega((m\lvert E\rvert)^{1-\epsilon})$

(b) Complexity of matching a solid pattern in several types of founder graphs, in increasing order of generality.

Table 13. Complexity charts for problem MATCH(SOLID, Y), where Y ranges over columns. Symbols M and N denote the total size of the pattern and the text, respectively, while m and n denote the respective *number of segments* (see Section A.2). For graphs, $|E|$ denotes the number of edges. Green cells are for truly subquadratic $\mathcal{O}(NM^{1-\epsilon})$ (for some $\epsilon > 0$) upper bounds, yellow cells indicate that the upper bounds are subquadratic whenever $N/n = \omega(1)$, and red cells are for a quadratic lower bound $\Omega((MN)^{1-\epsilon})$ (for every $\epsilon > 0$) conditioned on SETH, even when the alphabet is of constant size and the length of the variants in each segment is $\mathcal{O}(1)$ (our bounds are even tighter, check the referred results for details). A truly subquadratic upper bound for MATCH(SOLID,1-D) was already known. A quadratic conditional lower bound for MATCH(SOLID,ED) was also known, as it is directly implied by [30, Theorem 10], but this result only holds when the maximum variant length in the ED text of length n is $\omega(\log n)$. In Theorem A.2.11 we prove that the lower bound applies also to cases where the maximum variant length is $\mathcal{O}(1)$. All the other results are novel.

A.2 DEFINITIONS AND SUMMARY OF THE RESULTS

We start with basic definitions and notation on solid strings following [87]. Given an ordered alphabet Σ whose elements are called *letters*, we denote by Σ^n the set of *strings* $T = T[1]\dots T[n]$ (also called *solid* strings) of length $|T| = n$ over Σ, and by Σ^* the set of all possible strings of any length, including the *empty string* ε. By T_1T_2 or $T_1 \cdot T_2$ we denote the *concatenation* of two strings T_1 and T_2, i.e. $T_1T_2 = T_1[1]\dots T_1[|T_1|]T_2[1]\dots T_2[|T_2|]$. For any string T and integer $k > 0$, T^k denotes the concatenation of k copies of T. For any $1 \leqslant i \leqslant j \leqslant n$, $T[i..j]$ is the *fragment* of T starting at position i and ending at position j. The fragment $T[i..j]$ is an *occurrence* of the underlying *substring* $P = T[i]\dots T[j]$; we say that P occurs at *position* i in T, and that P is a *factor* of T. A *prefix* of T is a fragment of the form $T[1..j]$ and a *suffix* of T is a fragment of the form $T[i..n]$. Given a string T over Σ, we call *suffix tree* of T the compacted trie of all suffixes of $T \cdot \$$, where $\$ \notin \Sigma$. The suffix tree of a string $T[1..n]$ occupies $\mathcal{O}(n)$ space and it can be constructed in $\mathcal{O}(n)$ time for integer alphabets of size $|\Sigma| = \mathcal{O}(n^{\mathcal{O}(1)})$ [119, 258]. Unless specified otherwise, in this chapter we assume that $\Sigma = [0, \sigma]$ with $\sigma = n^{\mathcal{O}(1)}$; note that, without this assumption, the time complexity of some of our algorithms is increased by logarithmic factors.

Pattern \ Text	1-D	k-D	GD,ED
1-D	[153] $\mathcal{O}(n + n \log m)$	**Thm. A.2.5** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.13** $\Omega((MN)^{1-\epsilon})$
k-D	**Thm. A.2.6** $\Omega((MN)^{1-\epsilon})$	**Thm. A.2.7** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.14** $\Omega((MN)^{1-\epsilon})$
GD,ED	**Cor. A.2.12** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.14** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.14** $\Omega((MN)^{1-\epsilon})$

(a) Degenerate pattern, degenerate text

Pattern \ Text	1-F	k-F	F,EF
1-D	**Thm. A.2.8** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.15** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.15** $\Omega((MN)^{1-\epsilon})$
k-D	**Cor. A.2.15** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.15** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.15** $\Omega((MN)^{1-\epsilon})$
GD,ED	**Cor. A.2.15** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.15** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.15** $\Omega((MN)^{1-\epsilon})$

(b) Degenerate pattern, founder text

Pattern \ Text	1-D	k-D	GD,ED
1-F	**Thm. A.2.9** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.16** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.16** $\Omega((MN)^{1-\epsilon})$
k-F	**Cor. A.2.16** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.16** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.16** $\Omega((MN)^{1-\epsilon})$
F,EF	**Cor. A.2.16** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.16** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.16** $\Omega((MN)^{1-\epsilon})$

(c) Founder pattern, degenerate text

Pattern \ Text	1-F	k-F	F,EF
1-F	**Thm. A.2.10** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.17** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.17** $\Omega((MN)^{1-\epsilon})$
k-F	**Cor. A.2.17** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.17** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.17** $\Omega((MN)^{1-\epsilon})$
F,EF	**Cor. A.2.17** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.17** $\Omega((MN)^{1-\epsilon})$	**Cor. A.2.17** $\Omega((MN)^{1-\epsilon})$

(d) Founder pattern, founder text

Table 14. Complexity chart for problem MATCH(X, Y), where X ranges over rows and Y over columns of each table. The yellow cell in the upper-left corner indicates that the upper bound holds only for constant-size alphabets, as it has an exponential dependency on the alphabet size. Red cells denote a quadratic lower bound $\Omega((MN)^{1-\epsilon})$ (for every $\epsilon > 0$) conditioned on SETH, which holds even when the alphabet is of constant size and the length of the variants in each segment is $\mathcal{O}(1)$ (our bounds are even tighter, check the referred results for details). Orange cells denote that the quadratic lower bound only holds for variable strings in which the length of the variants in the segments is $\omega(\log n)$: it is an open problem whether these lower bounds also hold when the length of the variants is $\mathcal{O}(\log n)$. Letters in the time complexities are as in Table 13. All the results in this table are novel, except for the subquadratic upper bound for MATCH(1-D,1-D) that was already known.

An *elastic-degenerate* (ED) *string* T over an alphabet Σ is a sequence $T = T[1] \cdots T[n]$ of n finite sets, called *segments*, where each $T[i]$ is a subset of Σ^*; we call $|T| = n$ the *length* of T. The *size* $|T| = N$ of T is the total number of characters in T, i.e. $N = N_\varepsilon + \sum_{i=1}^{n} \sum_{S \in T[i]} |S|$, where N_ε is the total number of empty strings in the segments of T; the *cardinality* B of T is the total number of strings in all segments, i.e. $B = \sum_{i=1}^{n} |T[i]|$. We call the set $\mathcal{V}$ of distinct strings that appear in at least one segment of T its *vocabulary*. We also denote by N_i and B_i the size and the cardinality of segment $T[i]$, respectively; finally, for any $1 \leqslant i \leqslant j \leqslant n$, $T[i..j]$ denotes the fragment $T[i] \cdots T[j]$ between segments i and j of T.

If for every i the strings in $T[i]$ have all the same length k_i (called the *width* of $T[i]$), we say that T is a *generalised degenerate* (GD) string. If in addition all segments $T[i]$ have the same width k, T is a k-*degenerate* string (k-D, in short). In the special case $k = 1$, T is known in the literature as a *degenerate* or *indeterminate* string. Finally, if for every i it holds that $B_i = 1$, then we have a solid string. A schematic representation of these classes is in Figure 34.

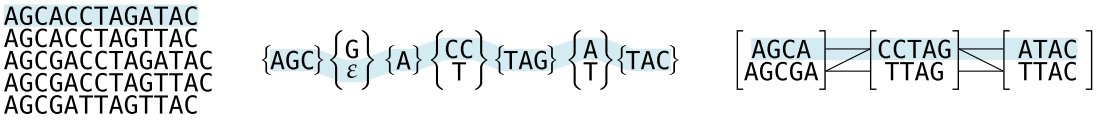


Figure 35. An ED string (center) and an EF graph (right) built from the same set of strings (left). Note that the strings on the left are a subset of the languages of the ED string and the EF graph.

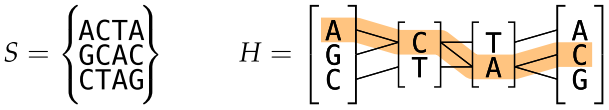


Figure 36. A k-D string S and the 1-F graph H obtained applying the transformation of Remark A.2.1. $\mathcal{L}(S)$ consists of the three length-4 strings in its only segment; $\mathcal{L}(H)$ consists of the 8 length-4 strings ACTA, ACAC, ACAG, GCTA, GCAC, GCAG, CTAC, CTAG, thus it contains 5 spurious strings with respect to $\mathcal{L}(S)$, including the highlighted one.

An *elastic founder* (EF) *graph* is a pair $G = (T, E)$, where T is an ED string s.t. the empty string ε is not an element of any of its segments[1], and $E = \bigcup_{i=1}^{n-1} E_i$, where E_i is the set of edges from $T[i]$ to $T[i+1]$, which can be identified with a subset of the Cartesian product $[1, B_i] \times [1, B_{i+1}]$. G is a *founder graph*, F graph in short (resp. a k-*founder* graph, k-F in short) if T is a GD (resp a k-D) string. $G[i..j] = (T[i..j], \bigcup_{\ell=i}^{j-1} E_\ell)$ is the fragment of G between $T[i]$ and $T[j]$. The size of G is $N + |E|$, i.e. the sum of the size of the underlying ED string T and the total number of edges of G. Figure 35 shows an ED string of length $n = 7$ and size $N = 17$, and an EF graph with $n = 3$, $N = 26$, $|E| = 6$, and hence total size 32.

The language $\mathcal{L}(T)$ of an ED string T consists of all the strings that can be obtained by concatenating one string per segment, maintaining their order: $\mathcal{L}(T) = \{S_1 \cdots S_n : S_i \in T[i] \ \forall\, i \in [1, n]\}$. The language of an EF graph G is defined analogously, but only strings that are connected by an edge can be concatenated: $\mathcal{L}(G) = \{S_1 \cdots S_n : S_i \in T[i] \ \forall\, i \in [1, n]$ and $(S_i, S_{i+1}) \in E_i \ \forall\, i \in [1, n-1]\}$. We remark that if T is an ED string (resp. EF graph), then in general $\mathcal{L}(T)$ may contain strings of different sizes, whereas if T is any among 1-D, 1-F, k-D, k-F, GD, F, then all strings in $\mathcal{L}(T)$ must have the same size.

For example, the language of the EF graph of Figure 35 consists of the 5 strings shown on the left; the language of the ED string includes in addition the strings {AGCATTAGATAC, AGCATTAGTAC, AGCGATTAGATAC}, thus consists of 8 strings in total.

Remark A.2.1. A common logical pitfall is to believe that it is possible to transform all non-elastic variable strings (GD strings, F graphs, k-D strings and k-F graphs) to equivalent 1-F graphs by splitting each segment into multiple segments (one containing the first letter of each string in the original segment, one containing the second letter, and so on) and adding appropriate edges to preserve letter sequences. However, the language of the 1-F graph obtained with this procedure is generally larger than that of the original variable string. Figure 36 exemplifies this issue when trying to transform a k-D string into a 1-F graph.

1 We keep this distinction from ED strings to match the existing literature.

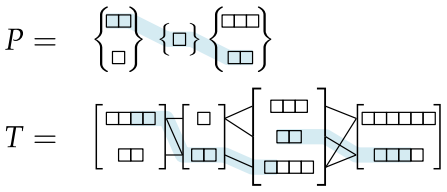


Figure 37. Representation of two possible occurrences of an ED string P into an EF graph T.

Recall that by *variable strings* we collectively refer to ED strings, EF graphs, and any of the special cases introduced above. In general, we say that there is an occurrence of a variable pattern P in a variable text T ending at $T[j]$ if there exists $1 \leqslant i \leqslant j$ such that a string in the language of P occurs as a substring of a string $t = t_i \cdot t_{i+1} \cdots t_j \in \mathcal{L}(T[i..j])$, with $t_k \in T[k]\ \forall\ k \in [i, j]$, ending within the last occurrence of t_j in t. In Figure 35, the solid pattern CTAGA occurs in ED ending at position 6, and in EF ending at position 3. Figure 37 visually represents two occurrences of an ED pattern in an EF text. We provide more technical definitions of variable pattern matching, specialised for the different cases studied in this chapter, in Sections A.3 and A.4.

PATTERN MATCHING ON VARIABLE TEXT: MATCH(X, Y)
Input: A solid or variable pattern P of type X and a variable text T of type Y
Output: All positions j such that a string in $\mathcal{L}(P)$ occurs in T ending at $T[j]$

Strings X and Y can be any of the types of variable strings mentioned above. Following the existing literature [143], the output of MATCH only specifies the ending segments of the occurrences, giving no information about the specific positions within the segments. In particular, if multiple occurrences of the pattern end within the same text segment, MATCH reports the segment only once. This convention is justified by the fact that the size of the language of a variable string, and hence the number of occurrences of a pattern, can be exponential in the input size. We say that MATCH has *constant alphabet* if the alphabet Σ of both the pattern and the text is of constant size.

In the rest of this section, we summarize the results that we will prove in Sections A.3 and A.4, consisting of upper and lower bounds for the problem MATCH(X, Y) for several choices of pattern type X and text type Y. We recall that we denote by m and M, respectively, the length and the size of the pattern; by n and N, respectively, the length and the size of the text; and by E the edges of an (elastic) founder graph text.

UPPER BOUNDS In Section A.4, we give two truly sub-quadratic algorithms for MATCH(X, Y) when the pattern is solid and the text is either a k-D string or a k-F graph.

Theorem A.2.1. MATCH(SOLID,k-D) *can be solved in* $\mathcal{O}(N + kn \log^2(\frac{m}{k})) = \mathcal{O}(N + N \log^2 m)$ *time.*

Theorem A.2.2. MATCH(SOLID,k-F) *can be solved in* $\mathcal{O}(\sqrt{m}(|E| + N + n \log^2 m))$ *time.*

In Section A.3, we consider the cases where the pattern is solid and the text is either a GD string or an F graph. In these cases, a mn term appears in the time

complexity of the algorithms we propose. Consequently, these algorithms are subquadratic only when $N/n = \omega(1)$, and it is an open problem whether strongly subquadratic algorithms exist when $N/n = \mathcal{O}(1)$.

Theorem A.2.3. MATCH(SOLID,GD) *can be solved in* $\mathcal{O}(mn + N)$ *time.*

Theorem A.2.4. MATCH(SOLID,F) *can be solved in* $\mathcal{O}(mn + N + |E|)$ *time.*

LOWER BOUNDS When both the pattern and the text are variable strings, we show conditional lower bounds for MATCH(X, Y). The quadratic conditional lower bound MATCH(1-D, 1-D) given in [153], which we discuss in more detail in Section A.5, holds only for non-constant alphabets, while a truly subquadratic algorithm exists for constant-size alphabets. All the other lower bounds we present hold even for constant-size alphabets. However, the reductions underlying Theorems A.2.5 and A.2.6 and Corollaries A.2.12 and A.2.13 only work when the length k of the strings in every segment is $\omega(\log n)$, where n is the length of the text, thus they do not rule out the existence of subquadratic-time algorithm when $k = \mathcal{O}(\log n)$. This is in contrast with the rest of the reductions, which all apply even when $k = \mathcal{O}(1)$ and thus are, in this sense, *stronger*. In particular, Theorem A.2.7 proves that MATCH(k-D,k-D) requires at least quadratic time even when $k = 2$. The conditional lower bounds we prove rely on a famous conjecture: the *Orthogonal Vectors Hypothesis* (OVH), which is, in turn, implied by the Strong Exponential Time Hypothesis (SETH) [157, 259]. See Conjecture A.5.1 in Section A.5.

Theorem A.2.5. *No algorithm can solve* MATCH(1-D,k-D) *on constant alphabet for every* k *in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Theorem A.2.6. *No algorithm can solve* MATCH(k-D,1-D) *on constant alphabet for every* k *in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Theorem A.2.7. *No algorithm can solve* MATCH(2-D,2-D) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Theorem A.2.8. *No algorithm can solve* MATCH(1-D,1-F) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Theorem A.2.9. *No algorithm can solve* MATCH(1-F,1-D) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Theorem A.2.10. *No algorithm can solve* MATCH(1-F,1-F) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Theorem A.2.11. *No algorithm can solve* MATCH(SOLID,ED) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, for any* ED *string such that the difference in the lengths of the strings in any segment is at most* 1*, and the length of all such strings is at most* 4*, unless OVH is false.*

Since a pattern of type 1-D is a special case of k-D, GD and ED (and k-D is a special case of GD and ED), and since 1-F is a special case of k-F, F, and EF, the lower bounds above propagate along the taxonomies for P and/or T. Therefore, we have the following corollaries. Corollaries A.2.12 and A.2.13 directly follow from Theorems A.2.5 and A.2.6, thus only hold for the general cases where the maximum length of the strings in the segments is $\omega(\log n)$; in contrast, Corollary A.2.14 follows from Theorem A.2.7, thus it applies even to the restricted case where the maximum length of any string in any segment is 2.

Corollary A.2.12. *No algorithm can solve* MATCH(1-D,Y) *for* Y = k-D, GD, ED *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless OVH is false.*

Corollary A.2.13. *No algorithm can solve* MATCH(X,1-D) *for* X = k-D, GD, ED *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless OVH is false.*

Corollary A.2.14. *No algorithm can solve* MATCH(X,Y) *for* X = k-D, GD, ED *and* Y = k-D, GD, ED *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless OVH is false.*

Corollaries A.2.15, A.2.16 and A.2.17 below follow from Theorems A.2.8, A.2.9 and A.2.10, respectively.

Corollary A.2.15. *No algorithm can solve* MATCH(X,Y) *for* X = 1-D, k-D, GD, ED *and* Y = 1-F, k-F, F, EF *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless OVH is false.*

Corollary A.2.16. *No algorithm can solve* MATCH(X,Y) *for* X = 1-F, k-F, F, EF *and* Y = 1-D, k-D, GD, ED *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless OVH is false.*

Corollary A.2.17. *No algorithm can solve* MATCH(X,Y) *for* X = 1-F, k-F, F, EF *and* Y = 1-F, k-F, F, EF *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless OVH is false.*

A.3 MATCHING A SOLID PATTERN IN A GD OR F TEXT

We start by formally defining an occurrence of a solid pattern in a GD string or an F graph. Let $T^{\ell,r}[i..j]$ be obtained from $T[i..j]$ by removing a prefix of length ℓ from all the strings of $T[i]$ and a suffix of length r from all the strings of $T[j]$, for some $\ell < k_i$ and $r < k_j$ (an example is in Figure 38).

Definition A.3.1. Let T be a GD string or an F graph. A solid pattern P has an occurrence ending at position j in T if $\exists i \in [1, j]$ s.t. there exist $\ell \in [0, k_i - 1]$ and $r \in [0, k_j - 1]$ such that $P \in \mathcal{L}(T^{\ell,r}[i..j])$. We call ℓ and $k_j - r$ the starting and ending *offset* of the occurrence, respectively.

In Theorem A.2.3, we show that the $\mathcal{O}(m^2n + N)$-time algorithm proposed in [143] for matching a solid pattern of length m in an ED text of length n and size N requires only $\mathcal{O}(mn + N)$ time when applied to GD texts.

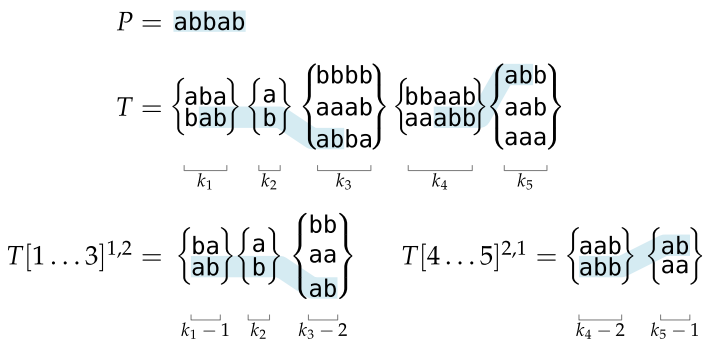


Figure 38. Two occurrences of a solid pattern in a GD string. The starting offset of the leftmost occurrence is 1, and its ending offset is $k_3 - 2 = 2$; the starting and ending offsets of the second occurrence are 2 and $k_5 - 1 = 2$, respectively.

Theorem A.2.3. MATCH(SOLID,GD) *can be solved in* $\mathcal{O}(mn + N)$ *time.*

Proof. Consider the pattern-matching algorithm of [143] for ED texts. The algorithm first constructs the suffix tree $\mathcal{T}_P$ of the pattern P in a preprocessing step; then it reads the text T segment by segment, from left to right. For any segment $T[i]$ of width k_i, the algorithm solves the following 4 sub-problems: (i) If $k_i \geqslant m$, does P occur in a string from $T[i]$? (*easy case*) (ii) Compute every suffix of a string $t \in T[i]$ that matches a prefix of P (*suffix case*); (iii) Compute every prefix of a string $t \in T[i]$ that matches a suffix of P (*prefix case*); (iv) If $k_i < m$, find the strings $t \in T[i]$ that are factors of P (*anchor case*). All occurrences of P are then computed from the solutions to these four problems. The easy and the suffix/prefix case can be solved both for ED and GD texts in $\mathcal{O}(N + m)$ time by using any standard pattern-matching algorithm (e.g. KMP [167]) and $\mathcal{T}_P$, respectively (see [143] for details).

We now focus on the anchor case for a segment $T[i]$ and show that, in the case of GD texts, it can be solved in time $\mathcal{O}(N_i + m)$ by solving an instance of the *Active Prefixes* (AP) problem [46]. AP applied to segment $T[i]$ takes as input a bit-vector U of length m s.t. $U[j] = 1$ iff the prefix $P[1..j]$ matches a suffix of a string from $\mathcal{L}(T[1..i-1])$; it outputs a bit-vector U of size m such that $U[j + k_i] = 1$ iff $U[j] = 1$ and $P[j + 1..j + k_i] \in T[i]$ (in other words, some string from $T[i]$ occurs in P starting at position $j + 1$: see Example A.3.1). To solve AP it thus suffices to compute all the occurrences of all the strings of $T[i]$ in P and check whether they extend some active prefixes stored in U, namely, if they start at some position $j + 1$ with $U[j] = 1$. The strings in $T[i]$ are all of the same length k_i, thus no two of them can occur at the same position in P. We have the following crucial observation.

Observation A.3.1. The cumulative number of occurrences in P of all the strings from a GD segment $T[i]$ is bounded by m.

Observation A.3.1 implies that all such occurrences can be computed and stored in an auxiliary bit-vector OCC_i of size m in $\mathcal{O}(N_i + m)$ time, using $\mathcal{T}_P$. U can then be obtained by left-shifting OCC_i by one position, taking its bit-wise AND with U, and shifting the result vector by k_i positions to the right. Solving the anchor case for every segment thus takes $\mathcal{O}(N + mn)$ time. Note that the difference in the time complexities of AP for GD texts and ED texts is because, in the latter case, Observation A.3.1 does

not hold, since the number of occurrences of the strings from $T[i]$ in P can only be bounded by m^2. □

Example A.3.1. Consider T and P as in Figure 38. The input of AP applied to $T[2]$ is $U = [1, 1, 0, 0, 0]$; the output is $V = [0, 1, 1, 0, 0]$. This is because the prefix of length 1 of P, which is a suffix of the first string of $T[1]$, and the prefix of length 2 of P, which is a suffix of the second string of $T[1]$, can be extended by string b of $T[2]$.

We next adapt the algorithm of Theorem A.2.3 to solve pattern matching on F graphs.

Theorem A.2.4. MATCH(SOLID,F) *can be solved in* $\mathcal{O}(mn + N + |E|)$ *time.*

Proof. Let $G = (T, E)$ be a founder graph of length n, size N and cardinality B, and let P be a solid pattern of length m. We denote by $T[i][j]$ the j-th string of $T[i]$, $j \in [1, B_i]$ (recall that B_i is the cardinality of segment i, and their sum over all i gives the total number B of strings in the segments of T), assuming any fixed order; an edge $(j, j') \in E_{i-1}$ thus connects string $T[i-1][j]$ to $T[i][j']$. We process separately the occurrences of P that span only one segment (easy case), only two segments, or more. The easy case is trivially solved in $\mathcal{O}(m + N)$ time using any linear-time pattern-matching algorithm. Let us now focus on the other two cases.

Analogously to the suffix and prefix subproblems listed in the proof of Theorem A.2.3 we precompute, for each string in each segment, the length of all suffixes that are equal to some prefix of P, and of all prefixes that are equal to some suffix of P. While in the case of GD text, it suffices to store the lengths of all such overlaps cumulatively for each segment, in the case of founder graphs, due to the presence of edges, we need to retain this information separately for each string in each segment. We thus compute two binary arrays $b_{i,j}$ and $e_{i,j}$ for each string $T[i][j]$, each of the same length k_i, s.t. $b_{i,j}[\ell] = 1$ if and only if the suffix of length ℓ of $T[i][j]$ is equal to a prefix of P, and $e_{i,j}[\ell] = 1$ if and only if the prefix of length ℓ of $T[i][j]$ is equal to a suffix of P. All such arrays can be constructed in $\mathcal{O}(m + N)$ total time using e.g. the suffix trees of P and of its reversal, and occupy total space $\mathcal{O}(N)$. See Figure 39.

Occurrences spanning only two segments. We consider all pairs of consecutive segments $T[i], T[i+1]$ such that $k_i + k_{i+1} \geqslant m$ (the only candidates for occurrences of this kind). For each edge $(j, j') \in E_i$, let p_j be the length of the longest suffix of $T[i][j]$ that overlaps a prefix of P, i.e. the largest index of $b_{i,j}$ set to 1, and let $s_{j'}$ be the length of the longest prefix of $T[i+1][j']$ that overlaps a suffix of P, i.e. the largest index of $e_{i+1,j'}$ set to 1. The following observation characterises the occurrences of P spanning $T[i], T[i+1]$ (see also Example A.3.2).

Observation A.3.2. P has an occurrence spanning $T[i], T[i+1]$ iff there exists an edge $(j, j') \in E_i$ such that P occurs in the concatenation of the suffix of length p_j of $T[i][j]$ and the prefix of length $s_{j'}$ of $T[i+1][j']$, which are equal, respectively, to $P[1\,..\,p_j]$ and $P[m - s_{j'} + 1\,..\,m]$.

The main tool we use to spot these occurrences is a data structure to efficiently answer the following queries: given any pair of positions $1 \leqslant s \leqslant p \leqslant m$, return the set occ of occurrences of P in the concatenation $P[1\,..\,p]P[s\,..\,m]$ of its prefix of length p and suffix of length $m - s + 1$. A classical data structure to answer these queries is the *border tree* [144]; recently, Pissis [229] introduced a data structure alternative

to border trees that can be constructed in $\mathcal{O}(m/\log_\sigma m)$ time and answers queries in $\mathcal{O}(1+|occ|)$ time. After constructing the data structure of [229] for P, we thus process each pair $T[i], T[i+1]$ such that $k_i + k_{i+1} \geqslant m$, and apply the following algorithm, whose correctness follows from Observation A.3.2: for each $(j, j') \in E_i$, query the border data structure with indices $p_j, m - s_{j'} + 1$. If the returned set occ is nonempty, break and return an occurrence of P in G ending at position $i+1$. Each such query takes $\mathcal{O}(1+|occ|)$ time [229], and $|occ| \leqslant m$. Since we stop asking queries as soon as we find a nonempty set of occurrences, the total time for $T[i], T[i+1]$ is $\mathcal{O}(|E_i| + m)$, implying time $\mathcal{O}(|E| + mn)$ to process the whole G.

Occurrences spanning at least three segments. This case is analogous to the anchor case of Theorem A.2.3, and can only happen when the second of the (at least three) segments has a width smaller than m. We process the segments of G from left to right and maintain an array V of size m that keeps track of the prefixes of P that match up to a certain segment (*partial occurrences*): however, now V is an array of integers, rather than a simple bit-vector, and it only keeps track of partial occurrences that span at least one full segment (thus excluding prefixes of P that match a proper suffix of some string from some segment: these shorter partial occurrences will be treated differently, as we explain in the following).

Let V_{i-1} denote the state of V after processing a segment $T[i-1]$ ($V_{i-1} = 0^m$ if $k_{i-1} > m$ or $i = 1$). When processing segment $T[i]$ (assuming $k_i < m$), our task is to compute the next state V_i s.t. $V_i[\ell] = j$ iff $P[1..\ell]$ is a suffix of some string from $\mathcal{L}(G[1..i])$ that ends with the whole string $T[i][j]$, and $V_i[\ell] = 0$ otherwise. Note that the values of V_i are uniquely defined: see Observation A.3.3. In particular, this implies that the first $k_i - 1$ positions of V_i, corresponding to partial occurrences that do not contain an entire string from $T[i]$, are always 0. Further, note that positions between k_i and $\min\{k_i + k_{i-1} - 1, m\}$ of V_i correspond to partial occurrences of P that contain an entire string from $T[i]$ but not from $T[i-1]$. We compute $V_i[k_i .. k_i + k_{i-1} - 1]$ and $V_i[k_i + k_{i-1} .. m]$ using two different procedures (the second sub-array is empty in the case $k_i + k_{i-1} > m$).

To process $T[i]$, we first compute an array OCC_i of size m such that $OCC_i[\ell] = j$ iff $T[i][j]$ occurs in P starting at position ℓ; by Observation A.3.1, OCC_i is well defined and can be computed in $\mathcal{O}(N_i + m)$ time using the suffix tree $\mathcal{T}_P$. Note that OCC_i contains all potential extensions of partial occurrences of P *with whole strings* from $T[i]$. We use the following crucial observation, which is a direct consequence of Observation A.3.1.

Observation A.3.3. Any partial occurrence of P ending at $T[i-1]$ can be extended by at most one occurrence of one string from $T[i]$.

In particular, the partial occurrence represented by $V_{i-1}[\ell] = j$ can only be extended by the unique string from $T[i]$ occurring in P at position $\ell + 1$, if any. This implies that it suffices to check the following necessary and sufficient conditions to compute $V_i[k_i + k_{i-1} .. m]$: for each position $\ell \in [k_i + k_{i-1}, m]$, we set $V_i[\ell] = j'$ iff $OCC_i[\ell - k_i + 1] = j'$, $V_{i-1}[\ell - k_i] = j$ and $(j, j') \in E_{i-1}$. To verify these conditions, collect all triplets (j, j', ℓ) such that $V_{i-1}[\ell - k_i] = j$ and $OCC_i[\ell - k_i + 1] = j'$, for all $\ell \in [k_i + k_{i-1}, m]$, and sort them lexicographically together with all the pairs from E_i, using radix sort. Then, scan the resulting sorted list and set $V_i[\ell] = j'$ iff a triplet (j, j', ℓ) is immediately preceded by the pair (j, j'). Since there is at most one

triplet for each value of ℓ, this requires $\mathcal{O}(m + |E_i|)$ total time per segment and thus $\mathcal{O}(mn + |E|)$ time over the whole G.

We set $V_i[k_i] = j$ if and only if $T[i][j]$ occurs in P starting at position 1. Let us now focus on computing $V_i[k_i + 1 \,..\, k_i + k_{i-1} - 1]$. This portion of V_i corresponds to partial occurrences of P matching a proper suffix of some string from $T[i-1]$ and extended with an occurrence of some string from $T[i]$ stored in $OCC_i[2 \,..\, k_{i-1}]$. Note that we cannot use the same technique as for $V_i[k_i + k_{i-1} \,..\, m]$, because two strings from $T[i-1]$ can have equal suffixes. Instead, we scan $OCC_i[2 \,..\, k_{i-1}]$: if $OCC_i[\ell] = j' \neq 0$, we check, for all $(j, j') \in E_{i-1}$, whether $b_{i-1,j}[\ell - 1] = 1$, and set $V_i[\ell + k_i - 1] = j'$ if this is the case. In other words, we check, for each occurrence of $T[i][j']$ starting at $P[\ell]$, whether the suffix of length $\ell - 1$ of some of the strings from $T[i-1]$ connected to $T[i][j']$ is equal to $P[1 \,..\, \ell - 1]$. See also Example A.3.3. This procedure has a total cost $\mathcal{O}(N_{i-1} + m)$ because each position of each $b_{i-1,j}$ is accessed at most once; and each time we read an edge, we access exactly one position of one array b, thus we read at most N_{i-1} edges. This implies a total time $\mathcal{O}(N + mn)$ over all segments.

Finally, to check whether some partial occurrence represented by $V_{i-1}[\ell] = j$ for $\ell \in [m - k_i + 2 \,..\, m]$ can be extended to a full occurrence of P with a proper prefix of some string from $T[i]$, it suffices to check, for each edge $(j, j') \in E_{i-1}$, whether $e_{i,j'}[m - \ell + 1] = 1$. This requires $\mathcal{O}(N + mn)$ total time because each position of each array e is accessed at most once. We obtain a total time complexity of $\mathcal{O}(N + mn + |E|)$. □

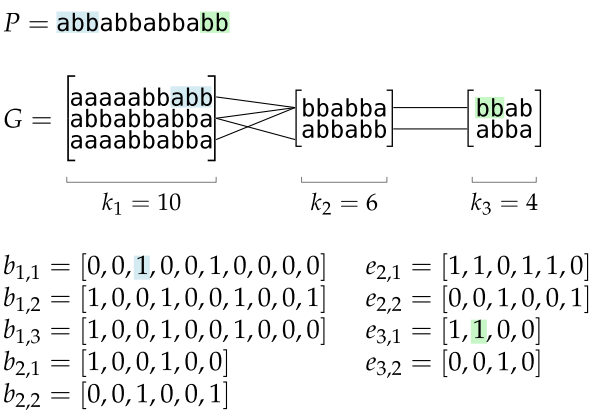


Figure 39. Arrays $b_{i,j}$ and $e_{i,j}$ (used in the proof of Theorem A.2.4) for an F graph of length 3 and a solid pattern of length 12. Highlighted in blue is a suffix/prefix overlap of length 3 and the corresponding entry of $b_{1,1}$; in green, a prefix/suffix overlap and the corresponding entry of $e_{3,1}$. Arrays $e_{1,\cdot}$ and $b_{3,\cdot}$ are not shown as they may not start or end any occurrence of P.

Example A.3.2. Consider the instance of Figure 39. To find occurrences of P spanning the first two segments only, we consider one edge (j, j') at a time, compute the lengths p_j and $s_{j'}$ from $b_{1,j}$, and $e_{2,j'}$ and check whether P occurs in the concatenation of $P[1 \,..\, p_j]$ and $P[m - s_{j'} + 1 \,..\, m]$, as described in the proof of Theorem A.2.4. Consider the edge $(1, 1)$, connecting $T[1][1]$ and $T[2][1]$. We have $p_1 = 6$ (as this is the largest index of $b_{1,1}$ set to 1, i.e. the length of the longest prefix of P which is also a suffix of $T[1][1]$) and $s_1 = 5$ (the largest index of $e_{2,1}$ set to 1). P does not occur in

the concatenation of $P[1..6] = \texttt{abbabb}$ and $P[8..12] = \texttt{bbabb}$, thus no occurrence of P uses edge $(1,1)$.

Now consider edge $(2,1)$. The longest suffix of $T[1][2]$ that is also a prefix of P is of length $p_2 = 10$; and the longest suffix of P that is also a prefix of $T[2][1]$ has length $s_1 = 5$. P occurs in the concatenation of $P[1..10]$ and $P[8..12]$, i.e. $\texttt{abbabbabba} \cdot \texttt{bbabb}$, thus an occurrence of P ending at segment $T[2]$ is reported. The algorithm will not examine the rest of the edges; therefore, although the pattern also occurs in the concatenation $P[1..7] \cdot P[8..12] = \texttt{abbabba} \cdot \texttt{bbabb}$, corresponding to edge $(3,1)$, this extra occurrence will not count towards the complexity of the algorithm.

Example A.3.3. Consider again the instance in Figure 39, and focus on the occurrences of P spanning the three segments, which are computed as described in the proof of Theorem A.2.4. We have $V_1 = [0,0,0,0,0,0,0,0,0,2,0,0]$ as the whole string $T[1][2]$ matches the prefix of length 10 of P. We know that the first $k_2 - 1 = 5$ positions of V_2 are 0, as they correspond to (possible) partial occurrences of P that do not contain a whole string from $T[2]$. We have $OCC_2 = [2,1,0,2,1,0,2,0,0,0,0,0]$ because $T[2][1]$ occurs in P starting at positions 2 and 5 and $T[2][2]$ occurs in P starting at positions 1, 4 and 7. Since $k_1 + k_2 = 15 > m = 12$, no occurrence of P can contain an occurrence of a whole string from $T[1]$ *and* from $T[2]$. Since $OCC_2[1] = 2$, we set $V_2[6] = 2$. To compute $V_2[7..12]$, we scan $OCC_2[2..10]$ from left to right. Reading $OCC_2[2] = 1$, we need to check the edges incoming to the first string of $T[2]$ – in this case, edges $(1,1)$, $(2,1)$ and $(3,1)$ – and access the first position of the corresponding array b. For edge $(1,1)$, we have $b_{1,1}[1] = 0$, thus we do not update any value of V_2; for edge $(2,1)$, we have $b_{1,2}[1] = 1$, thus we set $V_2[7] = 1$ to indicate that there is a partial match of $P[1..7]$ ending with an occurrence of the whole string $T[2][1]$. The same information is also captured by the edge $(3,1)$ because $b_{1,3}[1] = 1$. $OCC_2[3] = 0$ does not trigger any update of V_2; reading $OCC_2[4] = 2$, we check the only edge incoming to $T[2][2]$, which is $(2,2)$: since $b_{1,2}[3] = 0$, no value of V_2 is updated. Reading $OCC_2[5] = 1$, we check again edges $(1,1)$, $(2,1)$ and $(3,1)$, find out that $b_{1,2}[4] = 1$, and thus set $V_2[10] = 1$. The rest of the positions of OCC_2 do not cause any further update of V_2, which at the end of this procedure is $V_2 = [0,0,0,0,0,2,1,0,0,1,0,0]$.

A.4 MATCHING A SOLID PATTERN IN A k-D OR k-F TEXT

In this section, we investigate the complexity of the pattern-matching problem in cases where the pattern is solid and the text is either a k-D string or a k-F graph.

The problem of finding all the occurrences of a solid pattern of length m in a 1-D string of length $n > m$ and size N can be seen as an instance of the *Subset Matching* problem, defined by Cole and Hariharan [80]. This problem asks, given a 1-D text and a 1-D pattern, to find all the locations in the text where each segment of the pattern is a subset of the corresponding segment of the text; an $\mathcal{O}(n \log^2 m)$-time deterministic algorithm exists [81][2]. In this section, we leverage this result to prove sub-quadratic upper bounds for MATCH(SOLID,1-F), MATCH(SOLID,k-D)

2 The upper bound explicitly proved by the authors in the cited paper is $\mathcal{O}(N \log^2 N)$; however, some observations made by the same authors in [80, Section 4] that lead to better bounds apply, and indeed the authors state that Subset Matching can be solved deterministically in $\mathcal{O}(n \log^2 m)$ time both in the abstract of [81] and in their later work [82].

and MATCH(SOLID,k-F) by reducing each of these problems to several instances of MATCH(SOLID,1-D).

Theorem A.2.1. MATCH(SOLID,k-D) *can be solved in* $\mathcal{O}(N + kn\log^2(\frac{m}{k})) = \mathcal{O}(N + N\log^2 m)$ *time.*

Proof. Let P be a solid pattern of length m and T a k-D string of length n, cardinality B and total size N. In a preprocessing step, we compute all suffix/prefix and prefix/-suffix overlaps of P and each string in the vocabulary of T. More precisely, for each segment $T[i]$ we compute two binary arrays b_i and e_i of length k such that $b_i[j] = 1$ if and only if a suffix of length j of one of the strings in $T[i]$ is equal to a prefix of P, and $e_i[j] = 1$ if and only if a a prefix of length j of one of the strings in $T[i]$ is equal to a suffix of P. The arrays b_i and e_i occupy $\mathcal{O}(kn) = \mathcal{O}(N)$ words of space and can be computed in $\mathcal{O}(N)$ time by e.g. building the generalised suffix tree of all the strings in all segments of T.

Consider the case $m \geqslant 2k-1$, which implies that each occurrence of P contains at least 1 full string from some $T[i]$. Let P_k denote the set of length-k substrings of P and let $h(s)$ denote the lexicographic[3] rank of $s \in P_k$, to be used as a unique ID for s. The values $h(s)$ can be computed and stored in $\mathcal{O}(m)$ time and space by constructing the suffix tree of P and annotating the nodes at string depth k (possibly making them explicit) with their rank, obtained with a lexicographic traversal of the tree.

Using these values, we construct k instances of MATCH(SOLID,1-D), one per each possible starting offset of occurrences of P in T. The pattern $P^{(\ell)}$ of the ℓ-th instance, $\ell \in [0, k-1]$, is obtained from P by splitting it into consecutive non-overlapping fragments of length k: the first fragment starts at position $\ell+1$ and the possible remaining suffix of length $m-\ell-k\lfloor\frac{m-\ell}{k}\rfloor$ is discarded. Each of such fragments is then replaced by its ID. The length of $P^{(\ell)}$ is thus $\lfloor\frac{m-\ell}{k}\rfloor$, and the cumulative space occupied by all such instances for all $\ell \in [0, k-1]$ is $\mathcal{O}(m)$. The text $T^{(\ell)}$ of the ℓ-th instance is obtained from T by replacing each string $s \in T[i]$ by its ID, for all $i \in [1, n]$, as follows. If $s \in P_k$ and $h(s)$ occurs in $P^{(\ell)}$, then s is replaced by $h(s)$; otherwise, (thus if $h(s)$ does not occur in $P^{(\ell)}$ or s does not occur in P), s is discarded[4]. Each $T^{(\ell)}$ has length n and total size $\mathcal{O}(\min\{B, n\frac{m}{k}\})$, therefore they collectively occupy $\mathcal{O}(kB) = \mathcal{O}(N)$ space. See Figure 40 for an example of this construction.

The k patterns can be constructed all at the same time by traversing the suffix tree of P in $\mathcal{O}(m)$ total time. The k texts can be constructed simultaneously by using the suffix tree of P, further annotated as follows. At each node v at string depth k (which are all and only the nodes annotated with some ID $h(s)$), we store the list of values ℓ such that $\ell = i \mod k$ for some leaf i descending from v (that corresponds to the suffix starting at position i). Thus ℓ is in the list of v if and only if the length-k substring corresponding to v is one of the fragments P is split into to build $P^{(\ell)}$. Since these nodes partition the leaves, and the number of descending leaves bounds the size of each list, such lists collectively occupy $\mathcal{O}(m)$ space and can be computed in $\mathcal{O}(m)$ time with a single tree traversal. Armed with this annotated suffix tree, to construct the k texts we simply search each string of T in the tree and write the value

3 Note that any other total order would also work for this purpose.

4 Note that, applying this procedure, some of the segments of $T^{(\ell)}$ might be empty. To avoid so, one can use a special symbol \$ different from all the IDs $h(s)$ and place it in the otherwise empty segments of $T^{(\ell)}$.

$h(s)$ (if any) in the corresponding segment of all texts $T^{(\ell)}$ such that ℓ is in the list of the reached node. This procedure thus requires $\mathcal{O}(N)$ total time.

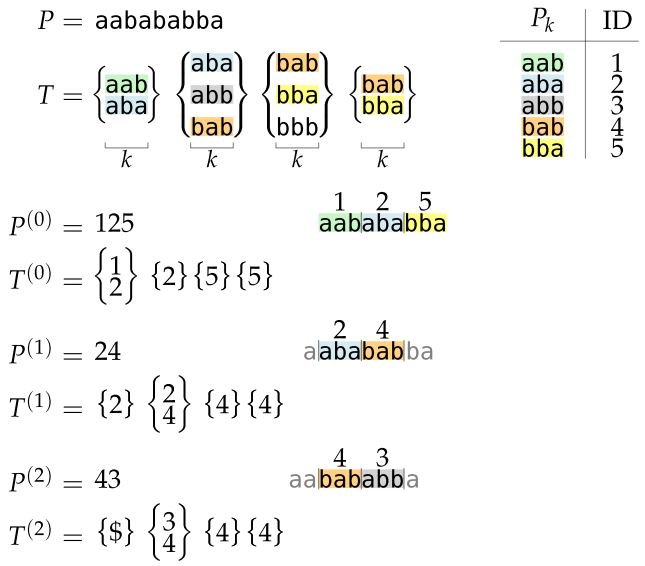


Figure 40. Illustration of the reduction described in the proof of Theorem A.2.1. Note that P occurs twice in T, ending at $T[3]$ with starting offset 0, and ending at $T[4]$ with starting offset 2. The first occurrence corresponds to an occurrence of $P^{(0)}$ in $T^{(0)}$ ending at $T^{(0)}[3]$. The second can be retrieved by the occurrence of $P^{(1)}$ in $T^{(1)}$ ending at $T^{(1)}[3]$ using the pre-computed arrays $b_1 = [1,0,1]$ and $e_4 = [0,1,0]$: indeed, $b_1[1] = 1$ and $e_4[2] = 1$, indicating that the suffix of length 1 of some string from $T[1]$ is equal to a prefix of P and a prefix of length 2 of some string from $T[4]$ is equal to a suffix of P, respectively, which complete the occurrence.

We now show that each occurrence of P in T that fully contains at least a string from a segment of T (which is always the case when $m \geqslant 2k-1$) corresponds to an occurrence of some $P^{(\ell)}$ in $T^{(\ell)}$ for some $\ell \in [0, k-1]$. We treat the other occurrences separately, as we will detail at the end of the proof. The key observation is that if an occurrence of P in T starts at offset $q = k-\ell+1$ in $T[i]$ and ends at offset r in $T[j]$, then the following hold: (i) a string from each of the segments $T[i+1], \ldots, T[j-1]$ occurs consecutively in P starting from position $\ell+1$; (ii) a prefix of length ℓ of P matches a suffix of some string in $T[i]$; and (iii) a suffix of length r matches a prefix of some string in $T[j]$. Observation (i) implies, by construction, an occurrence of $P^{(\ell)}$ in $T^{(\ell)}$ starting at position $i+1$; moreover, Observations (ii) and (iii) imply that $b_i[\ell] = e_j[r] = 1$. This gives us the following algorithm. For each possible offset $\ell = 0, 1, \ldots, k-1$:

1. find the occurrences of $P^{(\ell)}$ in $T^{(\ell)}$;
2. for each occurrence $T^{(\ell)}[i..j]$ (note that $j = i-1+\lfloor \frac{m-\ell}{k} \rfloor$), check if it corresponds to an occurrence of P in T by checking whether $b_{i-1}[\ell] = e_{j+1}[r] = 1$ and report an occurrence $T[i-1..j+1]$ if this condition holds.

Note that when $\ell = 0$ we only need to check whether $e_{j+1}[r] = 1$ and the corresponding occurrence is $T[i..j+1]$; and symmetrically, if $r = 0$, we only check if $b_{i-1}[\ell] = 1$, the occurrence being $T[i-1..j]$. For a fixed ℓ, Step (1) can be done in $\mathcal{O}(n \log^2(\frac{m}{k}))$

time using the algorithm from [81] for subset matching; and Step (2) requires $\mathcal{O}(1)$ time per occurrence. Since each $P^{(\ell)}$ can occur in at most n positions, the total time for step (2) over all $\ell = 0, 1, \ldots, k-1$ is $\mathcal{O}(kn) = \mathcal{O}(N)$; and the total time for Step (1) is $k \cdot \mathcal{O}(n \log^2(\frac{m}{k}))$.

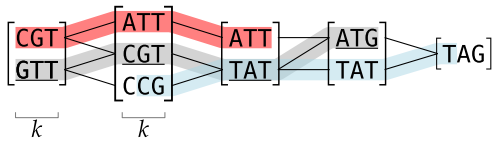
Finally, we can find all the occurrences of P in T that do not fully contain a string from some segment (which can only happen if $m < 2k-1$) in $\mathcal{O}(N)$ total time. These occurrences either (i) span exactly two consecutive segments of T, or (ii) are entirely contained in some string of some segment. To find all occurrences of type (ii) in $\mathcal{O}(N)$ time it suffices to run e.g. KMP [167]. To find the occurrences of type (i), we scan each array b_i: for each $j \in [1, k]$ s.t. $b_i[j] = 1$, we check whether $e_{i+1}[m-j] = 1$ and report an occurrence ending in segment $i+1$ if this is the case. This requires $\mathcal{O}(kn)$ total time for all $i \in [1, n-1]$. □

Let us now consider the case where the text is a k-F graph. Let P be a solid string of length m over an alphabet Σ, let ℓ be an integer which divides m, and let us write $P = P_1 \,..\, P_d$ where $|P_i| = \ell$ for $i = 1 \,..\, d$. We define the *ℓth spread* of P as the string $\mathbf{spr}^\ell(P) = \prod_{i=1}^{d-1} P_i \cdot \$ \cdot P_{i+1}$, where $\$ \notin \Sigma$. In other words, each fragment of length ℓ P_i is repeated twice, separated by a gadget letter \$, except for the first and last one.

Example A.4.1. Let $P = \texttt{GTTCGTTATATG}$. One has $\mathbf{spr}^3(P) = \texttt{GTT} \cdot \$ \cdot \texttt{CGT CGT} \cdot \$ \cdot \texttt{TAT TAT} \cdot \$ \cdot \texttt{ATG}$. Let $P' = \texttt{CGTATTATT}$. One has $\mathbf{spr}^3(P') = \texttt{CGT} \cdot \$ \cdot \texttt{ATT ATT} \cdot \$ \cdot \texttt{ATT}$.

Given a k-F graph $G = (T, \bigcup_{i=1}^{n} E_i)$ of length n, we define its *disentanglement* as the $(2k+1)$-D string $\mathcal{D}(G) = D_1 \,..\, D_{n-1}$ where for every $i = 1 \,..\, n-1$, the set D_i contains every string $t \cdot \$ \cdot t'$ such that $(t, t') \in E_i$. An example is shown in Figure 41.

$$P = \texttt{GTTCGTTATATG}$$
$$\mathbf{spr}^3(P) = \texttt{GTT\$CGTCGT\$TATTAT\$ATG}$$
$$P' = \texttt{CGTATTATT}$$
$$\mathbf{spr}^3(P') = \texttt{CGT\$ATTATT\$ATT} \quad \mathbf{spr}^3_1(P') = \texttt{CG\$TATTAT\$TATTAT\$T}$$



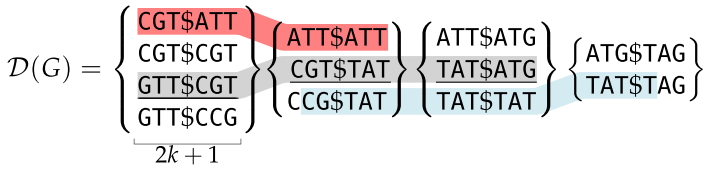


Figure 41. A k-F graph G (top) and its disentanglement $\mathcal{D}(G)$ (bottom). Pattern $P = \texttt{GTTCGTTATATG}$ occurs once in G (underlined), and pattern $\mathbf{spr}^3(P)$ occurs once in $\mathcal{D}(G)$ (also underlined). Patterns $P' = \texttt{CGTATTATT}$ occur twice in G (in red and blue respectively), and each occurrence can be detected in $\mathcal{D}(G)$ as in Theorem A.2.2 from the strings highlighted in red and blue.

We prove the following lemma.

Lemma A.4.1. *Given a* k-F *graph* $G = (T, E)$ *and a solid string* P *of length* m *such that* k *divides* m*, the string* $\mathbf{spr}^k(P)$ *occurs in* $\mathcal{D}(G)$ *if and only if there exists* i, j *with*

$P \in \mathcal{L}(G[i\,..\,j])$, *namely* P *occurs in* G *and can be constructed as a concatenation of entire successive strings in* $T[i]\,..\,T[j]$ *that are connected by edges.*

Proof. If $P \in \mathcal{L}(G[i\,..\,j])$ and $P = P_1\,..\,P_d$ is a factorisation of P in substrings of length k, then for every $\ell = 1\,..\,d-1$ one has $P_\ell \in T[i+\ell-1]$, $P_{\ell+1} \in T[i+\ell]$ and $(P_\ell, P_{\ell+1}) \in E_{i+\ell-1}$, which means that the set $\mathcal{D}(G)[i+\ell-1]$ contains the string $P_\ell \cdot \$ \cdot P_{\ell+1}$, and the concatenation of those strings for each ℓ is equal to $\mathbf{spr}^k(P)$. Conversely, observe that for every $\ell = 1\,..\,d-1$, one has $\mathbf{spr}^k(P)[(\ell-1)(2k+1)+k+1] = \$$. If $\mathbf{spr}^k(P)$ occurs in $\mathcal{D}(G)$, since by construction the letter $\$$ occurs only at position $k+1$ of each set in $\mathcal{D}(G)$, the occurrence has to start at the first position of a set, and that means that $\mathbf{spr}^k(P) \in \mathcal{L}(\mathcal{D}(G)[i\,..\,j])$ for some $1 \leqslant i \leqslant j \leqslant n$ (namely, it occurs with starting and ending offset 0). This implies that for such i, j and for each $1 \leqslant \ell \leqslant d-1$ one has $\mathbf{spr}^k(P)[(2k+1)(\ell-1)\,..\,(2k+1)\ell] \in \mathcal{D}(G)[i+\ell-1]$. But $\mathbf{spr}^k(P)[(2k+1)(\ell-1)\,..\,(2k+1)\ell] = P_\ell \$ P_{\ell+1}$, which means that $P_\ell \in T[i+\ell-1]$, $P_{\ell+1} \in T[i+\ell]$, and $(P_\ell, P_{\ell+1}) \in E_{i+\ell-1}$. Since this holds for every $1 \leqslant \ell \leqslant d-1$, we deduce that $P \in \mathcal{L}(G[i\,..\,j])$. □

Example A.4.2. Let $P = \texttt{GTTCGTTATATG}$, and $P' = \texttt{CGTATTATT}$ as in Example A.4.1, and let $G = (\tilde{T}, E)$ be the k-F graph from Figure 41. The pattern P occurs once in G, ending in the fourth segment (underlined occurrence in the figure). This corresponds to a unique occurence of $\mathbf{spr}^3(P) = \texttt{GTT} \cdot \$ \cdot \texttt{CGTCGT} \cdot \$ \cdot \texttt{TATTAT} \cdot \$ \cdot \texttt{ATG}$ in $\mathcal{D}(G)$, ending in the third segment. The string P' occurs twice in G (in red and in blue on the figure). However, the string $\mathbf{spr}^3(P') = \texttt{CGT} \cdot \$ \cdot \texttt{ATTATT} \cdot \$ \cdot \texttt{ATT}$ occurs only once in $\mathcal{D}(G)$ (in red). This is because the blue occurrence has nonzero starting and ending offsets.

As illustrated in the above example, Lemma A.4.1 allows us, given a k-F G and a pattern P whose length is a multiple of k, to use a $(2k+1)$-D string to detect the occurrences of P starting at the first position of a segment of G. We now extend this result to arbitrary pattern length and starting position, in order to prove Theorem A.2.2.

Theorem A.2.2. MATCH(SOLID,k-F) *can be solved in* $\mathcal{O}(\sqrt{m}(|E| + N + n\log^2 m))$ *time.*

Proof. Let $G = (T, E)$ be a k-F graph of length n and $P = P[1\,..\,m]$ be a solid pattern. Let us first assume that $k < \sqrt{m}$. We construct the disentanglement of G, which is a $(2k+1)$-D string $\mathcal{D}(G)$. Let us assume that P occurs in G starting at position i and ending at position j (we have $i < j$ because $k < \sqrt{m}$ and thus every occurrence spans more than one segment). This means, by definition, that there exist strings s, t and $P_1, .., P_{j-i+1}$, with $0 \leqslant |s|, |t| < k$, $0 < |P_1|, |P_{j-i+1}| \leqslant k$ and $|P_2| = \ldots = |P_{j-i}| = k$, such that $s \cdot P_1 \in T[i]$, $P_{j-i+1} \cdot t \in T[j]$ and $P_{\ell-i+1} \in T[\ell]$ for every $i < \ell < j$, and such that $P_1 \cdots P_{j-i+1} = P$. The lengths of s and t are the starting and ending offsets of the occurrence, respectively, which uniquely determine the string $\mathbf{spr}^k_{|s|}(P) \coloneqq P_1 \cdot \$ \cdot P_2 \cdot \mathbf{spr}^k(P_2\,..\,P_{j-i}) \cdot P_{j-i} \cdot \$ \cdot P_{j-i+1}$. Let $\hat{P} \coloneqq s \cdot P \cdot t$. We have that $\hat{P} \in \mathcal{L}(G[i\,..\,j])$, which, by Lemma A.4.1, means precisely that $\mathbf{spr}^k(\hat{P})$ occurs in $\mathcal{D}(G)$. We can rewrite $\mathbf{spr}^k(\hat{P}) = s \cdot \mathbf{spr}^k_{|s|}(P) \cdot t$, and we deduce that P occurs in G with starting offset $|s|$ in some segment if and only if $\mathbf{spr}^k_{|s|}(P)$ occurs in $\mathcal{D}(G)$ (observing the occurrences of $\$$ in the constructed strings, every occurrence of $\mathbf{spr}^k_{|s|}(P)$ is always part of an occurrence of $s \cdot \mathbf{spr}^k_{|s|}(P) \cdot t$, for some s, t of the right lengths). To find every occurrence of P in G, we can then search for the k patterns $\mathbf{spr}^k_0(P), .., \mathbf{spr}^k_{k-1}(P)$ in $\mathcal{D}(G)$.

Notice that $\mathcal{D}(G)$ has size $\mathcal{O}(k|E|)$. To search for $\mathbf{spr}_0^k(P), \ldots, \mathbf{spr}_{k-1}^k(P)$ we can apply a modified version of the algorithm from Theorem A.2.1. The difference is that, while in Theorem A.2.1 we need k searches for a single pattern (one for each possible starting offset in the k-D string), here we are already given k distinct patterns, each encoding a different possible starting offset in G, and we must search for occurrences of each of such patterns in $\mathcal{D}(G)$ that start at the beginning of a segment and end at the end of another one. Hence, we only need to compute the IDs for the length-$(2k+1)$ substrings of $\mathbf{spr}_0^k(P), \ldots, \mathbf{spr}_{k-1}^k(P)$ that contain a \$ at their $k+1$th position (as they are the only substrings that can match a full string from a segment of $\mathcal{D}(G)$). This can be done simultaneously for all $\mathbf{spr}_0^k(P), \ldots, \mathbf{spr}_{k-1}^k(P)$ in $\mathcal{O}(m)$ total time by using the suffix tree of P. Indeed, each substring for which we need to compute an ID consists of a substring of P of length $2k$, with an extra central dollar. Similarly, we can compute the IDs of all the strings from $\mathcal{D}(G)$ (and discard those that do not occur in any of the patterns) in $\mathcal{O}(k|E|)$ total time.

In addition to the substring IDs, we construct arrays e_i and b_i in $\mathcal{O}(k|E|)$ time (corresponding to the total size of $\mathcal{D}(G)$), as in Theorem A.2.1. The search takes $\mathcal{O}(n \log^2(\frac{m}{k}))$ time for each pattern; since each of the patterns is searched only once, we obtain a total time of $\mathcal{O}(k|E| + kn\log^2(\frac{m}{k})) = \mathcal{O}(\sqrt{m}(|E| + n\log^2 m))$ for the cases $k < \sqrt{m}$.

Now consider the case $k \geqslant \sqrt{m}$. Observe that, in this case, we have $n \leqslant \frac{N}{\sqrt{m}}$, because it always holds that $n \leqslant \frac{N}{k}$. By simply applying the algorithm of Theorem A.2.4 in this special case we obtain a time complexity in $\mathcal{O}(N\sqrt{m} + |E|\log m)) = \mathcal{O}(\sqrt{m}(N + |E|))$. Combining the two cases, the total time becomes $\mathcal{O}(\sqrt{m}(|E| + N + n\log^2 m))$. □

Example A.4.3. Consider again $P' = \texttt{CGTATTATT}$ as in Example A.4.1 and the two occurrences in the F graph shown in Figure 41. We can now detect its blue occurrence in G by searching for $\mathbf{spr}_1^3(P') = \texttt{CG}\cdot\$\cdot\texttt{TATTAT}\cdot\$\cdot\texttt{TATTAT}\cdot\$\cdot\texttt{T}$, that occurs in $\mathcal{D}(G)$.

A.5 MATCHING A VARIABLE PATTERN IN A k-D OR k-F TEXT

In this section, we study the complexity of finding all the occurrences of non-solid patterns in a k-D or k-F text. We begin by formally extending Definition A.3.1 to the more general case where both P and T are either GD strings or F graphs.

Definition A.5.1. A pattern P of type GD or F has an occurrence ending at position j in a text of type GD or F if $\exists i \in [1, j]$ such that $\mathcal{L}(P) \cap \mathcal{L}(T^{\ell,r}[i\,..\,j]) \neq \emptyset$ for some $\ell \in [0, k_i - 1]$ and $r \in [0, k_j - 1]$.

The case where both P and T are 1-D is well-studied in the literature. The definition of *indeterminate* string given in [153] coincides with our definition of 1-D string; in the same paper, the authors proposed an $\mathcal{O}(n \log m)$-time algorithm for MATCH(1-D,1-D) in the case where the alphabet is of constant size [153, Lemma 17]. A similar algorithm for constant-size alphabets was also proposed in [243]. These results were complemented in [153, Theorem 22] with a quadratic conditional lower bound for cases where the alphabet is not of constant size.

This lower bound clearly applies also to MATCH(1-D,k-D) and MATCH(k-D,1-D) when the alphabet size is not constant. In Section A.5.1, we prove that, in these cases,

a quadratic conditional lower bound holds even when the alphabet has only three letters. In Section A.5.2, we consider problems Match(1-D,1-F), Match(1-F,1-D), Match(1-F,1-F). For all these cases, we prove a quadratic conditional lower bound for constant-size alphabets. This implies that when the pattern is not solid and at least one between the pattern and the text is a graph, the best we can hope to achieve is a quadratic-time algorithm.

The conditional lower bounds we prove rely on a famous conjecture: the *Orthogonal Vectors Hypothesis*, which is, in turn, implied by SETH [157, 259].

Definition A.5.2 (Orthogonal Vectors (OV))**.** Given two sets $X, Y \subseteq \{0,1\}^d$ such that $|X| = |Y| = n$ and $d = \omega(\log n)$, determine whether there exist $x \in X$ and $y \in Y$ such that x and y are orthogonal, namely, $x \cdot y = \sum_{i=1}^{d} x[i] \cdot y[i] = 0$.

In the rest of the chapter, we will use the instance $X = \{x_1 = 010, x_2 = 100, x_3 = 011\}$, $Y = \{y_1 = 001, y_2 = 010, y_3 = 110\}$ of OV as our running example. Note that $x_2 = 100$, $y_2 = 010$ is a valid solution since $x_2 \cdot y_2 = 0$.

Conjecture A.5.1 (Orthogonal Vectors Hypothesis (OVH))**.** *No (deterministic or randomized) algorithm can solve OV on vector sets* $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$, *in time* $\mathcal{O}(n^{2-\epsilon}\mathrm{poly}(d))$ *for any* $\epsilon > 0$.

Here we summarize the general idea used in the proofs of this section. We start with an instance of OV: $X, Y \subseteq \{0,1\}^d$ such that $|X| = |Y| = n$. Then we construct in $\mathcal{O}(nd)$ time a pattern P and a text T such that there is a match of P in T if and only if there exists a pair of orthogonal vectors between X and Y. We will ensure that the size of both P and T is $\mathcal{O}(nd)$. In this way, a subquadratic algorithm for matching P in T would imply an $\mathcal{O}((nd)(nd)^{1-\epsilon}) = \mathcal{O}(n^{2-\epsilon}\mathrm{poly}(d))$ time algorithm for OV, which would contradict OVH. The reader familiar with the work of Backurs and Indyk [31] might wonder if some of the reductions they proposed – in particular, those proving SETH-hardness for pattern matching or membership testing with regular expressions of the type $\cdot|\cdot$ – may be directly applied or easily modified to prove lower bounds for the problems in this section. We remark that this is not the case, as the reductions in [31] rely heavily on the strings in each segment having different lengths, while the segments of k-D strings and k-F graphs contain strings of equal length.

A.5.1 *Matching a* 1-D *pattern in a* k-D *text*

We start by introducing a gadget that will be used in several reductions. Given a vector $y \in \{0,1\}^d$, let $Q(y)$ be a 1-D string with d segments $Q(y)[h]$, $1 \leqslant h \leqslant d$, defined as

$$Q(y)[h] = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix} \quad \text{if } y[h] = 0; \qquad Q(y)[h] = \{0\} \quad \text{if } y[h] = 1.$$

A similar encoding of the vectors is often used in the reductions from OV: the key property, which is clear by construction, is that a string x occurs in $Q(y)$ only if it encodes a vector orthogonal to y, as stated in the following lemma.

Lemma A.5.2. *Let* $x, y \in \{0,1\}^d$*, then the string* $x[1]x[2]\cdots x[d]$ *occurs in* $Q(y)$ *if and only if* $x \cdot y = 0$*.*

Theorem A.2.5. *No algorithm can solve* MATCH(1-D,k-D) *on constant alphabet for every* k *in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Proof. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ be an instance of OV. We define a 1-D pattern P and a k-D text T such that P occurs in T if and only if $\exists x \in X$ and $y \in Y$ with $x \cdot y = 0$. We start by constructing pattern gadgets $Q(y_i)$, for each $y_i \in Y$. We then concatenate such gadgets into a single 1-D pattern using an extra character \$ that will force synchronisation with T:

$$P = \left\{\$\right\} Q(y_1) \left\{\$\right\} \cdots \left\{\$\right\} Q(y_n).$$

We remark that the size of P is $M = \mathcal{O}(nd)$. To build the text T, we list all the vectors from X in one segment W, surrounded by $n-1$ segments of the form $Z = \{\$0^d\}$ on both sides:

$$T = \underbrace{\{\$\overbrace{0\cdots0}^{d \text{ times}}\}\cdots\{\$0\cdots0\}}_{n-1 \text{ times}} \underbrace{\left\{\begin{array}{c}\$x_1[1]\cdots x_1[d]\\ \vdots \\ \$x_n[1]\cdots x_n[d]\end{array}\right\}}_{W} \underbrace{\overbrace{\{\$0\cdots0\}}^{Z}\cdots\{\$0\cdots0\}}_{n-1 \text{ times}}.$$

Clearly, T is a $(d+1)$-D string of size $N = \mathcal{O}(nd)$. The idea is that Z can match any gadget $Q(y_i)$, while W allows matches only from gadgets encoding vectors that are orthogonal to a vector in X (Lemma A.5.2). Since P has n gadgets of length d, any occurrence of P in T must span a string in W (see Figure 42). Summing up, if P has a match in T starting at $T[i]$, then it must start at the first position of $T[i]$ because the \$ symbol matches nowhere else. Then i must be less or equal than n, and the intersection of $\mathcal{L}(Q(y_{n-i+1}))$ and $\mathcal{L}(W)$ must be non empty, implying that vector y_{n-i+1} is orthogonal to some vector of X. Therefore, deciding if there exists a pair of orthogonal vectors from X and Y can be reduced in $\mathcal{O}(nd)$ time to an instance of matching a 1-D pattern P of size $\mathcal{O}(nd)$ in a $(d+1)$-D text T of size $\mathcal{O}(nd)$. If we could find a match for P in T in $\mathcal{O}(N^{1-\epsilon}M)$ or $\mathcal{O}(NM^{1-\epsilon})$ time, then we could solve OV in $\mathcal{O}((nd)(nd)^{1-\epsilon}) = \mathcal{O}(n^{2-\epsilon}\,\mathrm{poly}(d))$ time, which would contradict OVH. □

Theorem A.2.6. *No algorithm can solve* MATCH(k-D,1-D) *on constant alphabet for every* k *in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Proof. The reduction is entirely analogous to that of Theorem A.2.5, except now we define a 1-D text $T = \left\{\$\right\} Q(y_1) \left\{\$\right\} \cdots \left\{\$\right\} Q(y_n)$ and $(d+1)$-D pattern $P = W$ of length 1 containing all the vectors from X. We can conclude as before. □

Remark that in both Theorems A.2.5 and A.2.6 we have $k = \omega(\log n)$, where n is both the number of vectors of OV and the length of the text; it remains open whether there exist subquadratic algorithms when $k = \mathcal{O}(\log n)$.

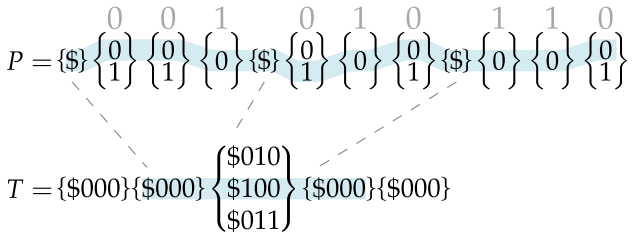


Figure 42. Example of the pattern and text constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ in the proof of Theorem A.2.5. The highlighted paths represent an occurrence of P in T, which identifies two orthogonal vectors: $x_2 = 100 \in X$ and $y_2 = 010 \in Y$. The dashed lines are drawn to emphasise the synchronisation forced by the symbol \$.

A.5.2 *Matching a* 1-F *pattern in a* 1-D *text*

In the following proofs, we will use a three-level strategy first introduced in [116, 117].

We start by defining two 1-D strings, P and T (one for the pattern and one for the text), underlying all the 1-F graphs in the reductions of this section. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ be any instance of OV. T and P are over the constant-size alphabet $\Sigma = \{0, 1, u, b, \$\}$. The role of the letters u and b is to identify parts of the segments that we will call their upper and bottom level. We build P as the concatenation of n 1-D string gadgets, one for each vector of X: given $x_i \in X$, we define

$$p(x_i) = \left\{ \begin{matrix} u \\ x_i[1] \\ b \end{matrix} \right\} \cdots \left\{ \begin{matrix} u \\ x_i[d] \\ b \end{matrix} \right\}.$$

We add the symbol \$ at the beginning and end of the pattern to force the occurrences to start/end in some specific parts of the text. The complete 1-D pattern then is

$$P = \{\$\}p(x_1)p(x_2)\cdots p(x_n)\{\$\}, \tag{132}$$

thus $|P| = nd + 2 = \mathcal{O}(nd)$ and $|P| = 3nd + 2 = \mathcal{O}(nd)$ (see Figure 43 for an example).

T consists of three different 1-D strings: T_{left}, $T^{\perp}$ and T_{right}. The 1-D string T_{left} consists of a single segment $\{\$\}$ followed by $n-1$ gadgets $U_{left} = \{u\}^{d-1} \left\{ \begin{matrix} \$ \\ u \end{matrix} \right\}$. In a symmetric way, T_{right} consists of $n-1$ gadgets $U_{right} = \left\{ \begin{matrix} b \\ \$ \end{matrix} \right\} \{b\}^{d-1}$ followed by $\{\$\}$. The 1-D string $T^{\perp}$ is built using a slight variation of the gadgets $Q(y_i)$ (see proof of Theorem A.2.5) extended with a three-level structure. Given $y_j \in Y$, we define $T_j^{\perp}[1] = \{u\} \cup Q(y_j)[1] \cup \{b, \$\}$, $T_j^{\perp}[h] = \{u\} \cup Q(y_j)[h] \cup \{b\}$ for $2 \leqslant h \leqslant d-1$ and $T_j^{\perp}[d] = \{u, \$\} \cup Q(y_j)[d] \cup \{b\}$. We define $T^{\perp}$ as the concatenation of $T_1^{\perp} \cdots T_n^{\perp}$. Finally, the full 1-D text T is the concatenation of T_{left}, $T^{\perp}$ and T_{right} (see Figure 44 for an example):

$$T = \{\$\}U_{left}^{n-1} T_1^{\perp} \cdots T_n^{\perp} U_{right}^{n-1}\{\$\}. \tag{133}$$

We remark that the size of T is $\mathcal{O}(nd)$, since it is built from three 1-D strings each of size $\mathcal{O}(nd)$.

Theorem A.2.8. *No algorithm can solve* MATCH(1-D,1-F) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Proof. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ and $\Sigma = \{0, 1, u, b, \$\}$. We build the pattern P as in Equation (132). To build the 1-F text G_T, we consider T as in Equation (133) and we first separately construct three different 1-F graphs: $G_{\mathit{left}} = (T_{\mathit{left}}, E_{\mathit{left}})$, $G^{\perp} = (T^{\perp}, E^{\perp})$ and $G_{\mathit{right}} = (T_{\mathit{right}}, E_{\mathit{right}})$. The set of edges E_{left} of G_{left} contain every possible edge between consecutive segments except for that of type $(u, \$)$, which has no edge incoming to $\$$. Symmetrically, the set E_{right} of G_{right} contains every possible edge between consecutive segments except for the one of the form $(\$, b)$, which has no edge outgoing from $\$$.

To define $E^{\perp}$, we first define the edges for each $T^{\perp}_j$. For these gadgets, we allow all the edges of the form (u, u), (b, b) and (x, y) for any $x, y \in \{0, 1\}$. In this way, any matching string of $\mathcal{L}(\mathcal{P})$ passing through $T^{\perp}_j$ must follow the upper level identified by letters u, the bottom level identified by letters b or the orthogonal level corresponding to $Q(y_j)$ (recall that any string $x \in \{0,1\}^d$ matches $Q(y_j)$ if and only if $x \cdot y_j = 0$). To complete $E^{\perp}$, for all consecutive segments $T^{\perp}_i[d]$ and $T^{\perp}_{i+1}[1]$ we build two sets of edges: the first connects the upper level of $T^{\perp}_i[d]$ (i.e. u and $\$$) to the upper and orthogonal level of $T^{\perp}_{i+1}[1]$; the second connects the orthogonal and bottom level of $T^{\perp}_i[d]$ with the bottom level of $T^{\perp}_{i+1}[1]$ (i.e. b and $\$$). Finally, $G_T = (T, E_T)$ is the union of G_{left}, $G^{\perp}$ and G_{right} where we add to E_T all the edges from the last segment of T_{left} to the upper level and orthogonal level of $T^{\perp}_1[1]$ and all the edges from the orthogonal and bottom level of $T^{\perp}_n[d]$ to the first segment of T_{right}. We remark that $|E_T| = \mathcal{O}(nd)$, thus since $|T| = \mathcal{O}(nd)$, the size of G_T is $\mathcal{O}(nd)$.

From the construction of the edges, we have that the only allowed edges from $\$$ are $(\$, u)$, $(\$, 0)$ and $(\$, 1)$ and the only allowed edges to $\$$ are $(0, \$)$, $(1, \$)$ and $(b, \$)$. Hence, the pattern must start either on the upper level or in G_{left} and finish either on the bottom level or in G_{right}. Since the bottom level (thus also G_{right}) can be reached only after reading the orthogonal level, we deduce by Lemma A.5.2 that the pattern has a match in G_T if and only if x_i matches with $Q(y_j)$ for some i, j, namely $x_i \cdot y_j = 0$. See Figure 43 for an example. □

Theorem A.2.9. *No algorithm can solve* MATCH(1-F,1-D) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Proof. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ and $\Sigma = \{0, 1, u, b, \$\}$. We build the text T as in Equation (133). To build the pattern $G_P = (P, E_P)$, we consider P as in Equation (132) and we construct the set of edges E_P with the same criteria used for G_T in the proof of Theorem A.2.8. At the beginning of the pattern, we add the edges $(\$, u)$, $(\$, x_1[1])$, and at the end we add the edges $(x_n[d], \$)$ and $(b, \$)$. For each $x_i \in X$, we add to $p(x_i)$ all the edges of the form (u, u), (b, b) and (x, y) for any $x, y \in \{0, 1\}$. Furthermore, from $p(x_i)$ to $p(x_{i+1})$ we add the edges: (u, u), $(u, x_{i+1}[1])$, $(x_i[d], b)$, (b, b). Clearly, $|E_P| \leqslant 4nd = \mathcal{O}(nd)$, thus the size of G_P is $|P| + |E_P| = \mathcal{O}(nd)$. Furthermore, we can deduce that the language of G_P is $\mathcal{L}(G_P) = \{\$u^{(i-1)d} \cdot x_i \cdot b^{(n-i)d}\$: i = 1, \ldots, n\}$. Remark that each x_i is over the alphabet $\{0, 1\}$. Thus, because of the position of the

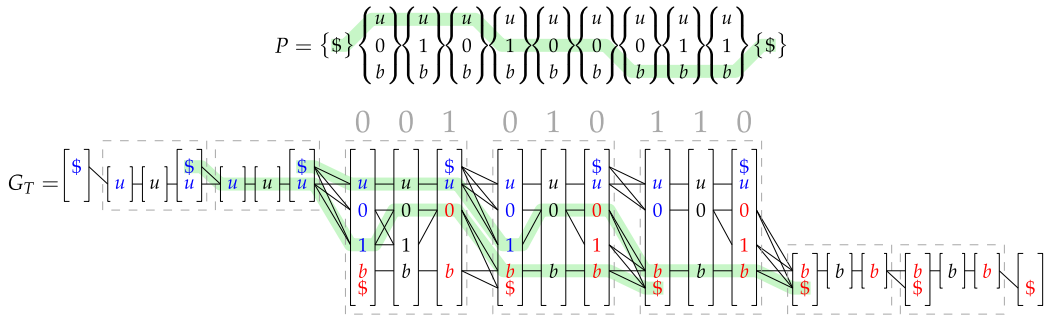


Figure 43. Example of a 1-D pattern P and 1-F text G_T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem A.2.8. Highlighted paths show some occurrences of P in T, given by the string $\$uuu100bbb\$ \in \mathcal{L}(P)$ which belongs also to $\mathcal{L}(T[4 \dots 14])$ and $\mathcal{L}(T[7 \dots 17])$. This corresponds to the fact that $x_2 = 100$ is orthogonal to $y_1 = 100$ and $y_2 = 010$.

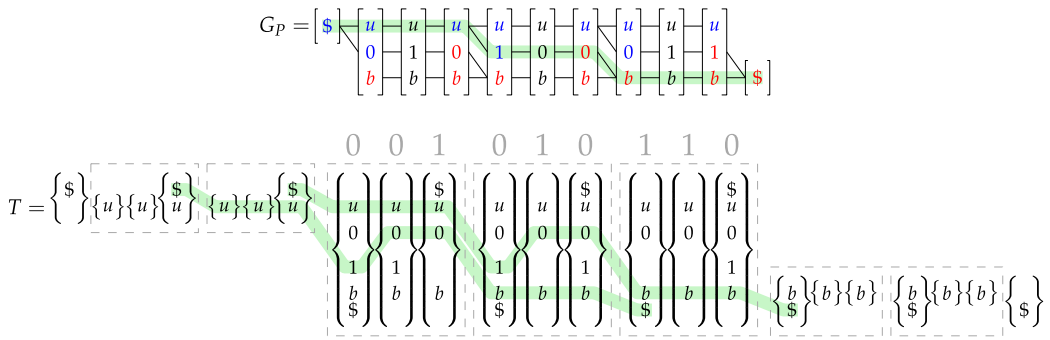


Figure 44. Example of a 1-F pattern G_P and a 1-D text T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem A.2.9, with occurrences of G_P in T highlighted.

symbol \$, Lemma A.5.2 applies, that is, there is an occurrence of P in T if and only if x_i matches with $Q(y_j)$ for some i, j, namely $x_i \cdot y_j = 0$. See Figure 44 for an example. □

Finally, the following result follows directly from a reduction that uses G_T as in the proof of Theorem A.2.8 and G_P as in the proof of Theorem A.2.9.

Theorem A.2.10. *No algorithm can solve* Match(1-F,1-F) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$, *unless OVH is false.*

A.5.3 *Matching a* 2-D *pattern in a* 2-D *text*

This section considers the problem Match(2-D,2-D). Note that Theorem A.2.5 and Theorem A.2.6 directly imply a quadratic conditional lower bound for the general Match(k-D,k-D) problem. However, in the reductions underlying the two theorems, the width k of the k-D string is $\mathcal{O}(d)$, i.e. linear in the length of the vectors of OV, which in turn is $\omega(\log n)$, where n is both the length of the text and the number of vectors of OV. Therefore, in principle, those results do not rule out the possibility that Match(k-D,k-D) can be solved faster than quadratically when e.g. $k = \mathcal{O}(1)$. In

Theorem A.2.7, we prove this is not the case, as a quadratic conditional lower bound holds already when $k = 2$ and the alphabet is of constant size.

The reduction is similar in spirit to those of Theorems A.2.5 and A.2.6 and relies on the following gadgets. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ be any instance of OV. We construct a pattern P and a text T over the alphabet $\Sigma = \{0, 1, u, b, \$\}$. To build P, we first define n 2-D strings gadgets, one for each vector $x_i \in X$:

$$p^{(2D)}(x_i) = \left\{ \begin{matrix} uu \\ x_i[1]x_i[1] \\ bb \end{matrix} \right\} \cdots \left\{ \begin{matrix} uu \\ x_i[d]x_i[d] \\ bb \end{matrix} \right\}.$$

P is obtained by concatenating these gadgets, and adding between $p^{(2D)}(x_i)$ and $p^{(2D)}(x_{i+1})$ the segment:

$$\left\{ \begin{matrix} uu \\ ux_{i+1}[1] \\ x_i[d]b \\ bb \end{matrix} \right\}.$$

Furthermore, at the beginning of the pattern, we append the segment $\left\{ \begin{matrix} \$u \\ \$x_1[1] \end{matrix} \right\}$, and at the end we append the segment $\left\{ \begin{matrix} x_n[d]\$ \\ b\$ \end{matrix} \right\}$. The complete 2-D pattern then is

$$P = \left\{ \begin{matrix} \$u \\ \$x_1[1] \end{matrix} \right\} p^{(2D)}(x_1) \left\{ \begin{matrix} uu \\ ux_2[1] \\ x_1[d]b \\ bb \end{matrix} \right\} p^{(2D)}(x_2) \cdots \left\{ \begin{matrix} uu \\ ux_n[1] \\ x_{n-1}[d]b \\ bb \end{matrix} \right\} p^{(2D)}(x_n) \left\{ \begin{matrix} x_n[d]\$ \\ b\$ \end{matrix} \right\}, \tag{134}$$

thus $|P| = \mathcal{O}(nd)$ and $|P| = \mathcal{O}(nd)$ (see Figure 45 for an example). Note that this pattern encodes exactly the edges in the construction of G_P (see proof of Theorem A.2.9).

To build the text T, we introduce a variation of the gadget $Q(y)$ used in Sections A.5.1 and A.5.2. Given a vector $y \in \{0,1\}^d$, let $R(y)$ be a 2-D string with $d+1$ segments, defined as follows (see Figure 45 for an example):

- for $1 < h < d+1$:

$$R(y)[h] = \begin{Bmatrix} 00 \\ 01 \\ 10 \\ 11 \end{Bmatrix} \quad \text{if } y[h-1] = 0 \text{ and } y[h] = 0;$$

$$R(y)[h] = \begin{Bmatrix} 00 \\ 10 \end{Bmatrix} \quad \text{if } y[h-1] = 0 \text{ and } y[h] = 1;$$

$$R(y)[h] = \begin{Bmatrix} 00 \\ 01 \end{Bmatrix} \quad \text{if } y[h-1] = 1 \text{ and } y[h] = 0;$$

$$R(y)[h] = \begin{Bmatrix} 00 \end{Bmatrix} \quad \text{if } y[h-1] = 1 \text{ and } y[h] = 1;$$

- for $h = 1$:

$$R(y)[1] = \begin{Bmatrix} 00 \\ 11 \end{Bmatrix} \quad \text{if } y[1] = 0; \qquad R(y)[1] = \begin{Bmatrix} 00 \end{Bmatrix} \quad \text{if } y[1] = 1;$$

- for $h = d+1$:

$$R(y)[d+1] = \begin{Bmatrix} 00 \\ 11 \end{Bmatrix} \quad \text{if } y[d] = 0; \qquad R(y)[d+1] = \begin{Bmatrix} 00 \end{Bmatrix} \quad \text{if } y[d] = 1.$$

The following result is analogous to Lemma A.5.2.

Lemma A.5.3. *Let $x, y \in \{0,1\}^d$, then the string $\rho(x) = x[1]^3 x[2]^2 \cdots x[d-1]^2 x[d]^3$ occurs in $R(y)$ if and only if $x \cdot y = 0$.*

Proof. Notice that the length of string $\rho(x)$ and the length of the strings in $\mathcal{L}(R(y))$ are both $2d+2$. Thus, any occurrence of $\rho(x)$ must span $R(y)$ entirely from left to right. Moreover, by simple case analysis, $x[1]^3$ matches in $R(y)[1]R(y)[2]$ if and only if $x[1] \cdot y[1] = 0$. The same holds for $x[d]^3$, which matches in $R(y)[d]R(y)[d+1]$ if and only if $x[d] \cdot y[d] = 0$. For a generic substring $x[h]^2$, one character must match in $R(y)[h]$ and the other in $R(y)[h+1]$. Every possible configuration of $R(y)[h]$ always has a 0 both in the first and in the second column, hence $x[h]^2 = 00$ can always match. For $x[h]^2 = 11$, the match is not possible if $R(y)[h]$ does not have any 1 in the second column and $R(y)[h+1]$ does not have any 1 in the first column, but this can happen only if $y[h] = 1$. Summing up, the substring $x[h]^2$ has an occurrence in $R(y)[h]R(y)[h+1]$ if and only if $x[h] \cdot y[h] = 0$. □

We are now able to build the text T. For each $y_j \in Y$, we define $T_j^{\perp}[1] = \{uu\} \cup R(y_j)[1] \cup \{bb, \$\$\}$, $T_j^{\perp}[h] = \{uu\} \cup R(y_j)[h] \cup \{bb\}$ for $2 \leqslant h \leqslant d$ and $T_j^{\perp}[d+1] = \{uu, \$\$\} \cup R(y_j)[d+1] \cup \{bb\}$. The full 2-D text T is (see Figure 45 for an example):

$$T = \{\$\$\} U_{left}^{n-1} T_1^{\perp} \cdots T_n^{\perp} U_{right}^{n-1} \{\$\$\} \tag{135}$$

where

$$U_{left} = \{uu\}^d \begin{Bmatrix} \$\$ \\ uu \end{Bmatrix} \quad \text{and} \quad U_{right} = \begin{Bmatrix} bb \\ \$\$ \end{Bmatrix} \{bb\}^d.$$

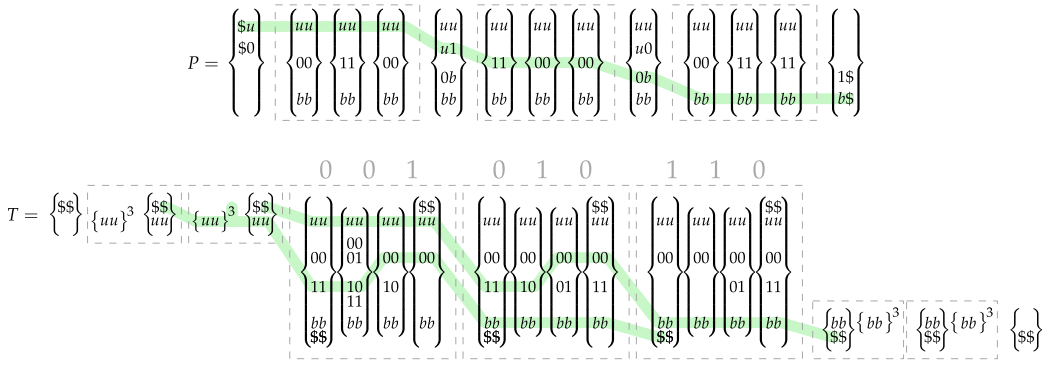


Figure 45. Example of a 2-D pattern P and 2-D text T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem A.2.7. Highlighted paths show some occurrences of P in T, given by the string $\$u^8 1^3 0^2 0^3 b^8 \$ \in \mathcal{L}(P)$ which corresponds to the fact that $x_2 = 100$ is orthogonal to $y_1 = 100$ and $y_2 = 010$.

Since $T_i^{\perp}$, U_{left}, U_{right} have all size $\mathcal{O}(d)$, then size of T is $\mathcal{O}(nd)$.

Theorem A.2.7. *No algorithm can solve* MATCH(2-D,2-D) *on constant alphabet in* $\mathcal{O}(M^{1-\epsilon}N)$ *nor in* $\mathcal{O}(MN^{1-\epsilon})$ *time for* $\epsilon > 0$*, unless OVH is false.*

Proof. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ and $\Sigma = \{0, 1, u, b, \$\}$. We build the pattern P as in Equation (134) and the text T as in Equation (135). Since any string in $\mathcal{L}(P)$ must start and end with the symbol \$, any occurrence of P in T must begin in the middle of a segment containing the string \$\$. Moreover, because of its length, any occurrence of P must either begin in the U_{left} part or end in the U_{right} part. In the first case, this implies that the occurrence must finish in the middle of a segment $T_j^{\perp}[1]$, and in the second case that it must start in the middle of a segment $T_i^{\perp}[d+1]$.

From the fact that the pattern always occurs with offset 1 w.r.t. the text, we can deduce that the only possible strings in $\mathcal{L}(P)$ that could match P are of the form:

$$\{\$u^{(i-1)(2d+2)} \cdot \rho(x_i) \cdot b^{(n-i)(2d+2)}\$: i = 1, \ldots, n\} \subsetneq \mathcal{L}(P).$$

This is true since at any matched character we are forced, by either the pattern or the text, to follow a match with the same rules explained in Theorem A.2.8 and Theorem A.2.9. Remark that $\rho(x_i)$ is over the alphabet $\{0, 1\}$. Thus, by Lemma A.5.3, we conclude that there is an occurrence of P in T if and only if $\rho(x_i)$ matches $R(y_j)$ for some i, j, namely $x_i \cdot y_j = 0$. See Figure 45 for an example of this reduction. □

A.6 MATCHING A SOLID PATTERN IN AN ED TEXT

In this section, we prove that a quadratic conditional lower bound for MATCH(SOLID,ED) holds even for a severely restricted type of ED strings, in which the length of the strings in each segment can differ by at most 1. Note that a quadratic conditional lower bound for MATCH(SOLID,ED) for general ED strings already exists, as it is directly implied by [30, Theorem 10]. The latter result is a reduction from OV to the problem of determining if a string P can be derived from a regular expression of the form $\cdot|\cdot$ (composition of concatenation, *or*, concatenation), and it implies a

lower bound for MATCH(SOLID,ED) by further applying the following simple reduction. Any regular expression of the form $\cdot|\cdot$ naturally corresponds to an ED string: the first concatenation generates strings of characters, which are the terms of an *or* operator and can be seen as strings belonging to the same segment of an ED string. The second concatenation operator is emulated by concatenating the segments, thus forming an ED string. We show this construction in the following example.

$$[a|bb]\,[aa|abb|abab]\,[bb|abab] \rightarrow \left\{\begin{matrix} a \\ bb \end{matrix}\right\} \left\{\begin{matrix} aa \\ abb \\ abab \end{matrix}\right\} \left\{\begin{matrix} bb \\ abab \end{matrix}\right\}$$

Let T' be the ED string obtained by appending a segment with a single new character \$ at the beginning and the end of the ED string we generated from the regular expression, and let $P' = \$P\$$. Through this reduction, any algorithm solving MATCH(SOLID,ED) in subquadratic time could be used to determine if P can be derived from T, contradicting the lower bound of [30, Theorem 10].

The result we present in this section is *stronger* than the lower bound implied by [30, Theorem 10] in the sense that the ED strings constructed from instances of OV through our reduction are way less general than those implied by the former result. We start by recalling a definition originally given in [155].

Definition A.6.1 (Global elasticity)**.** Given a degenerate string $T = T[1]\cdots T[n]$ let $|T[i]|_{max}$ (resp. $|T[i]|_{min}$) be the length of the longest (resp. shortest) string in $T[i]$. The *global elasticity* of T is

$$\Delta(T) = \sum_{i=1}^{n} (|T[i]|_{max} - |T[i]|_{min}).$$

The quantity Δ measures the difference between the length of the longest and the shortest string in the language $\mathcal{L}(T)$ and thus it gives a global property of the ED string. To obtain a measure of how much the lengths of the strings in the segments of ED string vary locally, we introduce the following definition.

Definition A.6.2 (Local elasticity)**.** Given a degenerate string $T = T[1]\cdots T[n]$, we define the *local elasticity* of T as

$$\Delta_{loc}(T) = \max_{i=1,\ldots,n} (|T[i]|_{max} - |T[i]|_{min}).$$

By definitions A.6.1 and A.6.2, it holds that $0 \leqslant \Delta_{loc}(T) \leqslant \Delta(T)$ and that $\Delta_{loc}(T) = 0$ if and only if $\Delta(T) = 0$, that is, if and only if T is a GD string. Intuitively, $\Delta_{loc}(T)$ measures "how far" is T from being a GD.

In the proof of [30, Theorem 10], the elastic degenerate string to which an instance of OV is reduced has local elasticity $\mathcal{O}(d)$ and global elasticity $\mathcal{O}(nd)$, where d is the length of the vectors of OV. In Theorem A.2.11, we prove that the quadratic conditional lower bound holds even if we restrict to ED strings with $\Delta_{loc} = 1$ (thus as close to GD strings as possible), on constant-size alphabet, and with global elasticity $\mathcal{O}(n)$.

Let us describe the gadgets we will use in our reduction. Let $X, Y \subseteq \{0,1\}^d$, $|X| = |Y| = n$ be any instance of OV. We will construct a solid pattern P and an ED text T over the constant-size alphabet $\Sigma = \{0, 1, \sharp, \flat, \$\}$. To build P, we define n gadgets, one for each vector of $x_i \in X$:

$$\alpha(x_i) = x_i[1]^3 \sharp x_i[2]^3 \sharp \cdots \sharp x_i[d]^3.$$

Then, we concatenate the gadgets in the following way:

$$P = \$\alpha(x_1)\flat^3\alpha(x_2)\flat^3 \cdots \flat^3\alpha(x_n)\$. \tag{136}$$

It can be easily verified that $\|P\| = |P| = \mathcal{O}(nd)$.

To define the ED text T, we first define the following sets of strings, which will be used as a tool to construct the segments of T:

$$U_{up} = \begin{Bmatrix} \$\$0 \\ \$\$1 \\ \flat\flat 0 \\ \flat\flat 1 \end{Bmatrix} \begin{Bmatrix} 00\sharp 0 \\ 00\sharp 1 \\ 11\sharp 0 \\ 11\sharp 1 \end{Bmatrix}^{d-1} \begin{Bmatrix} 00\flat \\ 11\flat \end{Bmatrix}$$

$$U_{mid} = \begin{Bmatrix} \$\$00 \\ \$\$11 \\ \flat\flat 00 \\ \flat\flat 11 \end{Bmatrix} \begin{Bmatrix} 0\sharp 00 \\ 0\sharp 11 \\ 1\sharp 00 \\ 1\sharp 11 \end{Bmatrix}^{d-1} \begin{Bmatrix} 0\flat\flat \\ 1\flat\flat \\ 0\$\$ \\ 1\$\$ \end{Bmatrix}$$

$$U_{down} = \begin{Bmatrix} \flat 000 \\ \flat 111 \end{Bmatrix} \begin{Bmatrix} \sharp 000 \\ \sharp 111 \end{Bmatrix}^{d-1} \begin{Bmatrix} \flat\flat \\ \$\$ \end{Bmatrix}.$$

We then define the *universal gadget* U such that $U[h] = U_{up}[h] \cup U_{mid}[h] \cup U_{down}[h]$ for all $1 \leqslant h \leqslant d+1$ (see Figure 46 for an example); we remark that $\Delta_{loc}(U) = 1$ and $\Delta(U) = 2$. Note that U will not appear as-is in T: it will rather be used to define the gadgets encoding the vectors from set Y. The following lemma will be useful to prove the correctness of our reduction in Theorem A.2.11. We say that a string t occurs in the up level of U (resp. mid, down level) if t occurs in U_{up} (resp. U_{mid}, U_{down}).

Lemma A.6.1. *For every $x \in \{0,1\}^d$ there exist exactly three disjoint occurrences of $\flat\alpha(x)\flat$ in U (one for each level).*

Proof. Comparing the length of $\flat\alpha(x)\flat$ to the lengths of the strings in $\mathcal{L}(U)$, we deduce that any occurrence must start at $U[1]$ and end at $U[d+1]$. In $U[1]$, we can either match $\flat x[1]$, $\flat x[1]^2$ or $\flat x[1]^3$. Since $U[2]$ is a 4-D string, we can extend the previous partial occurrences in at most one way (see Observation A.3.3), that is: $x[1]^2\sharp x[2]$ extends $\flat x[1]$ in the up level, $x[1]\sharp x[2]^2$ extends $\flat x[1]^2$ in the mid level and $\sharp x[2]^3$ extends $\flat x[1]^3$ in the down level. Iterating the above reasoning, we have the thesis. See Figure 46 for an example. □

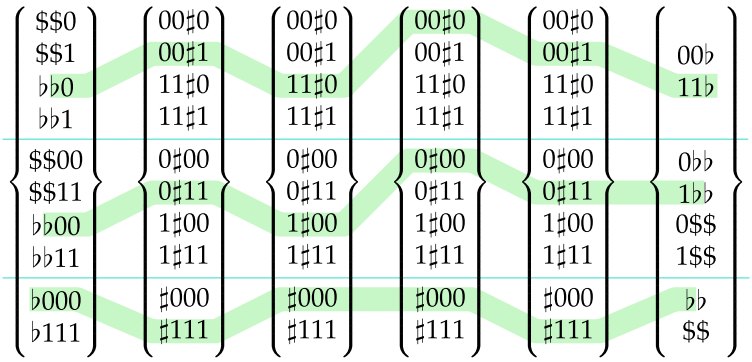


Figure 46. Occurrences of $\flat\alpha(x)\flat$ in U where $x = 01001$ (thus $d = 5$). The horizontal lines divide the up, mid and down levels.

Lemma A.6.1 holds also for strings $\$\alpha(x)\flat$ and $\flat\alpha(x)\$$.

We now construct a gadget $T_i^{\perp}$ for each vector $y_i \in Y$ by means of the universal gadget U: each of these gadgets will consist of $d+1$ segments, each containing a carefully selected subset of the strings from the segments of U. Gadget $T_i^{\perp}$, corresponding to vector $y_i \in Y$, is obtained from U using the same logic as gadgets $R(y_i)$ defined in Section A.5.3: we remove some strings from the segments of U_{mid} in such a way that the remaining strings in the language of U_{mid} represent the vectors orthogonal to y_i. More precisely, given a vector $y \in Y$, let $U^{\perp}_{mid}(y)$ be a GD string with $d+1$ segments, defined as follows:

- for $1 < h < d+1$:

$$U^{\perp}_{mid}(y)[h] = \begin{Bmatrix} 0\sharp00 \\ 0\sharp11 \\ 1\sharp00 \\ 1\sharp11 \end{Bmatrix} \quad \text{if } y[h-1] = 0 \text{ and } y[h] = 0;$$

$$U^{\perp}_{mid}(y)[h] = \begin{Bmatrix} 0\sharp00 \\ 1\sharp00 \end{Bmatrix} \quad \text{if } y[h-1] = 0 \text{ and } y[h] = 1;$$

$$U^{\perp}_{mid}(y)[h] = \begin{Bmatrix} 0\sharp00 \\ 0\sharp11 \end{Bmatrix} \quad \text{if } y[h-1] = 1 \text{ and } y[h] = 0;$$

$$U^{\perp}_{mid}(y)[h] = \{0\sharp00\} \quad \text{if } y[h-1] = 1 \text{ and } y[h] = 1;$$

- for $h = 1$:

$$U^{\perp}_{mid}(y)[1] = \begin{Bmatrix} \$\$00 \\ \$\$11 \\ \flat\flat00 \\ \flat\flat11 \end{Bmatrix} \quad \text{if } y[1] = 0; \qquad U^{\perp}_{mid}(y)[1] = \begin{Bmatrix} \$\$00 \\ \flat\flat00 \end{Bmatrix} \quad \text{if } y[1] = 1;$$

- for $h = d+1$:

$$U^{\perp}_{\mathtt{mid}}(y)[d+1] = \begin{Bmatrix} 0\flat\flat \\ 1\flat\flat \\ 0\$\$ \\ 1\$\$ \end{Bmatrix} \quad \text{if } y[d] = 0; \qquad U^{\perp}_{\mathtt{mid}}(y)[d+1] = \begin{Bmatrix} 0\flat\flat \\ 0\$\$ \end{Bmatrix} \quad \text{if } y[d] = 1.$$

We define $T^{\perp}_i[h] = U_{up}[h] \cup U^{\perp}_{\mathtt{mid}}(y_i)[h] \cup U_{down}[h]$ for all $1 \leqslant h \leqslant d+1$ and for all $y_i \in Y$. The proof of the following lemma is entirely analogous to that of Lemma A.5.3.

Lemma A.6.2. *Let $x_i, y_j \in \{0,1\}^d$; then the strings $\flat\alpha(x_i)\flat$, $\$\alpha(x_i)\flat$, and $\flat\alpha(x_i)\$$ occurs in the* mid *level of T_j if and only if $x_i \cdot y_j = 0$.*

For example, let $y = 010$ and $x = 100$, we have that

$$U^{\perp}_{\mathtt{mid}}(y) = \begin{Bmatrix} \$\$00 \\ \$\$11 \\ \flat\flat00 \\ \flat\flat11 \end{Bmatrix} \begin{Bmatrix} 0\sharp00 \\ 1\sharp00 \end{Bmatrix} \begin{Bmatrix} 0\sharp00 \\ 0\sharp11 \end{Bmatrix} \begin{Bmatrix} 0\flat\flat \\ 1\flat\flat \\ 0\$\$ \\ 1\$\$ \end{Bmatrix}$$

and $\flat\alpha(x)\flat = \flat111\sharp000\sharp000\flat$ occurs in $U^{\perp}_{\mathtt{mid}}(y)$ since x and y are orthogonal (see Figure 47 for a visual representation of the match).

Finally, the complete text is defined as:

$$T = U^{n-1}_{left}\{\flat\}T^{\perp}_1 T^{\perp}_2 \cdots T^{\perp}_n\{\flat\}U^{n-1}_{right}, \tag{137}$$

where

$$U_{left} = \begin{Bmatrix} \$\$\$ \\ \flat\flat\flat \end{Bmatrix} \begin{Bmatrix} 000 \\ 111 \end{Bmatrix} \begin{Bmatrix} \sharp000 \\ \sharp111 \end{Bmatrix}^{d-1} \quad \text{and} \quad U_{right} = \begin{Bmatrix} 000 \\ 111 \end{Bmatrix} \begin{Bmatrix} \sharp000 \\ \sharp111 \end{Bmatrix}^{d-1} \begin{Bmatrix} \flat\flat\flat \\ \$\$\$ \end{Bmatrix}.$$

Since $T^{\perp}_i$, U_{left}, U_{right} have all size $\mathcal{O}(d)$, the size of T is $\mathcal{O}(nd)$. Furthermore, since $\Delta_{\mathtt{loc}}(U_{left}) = \Delta_{\mathtt{loc}}(U_{right}) = 0$ and $\Delta_{\mathtt{loc}}(T^{\perp}_i) = 1$ for all $i = 1, \ldots, n$, then $\Delta_{\mathtt{loc}}(T) = 1$ and $\Delta(T) = \mathcal{O}(n)$. Armed with this reduction, we are ready to prove Theorem A.2.11.

Theorem A.2.11. *No algorithm can solve* Match(solid,ED) *on constant alphabet in $\mathcal{O}(M^{1-\epsilon}N)$ nor in $\mathcal{O}(MN^{1-\epsilon})$ time for $\epsilon > 0$, for any* ED *string such that the difference in the lengths of the strings in any segment is at most 1, and the length of all such strings is at most 4, unless OVH is false.*

Proof. The correctness of our reduction relies on Lemmas A.6.1 and A.6.2 and the following facts. (i) No occurrence of P in T can end within U^{n-1}_{left} or start within U^{n-1}_{right}; (ii) Any occurrence of P either starts within U^{n-1}_{left} or in the *first* segment of some gadget $T^{\perp}_i$, and ends either in U^{n-1}_{right} or in the *last* segment of some gadget $T^{\perp}_j$; (iii) Consider a partial occurrence of P ending in the last segment of some gadget $T^{\perp}_i$, $i \leqslant n-1$. If the occurrence ends matching some string in the *up* level of $T^{\perp}_i$, then it can only be extended by strings in the *up* or *mid* level of $T^{\perp}_{i+1}$; if it ends with a

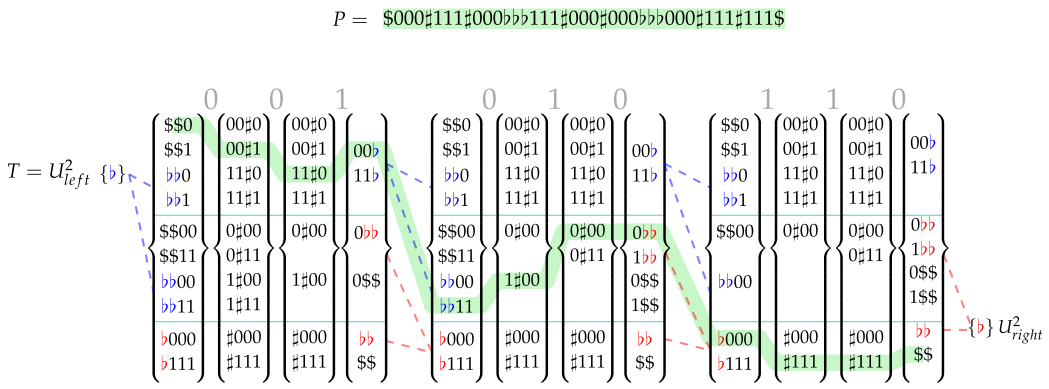


Figure 47. Example of a solid pattern P and ED text T constructed from $X = \{010, 100, 011\}$ and $Y = \{001, 010, 110\}$ following the proof of Theorem A.2.11: horizontal lines divide the *up*, *mid*, and *down* levels. Dashed lines highlight the allowed matching from $T_i^\perp$ to $T_{i+1}^\perp$, which are forced by the construction of the pattern P. The highlighted path shows an occurrence of P in T, which corresponds to the fact that $x_2 = 100$ is orthogonal to $y_2 = 010$.

match of some string in the *mid* or *down* level of $T_i^\perp$, then it can only be extended by strings in the *down* level of $T_{i+1}^\perp$. Furthermore, any partial occurrence ending at the segment $\{\flat\}$ right after U_{left}^{n-1} can only be extended by strings in the *up* or *mid* level of $T_1^\perp$, and no partial occurrence ending with a string in the *up* level of $T_n^\perp$ can be further extended (see Figure 47).

Fact (i) holds because, by construction, P is strictly longer than the strings in the languages $\mathcal{L}(U_{left}^{n-1})$ and $\mathcal{L}(U_{left}^{n-1})$. To prove Fact (ii), we notice that P always starts with \$ and the second letter is always 0 or 1. Since no string in the first segment of any gadget $T_i^\perp$ begins with 0 or 1, no partial occurrence of P starting within the last segment of a gadget $T_i^\perp$ can be extended to $T_{i+1}^\perp$ (nor to segment $\{\flat\}$). Symmetrically, the last two characters of P are always either 0 or 1 followed by \$, and since no string from the last segment of any gadget $T_i^\perp$ ends with 0 or 1, no occurrence can be completed with a \$ in the first segment of $T_{i+1}^\perp$.

To prove Fact (iii) notice that, in the pattern, $\flat\flat\flat$ occurs always between $\alpha(x_j)$ and $\alpha(x_{j+1})$. Furthermore, if $\alpha(x_j)\flat$ occurs in some segment $T_i^\perp$, then either (a) $x_j[d]\flat$ occurs in $T_i^\perp[d+1]$ (the last segment of $T_i^\perp$) and $\flat\flat x_{j+1}[1]$ occurs in $T_{i+1}^\perp[1]$ (the first segment of $T_{i+1}^\perp$) or (b) $x_j[d]\flat\flat$ occurs in $T_i^\perp[d+1]$ and $\flat x_{j+1}[1]$ occurs in $T_{i+1}^\perp[1]$. By construction (see Figure 47), Case (a) can only happen if $\alpha(x_j)\flat$ occurs in the up level and $\flat\flat\alpha(x_{j+1})$ occurs in the up or mid level. Similarly, Case (b) can only happen if $\alpha(x_j)\flat\flat$ occurs in the mid or down level and $\flat\alpha(x_{j+1})$ occurs in the down level.

By Fact (ii), and by the positions of the \$ and $\flat$ symbols in T and P, any complete occurrence of P must either start in the up or mid level of some $T_i^\perp$ or start in U_{left}^{n-1} and extend to the up or mid level of $T_1^\perp$. Similarly, it must either end in the mid or down level of some $T_j^\perp$ or in U_{right}^{n-1}, which can only be reached from the mid or down level of $T_n^\perp$. Combining this observation with Facts (i) and (iii) and Lemma A.6.1, we obtain that any occurrence of P must entirely traverse the mid level of some $T_i^\perp$, thus a substring of P containing some $\alpha(x_j)$ occurs in the mid level of $T_i^\perp$. By Lemma A.6.2, $\alpha(x_j)$ occurs in the mid level of $T_i^\perp$ if and only if y_i and x_j are orthogonal. □

We conjecture that the reduction underlying Theorem A.2.11 can be tweaked to prove a quadratic conditional lower bound for MATCH(SOLID,ED) in the case of a binary alphabet and $\Delta_{loc} = \mathcal{O}(1)$. We leave this as an open problem.

A.7 OPEN PROBLEMS

This chapter classified the complexity of problem MATCH(X, Y) as either truly subquadratic or at least quadratic (conditioned on SETH) for almost all combinations of types of variable strings X and Y. The two main cases that remain open are when X = SOLID and Y = GD, and when X = SOLID and Y = F. For these problems, corresponding to the yellow cells in Table 13, our algorithms, although subquadratic in most cases, are quadratic in the worst case, and no lower bound is known. Furthermore, it remains to determine the time complexity of matching a 1-D pattern in a k-D, GD or ED text when the maximum length of the strings in the segments of the text is at most logarithmic in the text length; and symmetrically, the time complexity of matching a k-D, GD or ED pattern in a 1-D text when the maximum length of the strings in the segments of the pattern is at most logarithmic. Finally, it remains to show whether the quadratic conditional lower bound for matching a solid pattern in an ED text holds when the alphabet is binary and has constant local elasticity.

B

ARE DEPTH-2 REGULAR EXPRESSIONS HARD TO INTERSECT?

We study the basic regular expression intersection testing problem, which asks to determine whether the intersection of the languages of two regular expressions is nonempty. A textbook solution to this problem is to construct the nondeterministic finite automaton that accepts the language of both expressions. This procedure results in a $\Theta(mn)$ running time, where m and n are the sizes of the two expressions, respectively. Following the approach of Backurs and Indyk [31] and Bringmann, Grønlund, and Larsen [62] on regular expression matching and membership testing, we study the complexity of intersection testing for homogeneous regular expressions of bounded depth involving concatenation, OR, Kleene star, and Kleene plus. Specifically, we consider all combinations of types of depth-2 regular expressions and classify the time complexity of intersection testing as either linear or quadratic, assuming SETH. The most interesting result is a quadratic conditional lower bound for testing the intersection of a "concatenation of +s" ($\circ+$) expression with a "concatenation of ORs" ($\circ|$) expression: this is the only hard case that does not involve the Kleene star operator and is not implied by existing lower bounds for the simpler membership testing problem. In Figure 48 there is a tree representations of a $\circ+$ and a $\circ|$ regular expression; their intersection, defined as the intersection of the languages they encode, is empty. If we add another c after the substring bc of the second regular expression, we obtain a $\circ|$ regular expression as in Figure 5, and in this case the intersection is given by the string bbcca. Furthermore, note that the $\circ|$ regular expressions are a particular kind of degenerate string, that we called 1-D strings in Appendix A.

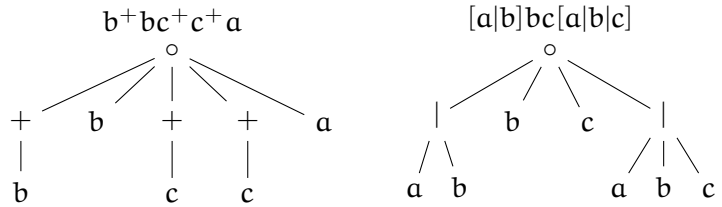


Figure 48. Example of a regular expression of type $\circ+$ (left) and $\circ|$ (right).

This chapter is based on the following submitted work: R. Ascone, G. Bernardini, A. Conte, V. Guerrini, G. Punzi. Are Depth-2 Regular Expressions Hard to Intersect. Submitted to *Journal of Computer and System Sciences*, 2025. doi:10.48550/arXiv.2507.03593 [21].

B.1 INTRODUCTION

A regular expression (regexp) $\mathcal{A}$ over an alphabet Σ is a formula that encodes a set of strings over Σ, called the *language* of $\mathcal{A}$, using the operators concatenation ($\circ$), OR ($|$), Kleene Star ($*$) and Kleene plus ($+$), formally defined in Appendix B.2. The *regular expression intersection testing* problem asks, given two regexps $\mathcal{A}$ and $\mathcal{B}$, to determine if the intersection of their languages is nonempty, that is, to decide whether a string exists that can be generated by both expressions. Similar to other basic tasks like regular expression pattern matching and membership testing, the intersection testing problem has a multitude of applications, including web services [260], data privacy [145], software security [175, 256], model checking [146], and satisfiability modulo theories [33, 74, 176, 248].

Given two regular expressions $\mathcal{A}$ of size m and $\mathcal{B}$ of size n, a textbook solution to intersection testing is to construct in linear time the nondeterministic finite automata accepting the language of $\mathcal{A}$ and $\mathcal{B}$, respectively, then to compute the automaton accepting the intersection of the two languages in quadratic time $\mathcal{O}(mn)$, and decide whether its language is nonempty in time $\mathcal{O}(mn)$. It is natural to wonder whether there exist special types of regular expressions for which intersection testing can be solved exponentially faster than in quadratic time, and if, on the contrary, for some types it can be proved that no truly subquadratic algorithm exists unless some popular hypotheses are falsified.

This question has been extensively considered in the literature for the related problems of regular expression membership testing and pattern matching, asking, respectively, if given a regexp $\mathcal{A}$ and a string t, the whole t or some of its fragments can be generated from $\mathcal{A}$. Backurs and Indyk [31] and Bringmann, Grønlund and Larsen [63] accomplished a systematic classification of the time complexity of *homogeneous* regexp pattern matching and membership testing as either quadratic or strongly subquadratic. A regexp is homogeneous if, given its representation as a rooted tree whose inner nodes correspond to operators and leaves correspond to letters of Σ, the inner nodes at any fixed level all correspond to the same operator. The *depth* of a homogeneous regexp is the depth of this tree, and its *type* is given by the sequence of operators on a root-to-leaf path of maximal length: see Figure 48.

Since a string is a depth-1 regular expression of type $\circ$, membership testing with a regexp of type t can be viewed as testing the intersection for types $\circ$ and t. This implies that if membership testing with type t is SETH-hard, so is testing the intersection of type t with any other type u that begins with the $\circ$ operator. This chapter systematically classifies the complexity of intersection testing for all types of homogeneous regexps of depth $\leqslant 2$ as either linear or quadratic, conditioned on the Strong Exponential Time Hypothesis (SETH) [157].

Our results. Our main results are summarised in Tables 15 and 16 and provide a full dichotomy for depth-2 homogeneous regexp intersection testing (blank cells are due to the symmetry of the problem). The hardness results on the first row of Table 15 are directly implied by the SETH-hardness of membership testing with regexps of type $\circ*$ proved by Backurs and Indyk [31]. The same authors devised near-linear-time algorithms for membership testing with all the other types of depth-2 homogeneous regexps. Interestingly, we prove that testing the intersection of a regexp of type $\circ|$ with one of type $\circ+$ is SETH-hard, despite the existing linear-time algorithms for

(t,u)-intersect.		ab^*a^*b	$[a\|b][b\|c]$	ab^+a^+a	$[aba\|ba]$	$[abba]^+$	$[baab]^*$
		$\circ*$	$\circ\|$	$\circ+$	$\|\circ$	$+\circ$	$*\circ$
ab^*a^*b	$\circ*$	**Cor.B.2.2** $\Omega((mn)^{1-\epsilon})$ [31]					
$[a\|b][b\|c]$	$\circ\|$		**Thm.B.4.1** $\mathcal{O}(m+n)$	**Thm.B.3.6** $\Omega((mn)^{1-\epsilon})$	**Thm.B.4.1** $\mathcal{O}(m+n)$		
ab^+a^+a	$\circ+$			**Thm.B.4.2** $\mathcal{O}(m+n)$			**Thm.B.4.4** $\mathcal{O}(m+n)$
$[aba\|ba]$	$\|\circ$				**Thm.B.4.3** $\mathcal{O}(m+n)$		
$[abba]^+$	$+\circ$						
$[baab]^*$	$*\circ$						

Table 15. Classification of the time complexity of the (t,u)-intersection testing problem for types $t,u \in \{\circ*, \circ|, \circ+, |\circ, +\circ, *\circ\}$. The lower bounds assume SETH.

membership testing with these types. We prove this result with a highly nontrivial reduction[1] from an unbalanced version of the Orthogonal Vectors problem [62]. The fact that regular expressions of type $\circ+$ have a form similar to those of type $\circ*$ might lead one to think that a minor modification of the reduction proposed in [31] for $(\circ*)$-membership testing would suffice to prove SETH-hardness for our problem. However, although some of the gadgets and assumptions in our reduction are indeed similar to those of Backurs and Indyk [31], their reduction relies heavily on the possibility of repeating each letter of the regexp 0 times: so heavily that removing this possibility leads to the existence of a linear-time algorithm for $(\circ+)$-membership testing. The lack of this possibility in our case thus requires entirely new techniques that exploit the form of the regexp of type $\circ|$. Interestingly enough, it is easy to test the intersection of two regexps of the same type $\circ|$ or $\circ+$ in linear time; thus, the quadratic conditional lower bound for $(\circ+, \circ|)$-intersection testing stems from the specific interplay of the two different types.

In addition to the cases mentioned above, we show that intersection testing is solvable in linear time for all the other combinations of depth-2 homogeneous regexps. The algorithms we propose are simple combinations of standard techniques. We note, however, that they all solve the more general problem of computing the whole intersection rather than just deciding whether it is nonempty.

Related work. The more general problem of testing the intersection of $k \geqslant 2$ regular expressions can be solved in time proportional to the product of the sizes of the regexps by building the product automaton, and is known to be **PSPACE**-complete when the number of expressions and their sizes are unbounded [168]. Recently, Su et al. [250] and Chen et al. [73] proposed algorithms for this problem that avoid explic-

1 An alternative reduction from gapped subsequence with length constraints matching [94], also conditioned on SETH, was proposed by an anonymous reviewer.

(t,u)-intersect.		$[a^+\|b^+\|c]$	$[a^*\|b^*\|c]$	$[a\|b\|c]^+$	$[a\|b\|c]^*$	$[a^*]^+$	$[a^+]^*$
		$\|+$	$\|*$	$+\|$	$*\|$	$+*$	$*+$
ab^*a^*b	$\circ*$						
$[a\|b][b\|c]$	$\circ\|$						
ab^+a^+a	$\circ+$						
$[aba\|ba]$	$\|\circ$	**Thm.B.4.6**		**Thm.B.4.5**		**Cor.B.2.1**	
$[abba]^+$	$+\circ$	$\mathcal{O}(m+n)$		$\mathcal{O}(m+n)$		$\mathcal{O}(m+n)$	
$[aab]^*$	$*\circ$						
$[a^+\|b^+\|c]$	$\|+$						
$[a^*\|b^*\|c]$	$\|*$						
$[a\|b\|c]^+$	$+\|$						
$[a\|b\|c]^*$	$*\|$						
$[a^*]^+$	$+*$						
$[a^+]^*$	$*+$						

Table 16. Time complexity of (t,u)-intersection testing for all depth-2 types t and $u \in \{|+, |*, +|, *|, +*, *+\}$.

itly constructing the product automaton and often reduce the time and search space in practice. Bala [32] showed that testing the intersection of an unbounded number of regexps remains **PSPACE**-complete when it is restricted to regexps without $+$ and with two nested stars, and it becomes **NP**-complete when there are no nested stars. Arrighi et al. [16] investigated the complexity of intersection testing for star-free language classes. Fernau et al. [122] studied the complexity of intersection testing on finite automata parameterised by the alphabet size and the maximum number of states of the input automata. De Oliveira Oliveira and Wehar [98] proved that the intersection of two DFAs with n states cannot be solved in $\mathcal{O}(n^{2-\epsilon})$ for any $\epsilon > 0$ assuming SETH. We remark that, since the automata constructed from regexps of types $\circ|$ or $\circ+$ are deterministic (modulo a simple linear-time transformation described in Theorem B.4.2), our conditional quadratic lower bound for $(\circ+, \circ|)$-intersection testing is a stronger result. Note that, in contrast, all the lower bounds in [31, 63] are for regexp types that give rise to NFAs.

The closely related problems of regular expression pattern matching and membership testing have been extensively investigated [50, 51, 52, 53, 54, 202], also in the streaming model [105]. The classification of the time complexity of pattern matching and membership testing with homogeneous regular expressions of [31, 63] has later been refined by Abboud and Bringmann [1] and Schepper [242]. A great deal of work has also been carried out in devising efficient algorithms for pattern matching and intersection testing for specific kinds of homogeneous regexps, including depth-2 expressions of type $\circ|$ (also known in the literature as *indeterminate, degenerate* or 1-*D* strings) or $|\circ$ (i.e. dictionaries of strings) and depth-3 expressions of type $\circ|\circ$ (known as *Elastic-Degenerate* (ED) strings). Dictionary matching can be solved in lin-

ear time with the classic Aho-Corasick algorithm [4]; indeterminate string matching is well-studied [81, 96, 135, 147] and can be solved in time $\mathcal{O}(n \log^2 m)$. To the best of our knowledge, indeterminate string intersection has not been considered explicitly in the literature, but a linear-time algorithm for intersecting the more general *Generalised Degenerate* strings (concatenations of sets of strings of equal length) applies [11]. Several parameterised algorithms also exist to find occurrences of a string in an ED text [15, 46, 49, 143]. Note that the roles of the string and the regexp are reversed with respect to the usual regexp pattern matching; as observed in [17], the quadratic conditional lower bound for membership testing of [31] implies a quadratic lower bound also for this version of the problem. Intersection testing for ED strings has been considered in [129] where, among other results, the authors showed SETH-hardness of the problem even when it is restricted to a binary alphabet.

Chapter organization. In Appendix B.2 we formally introduce the problem and the notation used throughout the chapter and consider the intersection of a depth-1 with a depth-2 regexp. In Appendix B.3 we prove SETH-hardness for $(\circ+, \circ|)$-intersection testing. In Appendix B.4 we present linear-time algorithms for all the other cases.

B.2 PRELIMINARIES

A regular expression (regexp) over an alphabet Σ and an operator set $O = \{\circ, |, +, *\}$ can be defined inductively as follows: c is a regular expression for any $c \in \Sigma$; given $\mathcal{A}$ and $\mathcal{B}$ two regular expressions, then $\mathcal{A} \circ \mathcal{B}$, $\mathcal{A}|\mathcal{B}$, $\mathcal{A}^+$ and $\mathcal{A}^*$ are regular expressions. Each regular expression $\mathcal{A}$ determines a language over Σ, denoted by $\mathcal{L}(\mathcal{A})$. In particular, given $c \in \Sigma$, $\mathcal{A}$ and $\mathcal{B}$ two regular expressions, we have: $\mathcal{L}(c) = \{c\}$, $\mathcal{L}(\mathcal{A} \circ \mathcal{B}) = \{w_1 w_2 : w_1 \in \mathcal{L}(\mathcal{A}) \text{ and } w_2 \in \mathcal{L}(\mathcal{B})\}$, $\mathcal{L}(\mathcal{A}|\mathcal{B}) = \mathcal{L}(\mathcal{A}) \cup \mathcal{L}(\mathcal{B})$, $\mathcal{L}(\mathcal{A}^+) = \{w_1 \cdots w_k : k \geqslant 1 \text{ and } w_1, \ldots, w_k \in \mathcal{L}(\mathcal{A})\}$, $\mathcal{L}(\mathcal{A}^*) = \mathcal{L}(\mathcal{A}) \cup \{\epsilon\}$, where ϵ denotes the empty string. We will sometimes omit the symbol $\circ$ and simply write $\mathcal{A}\mathcal{B}$ to denote the concatenation $\mathcal{A} \circ \mathcal{B}$.

We denote by $\Sigma_{\mathcal{A}} \subseteq \Sigma$ the set of letters appearing in $\mathcal{A}$. Given any regexp $\mathcal{A}$ and integer $k > 0$, $\mathcal{A}^k$ denotes the concatenation of k copies of $\mathcal{A}$. For any $c \in \Sigma$, we call c^k a *run* of letter c. Throughout the chapter, we assume that Σ is an integer alphabet of polynomial size[2], we denote by Σ^* the set of all strings over Σ (including ϵ), and $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. Furthermore, we denote by $[k]$ the set of integers $\{1, 2, \ldots, k\}$ and by $[k, \ell]$ any interval of integers.

Any regexp has a representation as a rooted tree with leaves labelled by letters from Σ and inner nodes labelled by operators. Each inner node has one or multiple children, depending on the operator: nodes labelled with $+$ or $*$ have exactly one child, while nodes labelled with $\circ$ or $|$ can have multiple children. We define the *size* of a regexp as the number of leaves in its tree representation[3]. A regular expression is *homogeneous of type* t *and depth* d if $t = t_1 \cdots t_d \in \{\circ, |, +, *\}^d$, at any level i of the tree representation all inner nodes are labelled with the operator t_i, and the depth of the tree is $d + 1$. We say that the type of any inner node at level i is t_i. Note that

2 Without this assumption, some of our upper bounds are increased by logarithmic factors. The same assumption was also implicitly made e.g. in [63, Lemma 4]

3 The size of a regexp is usually defined as the total number of nodes of the tree. We chose to count just the leaves to simplify some expressions across the chapter: clearly, for trees of constant depth, the two quantities are asymptotically the same.

leaves can be at any level. We assume that $t_i \neq t_{i+1}$; otherwise, we can merge the two levels into a single level. For example, the regexp $a^+b[c|a]$ is not homogeneous because both operators $+$ and $|$ label nodes at the first level of the tree representation; examples of depth-2 homogeneous regexps of type $\circ+$ and $\circ|$ are in Figure 48.

The (t,u)-*intersection testing* problem asks to decide if, given two homogeneous regular expressions of type t and u, respectively, the intersection of the languages of the two expressions is nonempty. Given two regular expressions $\mathcal{A}, \mathcal{B}$, we will use the shorthand $\mathcal{A} \sqcap \mathcal{B}$ to denote the language intersection $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B})$.

In this chapter, we study the computational complexity of the (t,u)-intersection testing problem for all combinations of types t, u of homogeneous regular expressions of depth 2. For completeness, we briefly report the computational complexity of the intersection problem when one of the two homogeneous regexps is of depth 1. When the type of the depth-1 regexp is $\circ$ (thus it corresponds to a string), the intersection testing problem coincides with the membership testing problem, whose computational complexity has been fully determined for all kinds of homogeneous regexps by Backurs and Indyk [31] and Bringmann, Grønlund and Larsen [63]. When the depth-1 regexp is of type $|, +$ or $*$, the intersection problem with any other regular expression can be solved in linear time with folklore algorithms.

Remark B.2.1. Given a regular expression $\mathcal{A}$ of depth 1 of type $|, +$ or $*$ and any other regular expression $\mathcal{B}$, we can determine in linear time whether $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$.

Thus, let us focus only on homogeneous regular expressions of depth 2. We first note that regular expressions of types $*+$ and $+*$ are equivalent to regexps of type $*$. Indeed, a regular expression of type $*+$ is of the form $(c^+)^* = (c^*)^+ = c^*$ for some $c \in \Sigma$. From Remark B.2.1, we immediately obtain Corollary B.2.1.

Corollary B.2.1. *Problems $(*+, t)$-intersection testing and $(+*, t)$-intersection testing can be solved in linear time for any type t of homogeneous regexp.*

Backurs and Indyk [31] proved that, assuming SETH, the complexity of membership testing with a regexp of type $\circ*$ (a.k.a. $(\circ, \circ*)$-intersection) is $\Omega((mn)^{1-\alpha})$ for all $\alpha > 0$. This result immediately implies the same conditional lower bound for the time complexity of the intersection testing problem for the cases stated in Corollary B.2.2.

Corollary B.2.2. *Problem $(\circ*, t)$-intersection testing for all types $t \in \{\circ*, \circ|, \circ+, |\circ, +\circ, *\circ\}$ cannot be solved in time $\mathcal{O}((mn)^{1-\alpha})$ for any $\alpha > 0$ unless SETH fails.*

B.3 SETH-HARDNESS OF $(\circ+, \circ|)$-INTERSECTION TESTING

Let $\mathcal{A}$ be of type $\circ+$ and size m and $\mathcal{B}$ be of type $\circ|$ and size n. We define a *position* in $\mathcal{A}$ or $\mathcal{B}$ as a node on the first level of their tree representation. Thus, for any position i, $\mathcal{A}[i]$ is either a letter a or a^+; $\mathcal{B}[i]$ is either a single letter or a set of letters. A *fragment* $\mathcal{A}[i..j]$ is the regular expression given by the concatenation of the positions $\mathcal{A}[i]\mathcal{A}[i+1]\cdots\mathcal{A}[j]$ (a fragment of $\mathcal{B}$ is defined analogously). An *alignment* of a string $S \in \mathcal{L}(\mathcal{A})$ with $\mathcal{A}$ is a monotonic surjective mapping $\phi_{\mathcal{A}}$ of the positions of S to the positions of $\mathcal{A}$ (i.e. $i < j \Rightarrow \phi_{\mathcal{A}}(i) \leqslant \phi_{\mathcal{A}}(j)$) such that $S[i..j] \in \mathcal{L}(\mathcal{A}[\phi_{\mathcal{A}}(i)..\phi_{\mathcal{A}}(j)])$ for all $i \leqslant j$. Let $k \leqslant n$ be the number of positions of $\mathcal{B}$: since all the strings $S \in \mathcal{L}(\mathcal{B})$ have length k, the alignment of S with $\mathcal{B}$ is defined by the identity function, that

is, $\phi_{\mathcal{B}}(i) = i$ for all $i \in [k]$. An alignment of $\mathcal{B}$ with $\mathcal{A}$ is defined as a monotonic surjective mapping ψ of the positions of $\mathcal{B}$ to the positions of $\mathcal{A}$ such that $\mathcal{B}[i\,..\,j] \sqcap \mathcal{A}[\psi(i)\,..\,\psi(j)] \neq \emptyset$ for all $i \leqslant j \leqslant k$. We say that a fragment $\mathcal{B}[i\,..\,j]$ and a fragment $\mathcal{A}[i'\,..\,j']$ are aligned if $i' = \psi(i)$ and $j' = \psi(j)$.

Observation B.3.1. Any alignment ψ of $\mathcal{B}$ with $\mathcal{A}$ corresponds to a string $S \in \mathcal{A} \sqcap \mathcal{B}$. This is because $\mathcal{B}[1\,..\,k] \sqcap \mathcal{A}[\psi(1)\,..\,\psi(k)] \neq \emptyset$ and since ψ is surjective, this is equal to $\mathcal{B} \sqcap \mathcal{A}$; and in particular, we have that $S[i] = c_i$ if $\mathcal{A}[\psi(i)] = c_i$ or $\mathcal{A}[\psi(i)] = c_i^+$.

Conversely, note that any string $S \in \mathcal{A} \sqcap \mathcal{B}$ must have length k and must align with both $\mathcal{A}$ and $\mathcal{B}$: in this case, the alignment $\phi_{\mathcal{A}}$ of S with $\mathcal{A}$ induces an alignment ψ_S of $\mathcal{B}$ with $\mathcal{A}$ such that $\psi_S(i) = \phi_{\mathcal{A}}(i)$ for all $i \in [k]$. See Figure 49 for an example.

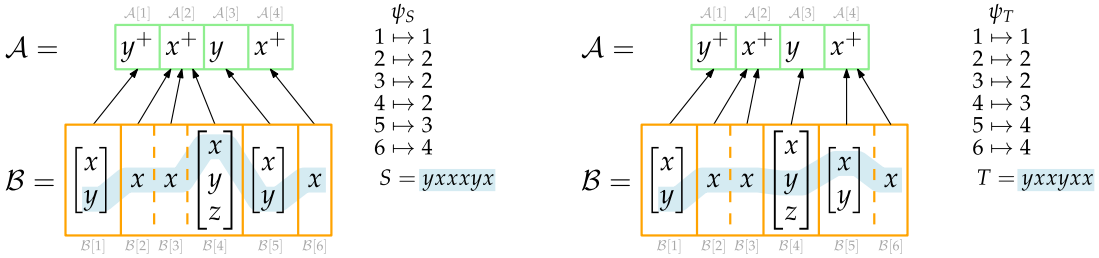


Figure 49. Two alignments of $\mathcal{A} = y^+x^+yx^+$ and $\mathcal{B} = [x|y]xx[x|y|z][x|y]x$, respectively corresponding to strings $S = yxxxyx$ (left) and $T = yxxyxx$ (right), both highlighted in $\mathcal{B}$. Dashed lines separate positions of $\mathcal{B}$ that are mapped to the same position of $\mathcal{A}$.

We next show a quadratic lower bound for $(\circ+, \circ|)$-intersection testing with a reduction from Orthogonal Vectors (OV).

Definition B.3.1 (Orthogonal Vectors)**.** Given two sets $A, B \subseteq \{0,1\}^d$ such that $|A| = M$, $|B| = N$ determine whether there exist $\alpha \in A$ and $\beta \in B$ such that α and β are orthogonal, namely, $\alpha \cdot \beta = \sum_{i=1}^{d} \alpha[i] \cdot \beta[i] = 0$.

Our reduction relies on the following restricted case of the Unbalanced Orthogonal Vectors Hypothesis (UOVH), which is known to be implied by SETH [62].

Conjecture B.3.1 (Unbalanced Orthogonal Vectors Hypothesis)**.** *For no $\epsilon > 0$ there is an algorithm for OV, restricted to $M = \Theta(N)$ and $d \leqslant N^{o(1)}$, that runs in time $\mathcal{O}(N^{2-\epsilon})$.*

Assumptions on the OV instance. We make the following assumptions on the OV instance $A = \{\alpha_1, \ldots, \alpha_M\}, B = \{\beta_1, \ldots, \beta_N\}$ with $M = \Theta(N)$, which can all be ensured via trivial linear-time modifications to any general instance.

1. M is an odd integer, N is an even integer, and $M < N$; furthermore, $N \equiv 0 \pmod 4$ (can be obtained by adding multiple copies of some vectors of A and/or B).
2. The length d of the vectors is odd (can be obtained by adding a 0 at the end of all vectors in case they have even length).
3. All vectors of set A begin and end with a 1; all vectors of B begin and end with a 0 (can be obtained by appropriately padding all the vectors at the beginning and the end).

4. A and B do not contain the vectors 1^d, 0^d, $10^{d-2}1$, and $01^{d-2}0$ (can be checked with a linear-time scan).
5. $\alpha_1 \not\perp \beta_j$ and $\alpha_M \not\perp \beta_j$ for all $j \in [N]$.
6. If there are i, j such that $\alpha_i \cdot \beta_j = 0$ then there are i', j' such that $\alpha_{i'} \cdot \beta_{j'} = 0$ and $i' \equiv j' \pmod 2$ (Assumption 3 in [31] justifies why this can be assumed w.l.o.g.).

Gadgets. Our reduction uses the constant-size alphabet $\Sigma = \{x, y, \$\}$. We begin by showing how to encode the vectors of A with regexp gadgets of type $\circ+$. Coordinate gadgets encode the entries of a vector with the following rationale: even positions correspond to gadgets that only use the letter x, while the gadgets for the odd positions only use y. The size of a gadget is determined by the value of the corresponding entry in the vector, namely, size-3 gadgets of the form ccc^+ encode entries equal to 1, and size-1 gadgets of the form c^+ encode 0 entries (where $c \in \{x, y\}$ according to the parity of the position of the vector). Formally, we thus define coordinate gadgets $C_A(v, k)$ for $v \in \{0, 1\}$ and $k \in [d]$ as follows:

$$C_A(v,k) \coloneqq \begin{cases} yyy^+ & \text{for } v = 1 \text{ and } k \equiv 1 \pmod 2 \\ xxx^+ & \text{for } v = 1 \text{ and } k \equiv 0 \pmod 2 \\ y^+ & \text{for } v = 0 \text{ and } k \equiv 1 \pmod 2 \\ x^+ & \text{for } v = 0 \text{ and } k \equiv 0 \pmod 2. \end{cases}$$

Vector gadgets $a_i^\perp$ are then defined as the concatenation of the coordinate gadgets for α_i:

$$a_i^\perp \coloneqq C_A(\alpha_i[1], 1) C_A(\alpha_i[2], 2) \cdots C_A(\alpha_i[d], d).$$

Due to Assumptions 2 and 3, all such vector gadgets begin with yyy^+ and end with yyy^+. To encode the vectors of B, we define coordinate gadgets which still associate letter y with odd positions and letter x with even positions, but this time size-1 gadgets encode the occurrences of 1, and size-3 gadgets the occurrences of 0; since $\mathcal{B}$ is of type $\circ|$, the gadgets do not contain the operator $+$ and the vectors of B are encoded with solid strings. Formally, we define the coordinate gadgets $C_B(v, k)$ for $v \in \{0, 1\}$ and $k \in [d]$ as follows:

$$C_B(v,k) \coloneqq \begin{cases} y & \text{for } v = 1 \text{ and } k \equiv 1 \pmod 2 \\ x & \text{for } v = 1 \text{ and } k \equiv 0 \pmod 2 \\ yyy & \text{for } v = 0 \text{ and } k \equiv 1 \pmod 2 \\ xxx & \text{for } v = 0 \text{ and } k \equiv 0 \pmod 2. \end{cases}$$

Vector gadgets b_j are then defined depending on the parity of j (given a fixed but arbitrary ordering of the vectors of B):

$$b_j \coloneqq \begin{cases} C_B(\beta_j[1], 1) C_B(\beta_j[2], 2) \cdots C_B(\beta_j[d], d) & \text{if } j \equiv 1 \pmod 2 \\ yyy C_B(\beta_j[1], 1) C_B(\beta_j[2], 2) \cdots C_B(\beta_j[d], d) yyy & \text{if } j \equiv 0 \pmod 2. \end{cases}$$

Due to Assumptions 2 and 3, all gadgets b_j with an odd j start and end with y^3, and those with an even j start and end with y^6. We also define another two gadgets of type $\circ+$

$$a_0^{\perp} \coloneqq (y^+x^+)^{\lfloor \frac{d}{2} \rfloor} y^+, \qquad a_{even}^{\perp} \coloneqq y^6 x^+ (y^+x^+)^{\lfloor \frac{d}{2} \rfloor - 1} y^6 \tag{138}$$

and three additional gadgets of type $\circ|$

$$b_0 \coloneqq \left(y^3\,[x|y]^3\right)^{\lfloor \frac{d}{2} \rfloor} y^3; \quad b_{even} \coloneqq y^6 x\,(yx)^{\lfloor \frac{d}{2} \rfloor - 1}\, y^6; \quad b_{odd} \coloneqq y^3 x\,(yx)^{\lfloor \frac{d}{2} \rfloor - 1}\, y^3. \tag{139}$$

Finally, we define $b_0^{(\$)} \coloneqq b_0\,[y|\$]$, $b_{even}^{(\$)} \coloneqq b_{even}\,[y|\$]$, and $b_{odd}^{(\$)} \coloneqq b_{odd}\,[y|\$]$.

Example B.3.1. Let $d = 5$, consider the vectors $\alpha_1 = 10011$, $\alpha_2 = 11001$, $\beta_1 = 00010$, $\beta_2 = 01010$, and denote by x^{3+} and y^{3+} the expressions xxx^+ and yyy^+, respectively. The associated gadgets are $a_1^{\perp} = y^{3+}x^+y^+x^{3+}y^{3+}$; $a_2^{\perp} = y^{3+}x^{3+}y^+x^+y^{3+}$; $b_1 = y^3x^3y^3xy^3$; $b_2 = y^6xy^3xy^6$. Note that $a_2^{\perp} \sqcap b_1 \neq \emptyset$, but $a_1^{\perp} \sqcap b_1 = \emptyset$: indeed, β_1 is orthogonal to α_2, but not to α_1. Furthermore, $a_0^{\perp} = y^+x^+y^+x^+y^+$; $a_{even}^{\perp} = y^6x^+y^+x^+y^6$; $b_0 = y^3\,[x|y]^3\,y^3\,[x|y]^3\,y^3$; $b_0^{(\$)} = y^3\,[x|y]^3\,y^3\,[x|y]^3\,y^3\,[y|\$]$; $b_{even}^{(\$)} = y^6xyxy^6\,[y|\$]$; and $b_{odd}^{(\$)} = y^3xyxy^3\,[y|\$]$.

Lemma B.3.2 is straightforward from the definitions of the vector gadgets $a_i^{\perp}$ and b_j.

Lemma B.3.2. *For any $i \in [M]$ and $j \in [N]$, $a_i^{\perp} \sqcap b_j \neq \emptyset$ (thus $a_i^{\perp}$ aligns with b_j) if and only if $\alpha_i \perp \beta_j$.*

Since gadgets b_j are strings, we have $a_i^{\perp} \sqcap b_j \neq \emptyset$ if and only if $b_j \in \mathcal{L}(a_i^{\perp})$, thus $\mathcal{L}(a_i^{\perp})$ contains the string b_j for each vector β_j orthogonal to α_i. We remark that, morally, $a_0^{\perp}$ and b_0 encode the vector 0^d, while b_{odd} and b_{even} encode the vector $01^{d-2}0$. This is consistent with the fact that (i) $b_0 \in \mathcal{L}(a_i^{\perp})$ and $b_j \in \mathcal{L}(a_0^{\perp})$ for all $i \in [M], j \in [N]$ (as 0^d is orthogonal to any vector), and (ii) $a_i^{\perp}$ does not align with b_{odd} or b_{even} (as by Assumption 4, A does not contain the vector $10^{d-2}1$, and thus $01^{d-2}0 \not\perp \alpha_i$ for all $i \in [M]$). Furthermore, since $\mathcal{L}(b_0)$ contains a run of the letter y, we can align $a_0^{\perp}$ with b_0^k for any $k \geqslant 1$. We summarise these properties, which are straightforward to verify from the gadget definitions, in Remark B.3.1.

Remark B.3.1. The following relations among the gadgets hold:

1. $a_0^{\perp} \sqcap (b_0)^k b_j \neq \emptyset$ for all $j \in [N]$ for all $k \geqslant 0$;
2. $a_i^{\perp} \sqcap (b_0)^k \neq \emptyset$ for all $i \in [0, M]$ for all $k \geqslant 1$;
3. $a_i^{\perp} \sqcap (b_0)^k b_j \neq \emptyset$ if and only if $\alpha_i \perp \beta_j$ for all $k \geqslant 0$;
4. $a_0^{\perp} \sqcap (b_0)^k b_{even} (b_0)^h \neq \emptyset$ and $a_0^{\perp} \sqcap (b_0)^k b_{odd} (b_0)^h \neq \emptyset$, but $a_i^{\perp} \sqcap (b_0)^k b_{even} (b_0)^h = \emptyset$ and $a_i^{\perp} \sqcap (b_0)^k b_{odd} (b_0)^h = \emptyset$ for all $i \in [M]$, for all $h, k \geqslant 0$;
5. $a_{even}^{\perp} \sqcap b_{even} \neq \emptyset$ but $a_{even}^{\perp} \sqcap b_0 = \emptyset$ and $a_{even}^{\perp} \sqcap b_{odd} = \emptyset$;

6. $a^{\perp}_{even} \sqcap b_p \neq \emptyset$ if and only if p is even;
7. Since there are exactly $d-1$ alternation of y and x in $a^{\perp}_i$ for $i \in [0, M]$, $a^{\perp}_i$ can align with at most one string b_j (including b_{even} or b_{odd} if $i = 0$); for instance, $a^{\perp}_0 \sqcap b_{even} b_0 b_{odd} = \emptyset$.

For all cases of Remark B.3.1, given a regexp $\mathcal{A}$ of type $\circ+$ and $\mathcal{B}$ of type $\circ|$, the following are equivalent: $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$, $\mathcal{A}\$ \sqcap \mathcal{B}\$ \neq \emptyset$, $\$\mathcal{A} \sqcap \$\mathcal{B} \neq \emptyset$, $\$\mathcal{A}\$ \sqcap \$\mathcal{B}\$ \neq \emptyset$. This holds even if the symbol $ is replaced by $[y|\$]$ in the regexp of type $\circ|$, or b_0 is replaced by $b^{(\$)}_0$.

Full reduction. We concatenate the gadgets encoding OV sets A and B to construct two regexps $\mathcal{A}$, $\mathcal{B}$ of types $\circ+$ and $\circ|$, respectively. Each regexp is partitioned into three fragments: $\mathcal{A} = \mathcal{A}_{pre}\mathcal{A}^{\perp}\mathcal{A}_{suf}$ and $\mathcal{B} = \mathcal{B}_{pre}\mathcal{B}^{\perp}\mathcal{B}_{suf}$. The pre and suf fragments are designed to force the alignment of a fragment of $\mathcal{A}^{\perp}$ with a fragment of $\mathcal{B}^{\perp}$ whenever we have a global alignment of $\mathcal{A}$ and $\mathcal{B}$, which in turn will imply that a pair of orthogonal vectors exists.

$\mathcal{A}^{\perp}$ and $\mathcal{B}^{\perp}$ are the central fragments of $\mathcal{A}$ and $\mathcal{B}$, respectively. They contain the gadgets $a^{\perp}_i$ and b_j, which do not appear anywhere else in $\mathcal{A}$ and $\mathcal{B}$, and are defined as follows:

$$\mathcal{A}^{\perp} \coloneqq a^{\perp}_1 \$ a^{\perp}_0 \$ a^{\perp}_2 \$ \cdots \$ a^{\perp}_{M-1} \$ a^{\perp}_0 \$ a^{\perp}_M \tag{140}$$

$$\mathcal{B}^{\perp} \coloneqq b^{(\$)}_0 b_1 \$ b^{(\$)}_0 b_2 \$ \cdots \$ b^{(\$)}_0 b_{N-1} \$ b^{(\$)}_0 b_N \$. \tag{141}$$

Remark B.3.2. All strings in the language of $\mathcal{A}^{\perp}$ contain $2M-2$ occurrences of $; all strings in the language of $\mathcal{B}^{\perp}$ contain at least N and at most 2N occurrences of $.

The two regular expressions $\mathcal{A}$ and $\mathcal{B}$ are then defined as

$$\mathcal{A} \coloneqq \mathcal{A}_{pre}\mathcal{A}^{\perp}\mathcal{A}_{suf} \tag{142}$$

$$\mathcal{A}_{pre} \coloneqq \left(a^{\perp}_0\right)^{2M+N} \left(a^{\perp}_0 \$\right)^{N-1} a^{\perp}_{even} \tag{143}$$

$$\mathcal{A}_{suf} \coloneqq \$ a^{\perp}_{even} \left(\$ a^{\perp}_0\right)^{N-1} \left(a^{\perp}_0\right)^{2M+N} \tag{144}$$

$$\mathcal{B} \coloneqq \mathcal{B}_{pre}\mathcal{B}^{\perp}\mathcal{B}_{suf} \tag{145}$$

$$\mathcal{B}_{pre} \coloneqq (b_0)^{2M+N} \left(b^{(\$)}_0 b^{(\$)}_{odd} b^{(\$)}_0 b^{(\$)}_{even}\right)^{\frac{2M+N-2}{4}} \tag{146}$$

$$\mathcal{B}_{suf} \coloneqq \left(b^{(\$)}_0 b^{(\$)}_{odd} b^{(\$)}_0 b^{(\$)}_{even}\right)^{\frac{2M+N-2}{4}} (b_0)^{2M+N}. \tag{147}$$

Remark B.3.3. All strings in the language of $\mathcal{A}$ have exactly $2M+2N-2$ occurrences of $, and all strings in the language of $\mathcal{B}$ have at least N and at most $4N+4M-4$ occurrences of $.

Recall that by Assumption 1, $N \equiv 0 \pmod 4$ and $2M \equiv 2 \pmod 4$ thus $2M+N-2 \equiv 0 \pmod 4$. Suppose that the intersection of the languages of $\mathcal{A}$ and $\mathcal{B}$ is not empty and consider a string $U \in \mathcal{A} \sqcap \mathcal{B}$. By Remark B.3.3, U has exactly $2M+2N-2$ occurrences of $. Lemma B.3.3 shows that the fragment $U[s_{\mathcal{A}} \,..\, e_{\mathcal{A}}]$ aligned with $\$\mathcal{A}^{\perp}\$$ and the fragment $U[s_{\mathcal{B}} \,..\, e_{\mathcal{B}}]$ aligned with $\mathcal{B}^{\perp}$ must overlap, and thus a fragment of $\mathcal{A}^{\perp}$ aligns with a fragment of $\mathcal{B}^{\perp}$: see also Figure 50.

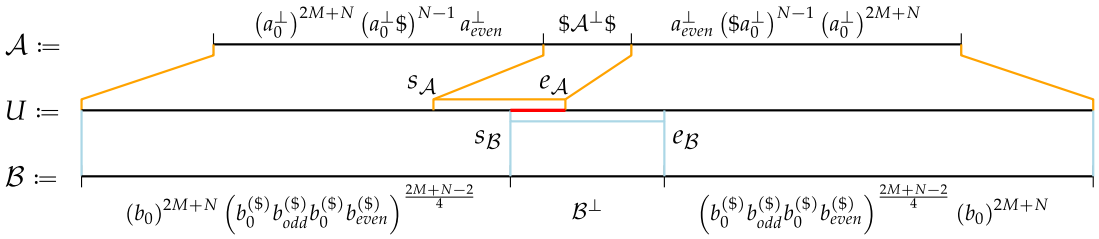


Figure 50. Visual representation of Lemma B.3.3. The blue lines are vertical to emphasise that the alignment of U with $\mathcal{B}$ is the identity function.

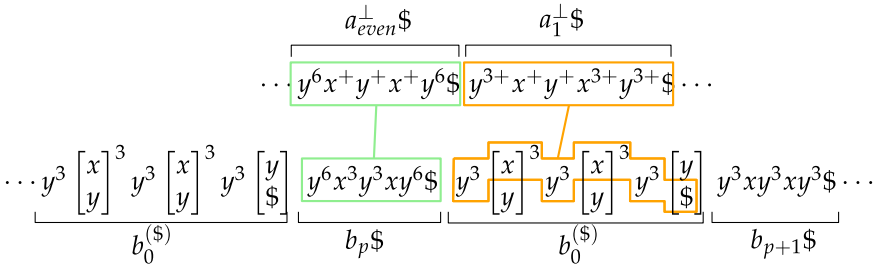


Figure 51. Example of the alignment of $a_{even}^\perp\$a_1^\perp\$$ with $b_p\$b_0^{(\$)}$, for an even p. Here $d = 5$, $\alpha_1 = 10011$, $\beta_p = 00010$, and $\beta_{p+1} = 01010$. Note that, coherently with Assumption 5, $a_1^\perp$ does not align with b_p nor with b_{p+1}.

Lemma B.3.3. *For any $U \in \mathcal{A} \sqcap \mathcal{B}$, let $U[s_\mathcal{A} \,..\, e_\mathcal{A}]$ and $U[s_\mathcal{B} \,..\, e_\mathcal{B}]$ be the fragments aligned with $\$\mathcal{A}^\perp\$$ and $\mathcal{B}^\perp$, respectively. Then $[s_\mathcal{A}, e_\mathcal{A}] \cap [s_\mathcal{B}, e_\mathcal{B}] \neq \emptyset$.*

Proof. By the definition of $\mathcal{A}$, $s_\mathcal{A}$ and $e_\mathcal{A}$ are the positions of the N-th and the $(N + 2M - 1)$-th occurrence of \$ in U, respectively. Thus the only way $[s_\mathcal{A}, e_\mathcal{A}] \cap [s_\mathcal{B}, e_\mathcal{B}]$ could be empty is if the first $N + 2M - 1$ occurrences of \$ in U are all all aligned within $\mathcal{B}_{pre}$ or symmetrically, if the last $N + 2M - 1$ occurrences are all aligned within $\mathcal{B}_{suf}$. A simple counting argument shows that this is impossible. In fact, by the definition of $\mathcal{B}$, at most $N + 2M - 2$ occurrences of \$ can be aligned within $\mathcal{B}_{pre}$; and symmetrically, within $\mathcal{B}_{suf}$. □

Lemma B.3.4. *If $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ then there exists a pair of orthogonal vectors.*

Proof. Since $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$, $\mathcal{A}$ can be aligned with $\mathcal{B}$. The general idea is to prove that for each possible alignment of $\mathcal{A}$ and $\mathcal{B}$ there exist $i \in [M], j \in [N]$ such that $a_i^\perp$ is aligned with $b_0^{(\$)} b_j$ or b_j, which by Remark B.3.1.3 and Lemma B.3.2 implies that $\alpha_i \perp \beta_j$. Let the intervals $[s_\mathcal{A}, e_\mathcal{A}]$ and $[s_\mathcal{B}, e_\mathcal{B}]$ be defined as in Lemma B.3.3. Since the intersection of these two intervals is not empty, we study the following three cases.

CASE $s_\mathcal{B} \leqslant s_\mathcal{A} < e_\mathcal{A} \leqslant e_\mathcal{B}$. Let us focus on how the extremities of $\$\mathcal{A}^\perp\$$, that is, $\$a_1^\perp$ and $a_M^\perp\$$, can be aligned inside $\mathcal{B}^\perp$. In principle, the first \$ could be either aligned with an explicit occurrence of \$ in $\mathcal{B}^\perp$ or with an "optional" occurrence in a set $[y|\$]$. The second case would imply that $\$a_1^\perp\$$ aligns with $[y|\$]b_j\$$ for some $j \in [N]$, which is a contradiction due to Assumption 5. Therefore, it must be aligned with an explicit occurrence of \$ in $\mathcal{B}^\perp$, which implies, in

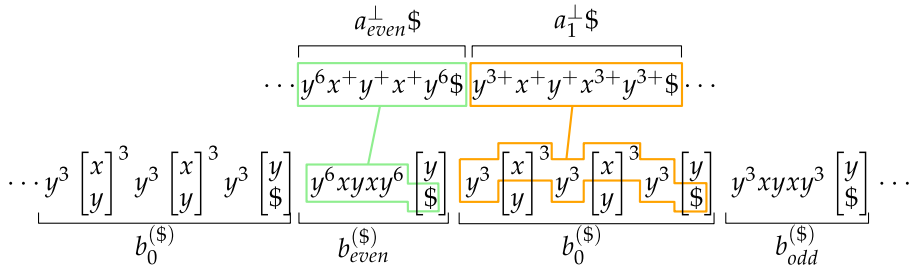
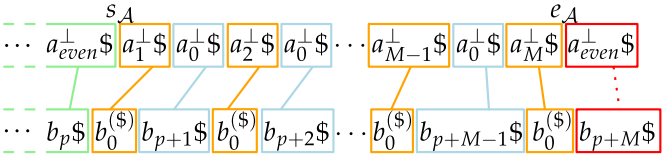


Figure 52. Example of the alignment $a^{\perp}_{even}\$a^{\perp}_1\$$ and $b^{(\$)}_{even}b^{(\$)}_0$ with $d = 5$ and $\alpha_1 = 10011$.

particular, that the whole $a^{\perp}_{even}\$a^{\perp}_1$ must be aligned within $\mathcal{B}^{\perp}$. Again by Assumption 5, $\$a^{\perp}_1\$$ cannot align with $\$b^{(\$)}_0 b_j\$$ for any $j \in [N]$; furthermore, by Remark B.3.1.6, $a^{\perp}_{even}\$$ aligns with $b_p\$$ if and only if p is even. The whole reasoning implies that $a^{\perp}_{even}\$a^{\perp}_1\$$ must be aligned with $b_p\$b^{(\$)}_0$ for some even p (see Figure 51). A similar reasoning proves that $a^{\perp}_M\$a^{\perp}_{even}\$$ must be aligned with $b^{(\$)}_0 b_q\$$ for some even q.

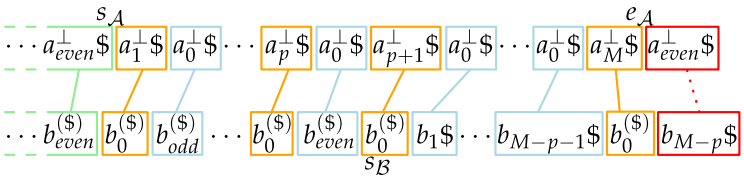
Let us study what happens in the middle part of $\mathcal{A}^{\perp}$. Fixed any $i \in [2, M-1]$, $\$a^{\perp}_i\$$ can only align with the following types of fragments of $\mathcal{B}^{\perp}$: $\$b^{(\$)}_0$ or $\$b^{(\$)}_0 b_j\$$ or $[y|\$]b_j\$$. By Lemma B.3.2 and Remark B.3.1.3, the last two cases can occur if and only if $\alpha_i \perp \beta_j$. In contrast, $\$a^{\perp}_0\$$ can align with $\$b^{(\$)}_0$ or $\$b^{(\$)}_0 b_j\$$ or $[y|\$]b_j\$$ without any implications on vector orthogonality. Now since $a^{\perp}_{even}\$a^{\perp}_1\$$ is aligned with $b_p\$b^{(\$)}_0$ for some even p, $a^{\perp}_{even}\$a^{\perp}_1\$a^{\perp}_0\$$ must be aligned with $b_p\$b^{(\$)}_0 b_{p+1}\$$. Suppose for a contradiction that $\alpha_i \not\perp \beta_{p+i}$ for all $i \in [M]$. Then $\alpha_2 \not\perp \beta_{p+2}$ implies that $a^{\perp}_{even}\$a^{\perp}_1\$a^{\perp}_0\$a^{\perp}_2\$a^{\perp}_0\$$ must be aligned with $b_p\$b^{(\$)}_0 b_{p+1}\$b^{(\$)}_0 b_{p+2}\$$, and in particular, $b^{(\$)}_0$ is aligned with $a^{\perp}_2\$$ and $b_{p+2}\$$ is aligned with $a^{\perp}_0\$$. Iterating this process, we get that $a^{\perp}_{even}\$a^{\perp}_1\$a^{\perp}_0\$a^{\perp}_2\$a^{\perp}_0\$\cdots\$a^{\perp}_{M-1}\$a^{\perp}_0\$a^{\perp}_M\$$ is aligned with $b_p\$b^{(\$)}_0 b_{p+1}\$b^{(\$)}_0 b_{p+2}\$\cdots\$b^{(\$)}_0 b_{p+M-1}\$b^{(\$)}_0$. This leads to a contradiction because it would force $a^{\perp}_{even}\$$ to align with $b_{p+M}\$$, which is impossible since $p + M$ is odd by Assumption 1. We conclude that a pair of orthogonal vectors exists.



CASE $s_{\mathcal{A}} < s_{\mathcal{B}}$. Since $\mathcal{B}[s_{\mathcal{B}}] = y$ and $\mathcal{A}[e_{\mathcal{A}}] = \$$, we have $s_{\mathcal{B}} \neq e_{\mathcal{A}}$. Since $s_{\mathcal{A}} < s_{\mathcal{B}}$, the fragment $a^{\perp}_{even}\$a^{\perp}_1\$$ must be aligned with some fragment of $\mathcal{B}$ starting to the left of $\mathcal{B}^{\perp}$, i.e. in $\mathcal{B}_{pre}$. In particular, $a^{\perp}_{even}\$a^{\perp}_1\$$ can only align with some occurrence of $b^{(\$)}_{even}b^{(\$)}_0$ (in the case where $s_{\mathcal{A}} = s_{\mathcal{B}} - 1$, it is aligned with the last occurrence of $b^{(\$)}_{even}$ in $\mathcal{B}_{pre}$ and the first occurrence of $b^{(\$)}_0$ within $\mathcal{B}^{\perp}$). Indeed, by Remark B.3.1.5, $a^{\perp}_{even}\$$ can only align with $b_{even}\$$, and by Remark B.3.1.4, $a^{\perp}_1$ does not align with $b^{(\$)}_0 b_{odd}$: see Figure 52. In principle, $a^{\perp}_{even}\$a^{\perp}_1\$a^{\perp}_0\$$ could be aligned with $b^{(\$)}_{even}b^{(\$)}_0 b^{(\$)}_{odd}b^{(\$)}_0$, and in particular, the last two positions of $b^{(\$)}_{odd}$ and the whole $b^{(\$)}_0$ (except its last posi-

tion) could be aligned with the last position of $a_0^\perp$ (which corresponds to y^+: see Equations 138, 139). However, this would imply that $a_2^\perp\$$ would have to align with $b_{even}^{(\$)}$, which is not possible by Remark B.3.1.4. By Remark B.3.1.7, $a_{even}^\perp\$a_1^\perp\$a_0^\perp\$$ cannot be aligned with $b_{even}^{(\$)}b_0^{(\$)}b_{odd}^{(\$)}b_0^{(\$)}b_{even}^{(\$)}$ or any longer fragment; therefore $a_{even}^\perp\$a_1^\perp\$a_0^\perp\$$ must be aligned with $b_{even}^{(\$)}b_0^{(\$)}b_{odd}^{(\$)}$ so that $a_2^\perp\$$ can be aligned with $b_0^{(\$)}$. Proceeding with the alignment and iterating this reasoning, we get that for any odd i such that $a_i^\perp$ is aligned within $\mathcal{B}_{pre}$, $a_i^\perp\$a_0^\perp\$a_{i+1}^\perp\$a_0^\perp\$$ must be aligned with $b_0^{(\$)}b_{odd}^{(\$)}b_0^{(\$)}b_{even}^{(\$)}$.

Therefore, for any even p, $a_{even}^\perp\$a_1^\perp\$a_0^\perp\$\cdots\$a_p^\perp\$a_0^\perp\$$ must be aligned with either (i) $b_{even}^{(\$)}(b_0^{(\$)}b_{odd}^{(\$)}b_0^{(\$)}b_{even}^{(\$)})^{\frac{p}{2}}$ or (ii) $b_{even}^{(\$)}(b_0^{(\$)}b_{odd}^{(\$)}b_0^{(\$)}b_{even}^{(\$)})^{\frac{p}{2}}b_0^{(\$)}$, implying that the smallest i such that $a_i^\perp$ is aligned with a fragment of $\mathcal{B}^\perp$ is always odd. Case (ii) occurs if and only if the last $a_0^\perp\$$ in $a_{even}^\perp\$a_1^\perp\$a_0^\perp\$\cdots\$a_p^\perp\$a_0^\perp\$$ aligns with $b_{even}^{(\$)}b_0^{(\$)}$. This forces $a_{p+1}^\perp\$$ to align with $b_1\$$, which implies that $\alpha_{p+1} \perp \beta_1$ by Lemma B.3.2. So, suppose we are in Case (i). We now study the alignment of $a_{p+1}^\perp\$a_0^\perp\$\cdots\$a_0^\perp\$a_M^\perp\$$ (that is the remaining part of $\mathcal{A}^\perp$). Suppose for the sake of contradiction that $\alpha_{p+i} \not\perp \beta_i$ for all $i \in [1, M-p]$. Since $\alpha_{p+1} \not\perp \beta_1$, then $a_{p+1}^\perp\$$ can only be aligned with $b_0^{(\$)}$ because, by Remark B.3.1.3, if $a_{p+1}^\perp\$ \sqcap b_0^{(\$)}b_1\$ \neq \emptyset$ (thus $a_{p+1}^\perp \sqcap b_0b_1 \neq \emptyset$) then $\alpha_{p+1} \perp \beta_1$. Therefore, under the assumption that $\alpha_{p+i} \not\perp \beta_i$ for each $i \in [1, M-p]$, the only possibility, by Remark B.3.1.1 and B.3.1.2, would be to align $a_{p+1}^\perp\$a_0^\perp\$\cdots\$a_0^\perp\$a_M^\perp\$$ with $b_0^{(\$)}b_1\$\cdots b_{M-p-1}\$b_0^{(\$)}$. This would force the alignment of $a_{even}^\perp\$$ with $b_{M-p}\$$, which is not possible because $M-p$ is odd by Assumption 1.



CASE $s_{\mathcal{B}} \leqslant s_{\mathcal{A}}$. The case $e_{\mathcal{A}} \leqslant e_{\mathcal{B}}$ has already been considered. If $e_{\mathcal{B}} < e_{\mathcal{A}}$, we can proceed symmetrically to the case $s_{\mathcal{A}} < s_{\mathcal{B}}$. We briefly sketch the proof. Suppose $\alpha_i \not\perp \beta_N$, for all i. Then, $b_0^{(\$)}b_N\$$ must align with $a_q^\perp\$a_0^\perp\$$, for some q. We first prove the fragment $a_{q+1}^\perp\$a_0^\perp\$a_{q+2}^\perp\$\cdots\$a_0^\perp\$a_M^\perp\$a_{even}^\perp$ must align with a fragment of $\mathcal{B}_{suf}$ and by Remarks B.3.1.2, B.3.1.4 and B.3.1.5, q must be odd. Secondly, assuming that $\alpha_i \not\perp \beta_{N-q+i}$ for $i \in [q]$, we get that $a_1^\perp\$a_0^\perp\$a_2^\perp\$\cdots\$a_0^\perp\$a_q^\perp\$a_0^\perp\$$ aligns with $b_0^{(\$)}b_{N-q+1}\$\cdots\$b_0^{(\$)}b_N\$$. This forces the alignment of $a_{even}^\perp\$$ with $b_{N-q}\$$, which is a contradiction since $N-q$ is odd. □

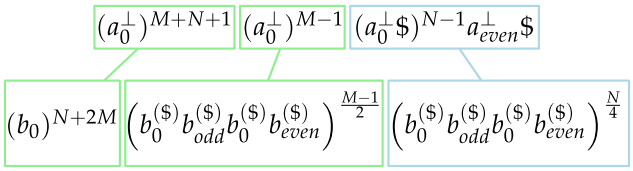
Lemma B.3.5. *If* $\exists\, j \in [N]$ *and* $i \in [M]$ *such that* $\alpha_j \perp \beta_i$, *then* $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$.

Proof. By Assumption 6, we have $i \equiv j \pmod 2$. Consider the case $i = j$. We show that in this case, $\mathcal{A}$ and $\mathcal{B}$ align, which implies that $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{B}) \neq \emptyset$. We start by showing how $\mathcal{A}_{pre}$ aligns with $\mathcal{B}_{pre}$. Using Remark B.3.1, we can prove the following facts straightforwardly:

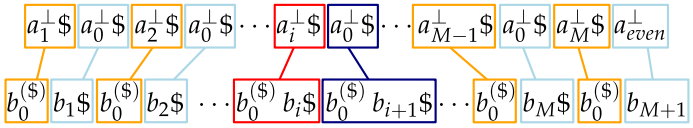
- $(a_0^\perp)^k \sqcap (b_0)^h \neq \emptyset$ for all $k \leqslant h$;

- $(a_0^\perp)^2 \sqcap b_0^{(\$)} b_{odd}^{(\$)} b_0^{(\$)} b_{even}^{(\$)} \neq \emptyset$;
- $(a_0^\perp \$)^4 \sqcap b_0^{(\$)} b_{odd}^{(\$)} b_0^{(\$)} b_{even}^{(\$)} \neq \emptyset$ and $(a_0^\perp \$)^3 a_{even}^\perp \$ \sqcap b_0^{(\$)} b_{odd}^{(\$)} b_0^{(\$)} b_{even}^{(\$)} \neq \emptyset$.

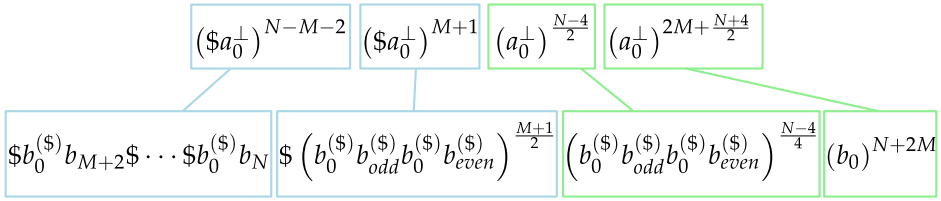
Therefore, we can align $\mathcal{A}_{pre}$ with $\mathcal{B}_{pre}$ in the following way:



We then show how to align $\mathcal{A}^\perp$ with $\mathcal{B}^\perp$. By Assumption 5 we have $i \neq 1$, and by Remark B.3.1.1 and B.3.1.2 we can align $a_k^\perp \$ a_0^\perp \$$ with $b_0^{(\$)} b_k \$$ for all $k \in [1, i-1]$. Since $\alpha_i \perp \beta_i$, by Remark B.3.1.3 we can align $a_i^\perp \$ a_0^\perp \$$ with $b_0^{(\$)} b_i \$ b_0^{(\$)} b_{i+1} \$$. Again by Assumption 5 we have $i \neq M$, thus we can align $a_k^\perp \$ a_0^\perp \$$ with $b_0^{(\$)} b_{k+1} \$$ for all $k \in [i+1, M-1]$. For $k = M$, we have to align $a_M^\perp \$ a_{even}^\perp$ with $b_0^{(\$)} b_{M+1}$, which is possible by Remark B.3.1.6 since by Assumption 1 M is odd, thus $M+1$ is even.



Similarly to the first part of the proof, we can align $\mathcal{A}_{suf}$ with the remaining part of $\mathcal{B}$.



In the general case where $i \equiv j \pmod 2$, i.e. $|i-j| = p$ for an even p (see Assumption 6), the proof is analogous: Figure 53 outlines an alignment of $\mathcal{B}$ with $\mathcal{A}$ when $j < i$. □

Since by construction the sizes of $\mathcal{A}$ and $\mathcal{B}$ are $\mathcal{O}((N+M)d) = \mathcal{O}(Nd)$, we have obtained Theorem B.3.6.

Theorem B.3.6. *The $(\circ+, \circ|)$-intersection testing problem cannot be solved in time $\mathcal{O}((mn)^{1-\alpha})$, for any $\alpha > 0$, unless SETH fails.*

B.4 LINEAR-TIME ALGORITHMS

This section presents linear-time algorithms for solving the intersection testing problem for all combinations of types of depth-2 homogeneous regular expressions except for those listed in Corollaries B.2.1 and B.2.2 and Theorem B.3.6.

Theorem B.4.1. *The $(\circ|, t)$-intersection testing problem can be solved in $\mathcal{O}(m+n)$ time for any type $t \in \{\circ|, |\circ, +\circ\}$.*

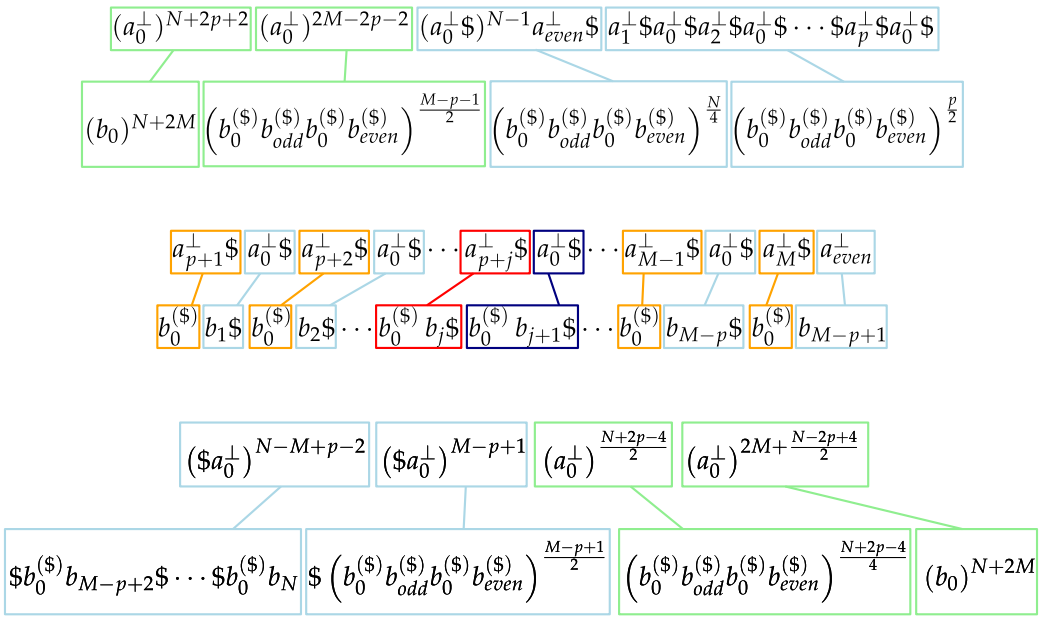


Figure 53. Scheme of an alignment of $\mathcal{A}$ and $\mathcal{B}$ in the case $\alpha_i \perp \beta_j$ with $j < i$ and $i \equiv j \pmod 2$: $p \in [M]$ is such that $p + j = i$ (in particular, p is an even number).

Proof. Let $\mathcal{A}$ be a regexp of type $\circ|$ of size m, and consider its tree representation. For each node i on the first level, we denote by $\mathcal{A}_i$ the set of leaves descending from i. We can represent $\mathcal{A}$ as the concatenation of these sets of letters $\mathcal{A} = \mathcal{A}_1 \circ \mathcal{A}_2 \circ \cdots \circ \mathcal{A}_k$. Given $\mathcal{B}$ a regexp of type t and size n, we propose different solutions depending on its type t.

$t = \circ|$. We can represent $\mathcal{B}$ as a concatenation of sets of letters $\mathcal{B} = \mathcal{B}_1 \circ \mathcal{B}_2 \circ \cdots \circ \mathcal{B}_h$. We have that $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ if and only if $k = h$ and $\mathcal{A}_i \cap \mathcal{B}_i \neq \emptyset$ for all $i \in [k]$, which can be checked in $\mathcal{O}(m+n)$ total time by radix-sorting $\mathcal{A}_i$ together with $\mathcal{B}_i$ for all $i \in [k]$. In particular, if $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ we can express $\mathcal{A} \sqcap \mathcal{B}$ as $(\mathcal{A}_1 \cap \mathcal{B}_1) \circ (\mathcal{A}_2 \cap \mathcal{B}_2) \circ \cdots \circ (\mathcal{A}_k \cap \mathcal{B}_h)$, i.e. we can represent the intersection as a regexp of type $\circ|$ in $\mathcal{O}(m+n)$ time.

$t = |\circ$. In this case, $\mathcal{B}$ is of the form $[T_1|T_2|\cdots|T_h]$ for some nonempty strings T_j. Remark that if $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ then there exists T_i such that $|T_i| = k$. Therefore, with a linear preprocessing we can discard the strings in $\mathcal{B}$ that have length different from k. This leads to a set $\mathcal{B}'$ of strings all of the same length, which is known in the literature as a segment of a Generalised Degenerate (GD) string [11]. Since regexps of type $\circ|$ are known as indeterminate strings, which in turn are a special case of GD strings, we can use the algorithm of [11] for GD strings intersection to solve the problem in $\mathcal{O}(m+n)$ time.

$t = +\circ$. In this case, $\mathcal{B}$ is of the form $[T]^+$ for some nonempty string T of length n. Remark that if $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ then $k = \ell n$ for some $1 \leqslant \ell \leqslant k$. Therefore, to solve the intersection testing problem, we first check if $k = \ell n$, and then answer a membership query for T^ℓ and $\mathcal{A}$ using the linear-time algorithm proposed in [31]. □

Theorem B.4.2. *The* $(\circ+, t)$*-intersection testing problem can be solved in* $\mathcal{O}(m+n)$ *time for any type* $t \in \{\circ+, |\circ, +\circ\}$.

Proof. Let $\mathcal{A}$ be a regexp of type $\circ+$ and size m. The following transformations preserve the language generated by $\mathcal{A}$: $c^+c^\ell \equiv c^\ell c^+$ and $(c^+)^\ell \equiv c^{\ell-1}c^+$ for any $c \in \Sigma$ and $\ell \in \mathbb{N}$. From now on, we denote by $c^{\ell+}$ the regexp $c^{\ell-1}c^+$. Applying the above rule, we can rewrite $\mathcal{A} = a_1^{x_1} \circ \ldots \circ a_k^{x_k}$ where $x_i \in \{\ell_i, \ell_i+\}$, $\ell_i \in [m]$ and $a_i \neq a_{i+1}$ for all $i \in [k-1]$, $k \leqslant m$, and we can obviously represent each $a_i^{x_i}$ with a pair (a_i, x_i). For example, the regexp $b^+bc^+c^+a$ of Figure 48 can be rewritten as $b^{2+}c^{2+}a$.

Given $\mathcal{B}$ a regexp of type t, we propose different linear-time algorithms depending on t.

$t = \circ+$. Let $\mathcal{B}$ be of type $\circ+$ and consider its decomposition $\mathcal{B} = b_1^{y_1} \circ \ldots \circ b_h^{y_h}$. We have $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ if and only if $k = h$ and $a_i^{x_i} \sqcap b_i^{y_i} \neq \emptyset$ for all $i \in [k]$. A necessary condition for the intersection of $a_i^{x_i}$ and $b_i^{y_i}$ to be nonempty is that $a_i = b_i = c$; furthermore, we can divide the following cases, each of which can be checked in $\mathcal{O}(1)$ time:

1. $x_i = x, y_i = y$. Then $c^x \sqcap c^y = c^y$ if and only if $x = y$ ($\emptyset$ otherwise);
2. $x_i = x+, y_i = y$. Then $c^{x+} \sqcap c^y = c^y$ if and only if $y \geqslant x$ ($\emptyset$ otherwise);
3. $x_i = x+, y_i = y+$. Then $c^{x+} \sqcap c^{y+} = c^{\max\{x,y\}+}$.

It then suffices to scan $\mathcal{A}$ and $\mathcal{B}$ in parallel and test the intersection of $a_i^{x_i}$ with $b_i^{y_i}$ by checking Cases 1-3 to solve the problem in $\mathcal{O}(m+n)$ time. If $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$, it holds that $\mathcal{A} \sqcap \mathcal{B} = (a_1^{x_1} \sqcap b_1^{y_1}) \circ (a_2^{x_2} \sqcap b_2^{y_2}) \cdots (a_k^{x_k} \sqcap b_k^{y_k})$, thus we can express the intersection as a regexp of type $\circ+$.

$t = |\circ$. $\mathcal{B}$ is of the form $[T_1|T_2|\cdots|T_h]$ for some nonempty strings T_j. Decomposing each T_j into maximal runs of single letters and checking Cases 1 and 2 determines if $T_j \in \mathcal{L}(\mathcal{A})$ in $\mathcal{O}(|T_j|)$ time, thus solving $(+\circ, |\circ)$-intersection testing in $\mathcal{O}(m+n)$ total time.

$t = +\circ$. $\mathcal{B}$ equals $[T]^+$ for some nonempty string T of length n. We decompose T into maximal single-letter runs to obtain $T = b_1^{\ell_1} \cdots b_h^{\ell_h}$ with $\ell_i \geqslant 1$ and $b_i \neq b_{i+1}$ for $i \in [h-1]$, and consider the decomposition of $\mathcal{A} = a_1^{x_1} \cdots a_k^{x_k}$. If $h > k$, there are strictly more single-letter runs in any string in $\mathcal{L}(\mathcal{B})$ than in any string in $\mathcal{L}(\mathcal{A})$, which implies that the intersection is empty. Therefore, we can assume that $h \leqslant k$. Let us first consider the case where $b_1 \neq b_h$, that is, T starts and ends with runs of different letters. We compare the decompositions of $\mathcal{A}$ and T from left to right. Clearly, a necessary condition for the intersection to be nonempty is that $a_1 = b_1$ and $a_j = b_{\rho(j)}$ for all $j > 1$, where $\rho(j) = ((j-1) \mod h) + 1$. In particular, let $b_{\rho(j)}^{\ell_{\rho(j)}} = c^y$: if $a_j^{x_j} = c^{x+}$, the intersection is nonempty if and only if $x \leqslant y$; otherwise, if $a_j^{x_j} = c^x$, it is nonempty if and only if $x = y$ (see Cases 1-2 above). If the intersection between $a_j^{x_j}$ and $b_{\rho(j)}^{\ell_{\rho(j)}}$ is successful, we continue comparing $a_{j+1}^{x_{j+1}}$ and $b_{\rho(j+1)}^{\ell_{\rho(j+1)}}$. The last step is to compare $a_k^{x_k}$ with $b_h^{\ell_h}$ to decide if $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ (if $\rho(k) \neq h$ then the intersection is for sure empty). Since $k = \mathcal{O}(n)$ in the worst case, this algorithm takes $\mathcal{O}(m+n)$ time.

Now consider the case $b_1 = b_h$, i.e. T starts and ends with the same letter. The algorithm is the same as the previous case, except every time we need to check the intersection of $b_h^{\ell_h}$ with $a_j^{x_j}$ for some $j < k$, we instead check the intersection of $a_j^{x_j}$ with $b_h^{\ell_h} b_1^{\ell_1} = b_h^{\ell_h+\ell_1}$, and move on to checking the intersection of $a_{j+1}^{x_{j+1}}$ with $b_2^{\ell_2}$ at the next step. This modification does not change the complexity of the algorithm, which still requires $O(m+n)$ time. □

Theorem B.4.3. *The* (t,u)*-intersection testing problem can be solved in* $O(m+n)$ *time for all types* $t,u \in \{|\circ, +\circ\}$.

Proof. $(|\circ, |\circ)$. Both $\mathcal{A}$ and $\mathcal{B}$ are finite dictionaries of nonempty strings: $\mathcal{A} = [S_1|S_2|\cdots|S_k]$, $\mathcal{B} = [T_1|T_2|\cdots|T_h]$. Therefore, testing if $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ is equivalent to checking whether $S_i = T_j$ for some $1 \leqslant i \leqslant k, 1 \leqslant j \leqslant h$. This can be easily done in $O(m+n)$ time by radix-sorting the strings of $\mathcal{A}$ and $\mathcal{B}$ all together, then comparing pairs of consecutive strings in the sorted sequence. Since, after each string comparison, one of the two elements is discarded (unless the two strings are equal, in which case we conclude that $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$), comparing the strings letter by letter gives $O(m+n)$ total time.

$(+\circ, |\circ)$. Let $\mathcal{A}$ be a regexp of type $+\circ$ and $\mathcal{B}$ of type $|\circ$. Then, $\mathcal{A}$ is of the form $[T]^+$ and $\mathcal{B}$ is of the form $[T_1|T_2|\cdots|T_h]$ for some nonempty strings T_j. Since $T_j \in \mathcal{L}(\mathcal{A})$ if and only if $T_j = T^k$ for some $k \geqslant 1$, and this condition can be verified in $O(|T_j|)$ time with a linear scan of T_j, computing the intersection requires $O(m+n)$ total time.

$(+\circ, +\circ)$. We have $\mathcal{A} = [T_1]^+$ and $\mathcal{B} = [T_2]^+$ for some nonempty strings T_1, T_2. We say that a nonempty string D is a divisor of T if $T = D^i$ for some $i \geqslant 1$. A sufficient condition for $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ is that $T_1 = D^i$ and $T_2 = D^j$ for the same nonempty string D (i.e. D is a common divisor), and in particular, $[D]^{\operatorname{lcm}(i,j)+} \subseteq \mathcal{A} \sqcap \mathcal{B}$. A necessary condition for $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ is that i, j exist such that $T_1^i = T_2^j$; using Theorem 1 in [177], we can deduce that if this is the case, T_1 and T_2 have a common divisor. Therefore, to decide whether the intersection is not empty, it is sufficient and necessary to check if T_1 and T_2 have a common divisor. This can be done in $O(m+n)$ time using Theorem 2 in [177]. □

Theorem B.4.4. *The* $(*\circ, t)$*-intersection testing problem can be solved in* $O(m+n)$ *time for all types* $t \in \{\circ|, \circ+, |\circ, +\circ, *\circ\}$.

Proof. Let $\mathcal{A} = [T]^*$ of type $*\circ$, and define $\mathcal{A}' = [T]^+$ of type $+\circ$. For all regular expressions $\mathcal{B}$ of type $t \in \{\circ|, \circ+, |\circ, +\circ\}$, we have that $\epsilon \notin \mathcal{L}(\mathcal{B})$, therefore $\mathcal{A} \sqcap \mathcal{B} = \mathcal{A}' \sqcap \mathcal{B}$. Thus we can reduce $(*\circ, t)$-intersection to $(+\circ, t)$-intersection for all types $t \in \{\circ|, \circ+, |\circ, +\circ\}$ and obtain the statement from Theorems B.4.1, B.4.2 and B.4.3. The remaining case of $(*\circ, *\circ)$-intersection is trivial since $\epsilon \in \mathcal{A} \sqcap \mathcal{B}$ for all $\mathcal{A}, \mathcal{B}$ of type $*\circ$. Furthermore, to compute explicitly the whole intersection, one can consider $\mathcal{A}'$, $\mathcal{B}'$ obtained for $\mathcal{A}$, $\mathcal{B}$ by replacing the root label $*$ with $+$, and then apply the algorithm for $(+\circ, +\circ)$-intersection described in Theorem B.4.3. □

Intersection testing for the remaining types of regular expressions is always solvable in linear time, independently of the type of the other regexp, as proved in Theorems B.4.5 and B.4.6.

Theorem B.4.5. *The* $(+|, t)$*-intersection testing problem and the* $(*|, t)$*-intersection testing problem can be solved in* $\mathcal{O}(m+n)$ *time for any type* t *of homogeneous regexp of depth* 2*.*

Proof. Let $\mathcal{A} = [c_1|\cdots|c_m]^+$ be of type $+|$. In this case we have $\mathcal{L}(\mathcal{A}) = \Sigma_{\mathcal{A}}^+$, and analogously, when $\mathcal{A}$ is of type $*|$, its language is $\Sigma_{\mathcal{A}}^*$. To solve intersection testing for $\mathcal{A}$ of type $+|$ or $*|$ and any other homogeneous depth-2 regular expression $\mathcal{B}$, it thus suffices to check whether $\mathcal{L}(\mathcal{B})$ contains a string over the alphabet $\Sigma_{\mathcal{A}}$ (including the empty string if $\mathcal{A}$ is of type $*|$). Given a letter c in $\Sigma_{\mathcal{A}} \cup \Sigma_{\mathcal{B}}$, we can check in constant time whether c belongs to $\Sigma_{\mathcal{A}}$ by radix-sorting $\Sigma_{\mathcal{A}}$ and $\Sigma_{\mathcal{B}}$ together, keeping track of which letters belong to either set and removing duplicates, and storing a map for the rank of c in $\Sigma_{\mathcal{A}} \cup \Sigma_{\mathcal{B}}$. This provides an $\mathcal{O}(m+n)$-time solution for all types listed below (the types $t \in \{*+, +*\}$ were already considered in Corollary B.2.1), as a linear scan of $\mathcal{B}$ and/or of the sorted sequence $\Sigma_{\mathcal{A}} \cup \Sigma_{\mathcal{B}}$ then suffices to solve the problem. We provide more details for the case where $\mathcal{A}$ is of type $+|$: the only difference when $\mathcal{A}$ is of type $*|$ is having to consider also the empty string ϵ.

$t \in \{+|, *|\}$. It suffices to scan the sorted sequence to check whether $\Sigma_{\mathcal{A}} \cap \Sigma_{\mathcal{B}} \neq \emptyset$.

$t \in \{\circ+, +\circ, *\circ\}$. The intersection is nonempty if and only if $\Sigma_{\mathcal{B}} \subseteq \Sigma_{\mathcal{A}}$, which again can be checked by scanning the sorted sequence once.

$t = \circ*$. The intersection is nonempty if all letters occurring in $\mathcal{B}$ without a $*$ (i.e. direct children of the root in the tree representation) belong to $\Sigma_{\mathcal{A}}$.

$t \in \{|*, |+\}$. Here, $\mathcal{B}$ is a set of single-letter runs, it thus suffices to check whether the letter of one of such runs is in $\Sigma_{\mathcal{A}}$.

$t \in \{\circ|, |\circ\}$. In the first case, $\mathcal{B}$ is a concatenation of finite sets of letters, and it suffices to check whether in each set there is at least one letter that belongs to $\mathcal{A}$. In the second case, $\mathcal{B}$ is a finite dictionary of nonempty strings, and it suffices to check whether the alphabet of at least one of them is contained in $\Sigma_{\mathcal{A}}$. □

Theorem B.4.6. *The* $(|+, t)$*-intersection testing problem and the* $(|*, t)$*-intersection testing problem can be solved in* $\mathcal{O}(m+n)$ *time for any type* t *of homogeneous regexp of depth* 2*.*

Proof. Let $\mathcal{A}$ be of type $|+$. Then, $\mathcal{A}$ is of the form $[T_1|\cdots|T_m]$, with $T_j = c_j$ or $T_j = c_j^+$ for some $c_j \in \Sigma$ (i.e. T_j is either a single letter or a single-letter run). Let $\Sigma_{\mathcal{A}} = \{c_1, ..., c_k\}$ be the set of letters used by $\mathcal{A}$. Let us consider all possible types t for $\mathcal{B}$. The cases $t \in \{+*, *+, +|, *|\}$ were considered in Corollary B.2.1 and Theorem B.4.5. Solutions for the other types follow.

$t \in \{|+, |*\}$. $\mathcal{B}$ is a disjunction of runs, it thus suffices to check whether $\Sigma_{\mathcal{A}} \cap \Sigma_{\mathcal{B}} \neq \emptyset$, proceeding as in Theorem B.4.5.

$t = |\circ$. $\mathcal{B}$ is a dictionary $[S_1|\cdots|S_h]$, where each S_i is a nonempty string. We can proceed as in case $(|\circ, |\circ)$ of Theorem B.4.3.

$t = \circ|$. $\mathcal{B}$ is of the form $\mathcal{B}_1 \circ \mathcal{B}_2 \circ \cdots \circ \mathcal{B}_h$, where each $\mathcal{B}_i$ is a set of letters. Let $\Sigma_{\mathcal{A}}^{(+)} \subseteq \Sigma_{\mathcal{A}}$ be the set of letters that occur in $\mathcal{A}$ with a $+$: the intersection is nonempty if and only if $\mathcal{L}(\mathcal{B})$ contains a run of length h of some letter in $\Sigma_{\mathcal{A}}^{(+)}$.

To check this condition, we scan $\mathcal{B}$ maintaining a counter for each letter in $\Sigma_{\mathcal{B}}$ to find the subset $\Sigma_{\mathcal{B}}^{(h)}$ of letters that occur h times, corresponding to all single-letter runs of $\mathcal{L}(\mathcal{B})$. If $\Sigma_{\mathcal{B}}^{(h)}$ is nonempty, we radix-sort it with $\Sigma_{\mathcal{A}}^{(+)}$ and scan the sorted sequence to find the answer in $\mathcal{O}(m+n)$ total time.

$t = \circ*$. $\mathcal{B}$ is a concatenation of symbols b_i or b_i^* with $b_i \in \Sigma_{\mathcal{B}}$. Then, $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ if and only if either (i) there is only one letter $b \in \Sigma_{\mathcal{B}}$ occurring in $\mathcal{B}$ without $*$, and $b \in \Sigma_{\mathcal{A}}$; or (ii) no letter occurs in $\mathcal{B}$ without $*$, and $\Sigma_{\mathcal{A}} \cap \Sigma_{\mathcal{B}} \neq \emptyset$. Both of these conditions can be trivially checked in $\mathcal{O}(m+n)$ time.

$t = \circ+$. $\mathcal{B}$ can be made into the form $b_1^{y_1} \cdots b_h^{y_h}$ using the transformation in Theorem B.4.2, where $y_i \in \{\ell_i, \ell_i+\}$, $\ell_i \in [m]$ and $b_i \neq b_{i+1}$ for all $i \in [h-1]$. Then, $\mathcal{A} \sqcap \mathcal{B} \neq \emptyset$ if and only if $h = 1$ and $b_1 \in \Sigma_{\mathcal{A}}$.

$t \in \{+\circ, *\circ\}$. $\mathcal{B}$ is of the form $[T]^+$, (resp. $[T]^*$) for some nonempty string T. For the intersection to be nonempty, we need that $T = c^\ell$ for some $c \in \Sigma_{\mathcal{A}}$, with the further constraint $\ell = 1$ if c appears in $\mathcal{A}$ without the operator $+$ (resp. $*$). This can be trivially checked in $\mathcal{O}(m+n)$ time.

Analogous solutions work for $\mathcal{A}$ of type $|*$, the only difference being having to consider the possibility that $\epsilon \in \mathcal{L}(\mathcal{A})$. □

B.5 CONCLUSIONS

We have provided a full dichotomy for the complexity of intersection testing of depth-2 homogeneous regular expressions, showing that in most cases the problem is solvable in linear time. Moreover, for such cases, we showed that the whole intersection can be computed explicitly as either a regular expression or a set, still in linear time. The main result presented in the chapter is that, even though $(\circ|, \circ)$ and $(\circ+, \circ)$-intersection testing are solvable in linear time, there exists a quadratic lower bound, conditioned on SETH, for $(\circ+, \circ|)$-intersection testing. A natural direction for future investigation is to determine the complexity of intersection testing for homogeneous regular expressions of higher depth, and, more generally, answer the question: given a homogeneous regular expression of type t, for which types t' (if any) a conditional quadratic lower bound for (t, t')-intersection testing exists?

BIBLIOGRAPHY

[1] A. Abboud and K. Bringmann. Tighter connections between formula-sat and shaving logs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.ICALP.2018.8.

[2] K. R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987. doi:10.1137/0216067.

[3] H. E. Aguirre, H. Okazaki, and Y. Fuwa. An evolutionary multiobjective approach to design highly non-linear boolean functions. In *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007*, pages 749–756. ACM, 2007. doi:10.1145/1276958.1277112.

[4] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.

[5] J. N. Alanko, E. Biagi, S. J. Puglisi, and J. Vuohtoniemi. Subset wavelet trees. In *21st International Symposium on Experimental Algorithms (SEA)*, volume 265 of *LIPIcs*, pages 4:1–4:14, 2023. doi:10.4230/LIPICS.SEA.2023.4.

[6] A. Alhazov, R. Freund, and S. Verlan. P systems working in maximal variants of the set derivation mode. In A. Leporati, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing - 17th International Conference, CMC 2016, Milan, Italy, July 25-29, 2016, Revised Selected Papers*, volume 10105 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2016. doi:10.1007/978-3-319-54072-6_6.

[7] A. Alhazov, R. Freund, S. Ivanov, and S. Verlan. Variants of derivation modes for which catalytic P systems with one catalyst are computationally complete. *J. Membr. Comput.*, 3(4):233–245, 2021. doi:10.1007/S41965-021-00085-Z.

[8] A. Alhazov, R. Freund, and S. Ivanov. On the spectrum between reaction systems and string rewriting. *Nat. Comput.*, 23(2):159–175, 2024. doi:10.1007/S11047-024-09986-1.

[9] E. Allender, L. Hellerstein, P. McCabe, T. Pitassi, and M. E. Saks. Minimizing DNF formulas and ac^0_d circuits given a truth table. In *21st Annual IEEE Conference on Computational Complexity (CCC 2006), 16-20 July 2006, Prague, Czech Republic*, pages 237–251. IEEE Computer Society, 2006. doi:10.1109/CCC.2006.27.

[10] M. Alzamel, L. A. K. Ayad, G. Bernardini, R. Grossi, C. S. Iliopoulos, N. Pisanti, S. P. Pissis, and G. Rosone. Degenerate string comparison and applications. In *18th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 113 of *LIPIcs*, pages 21:1–21:14, 2018. doi:10.4230/LIPICS.WABI.2018.21.

[11] M. Alzamel, L. A. K. Ayad, G. Bernardini, R. Grossi, C. S. Iliopoulos, N. Pisanti, S. P. Pissis, and G. Rosone. Comparing degenerate strings. *Fundam. Informaticae*, 175(1-4): 41–58, 2020. doi:10.3233/FI-2020-1947.

[12] B. Aman and G. Ciobanu. Solving subset sum and SAT problems by reaction systems. *Nat. Comput.*, 23(2):177–187, 2024. doi:10.1007/S11047-024-09972-7.

[13] A. Amir and M. Itzhaki. Reconstructing general matching graphs. In S. Inenaga and S. J. Puglisi, editors, *35th Annual Symposium on Combinatorial Pattern Matching, CPM 2024, June 25-27, 2024, Fukuoka, Japan*, volume 296 of *LIPIcs*, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.CPM.2024.2.

[14] P. Antoniou, M. Crochemore, C. S. Iliopoulos, I. Jayasekera, and G. M. Landau. Conservative string covering of indeterminate strings. In *Proceedings of the Prague Stringology Conference*, pages 108–115, 2008.

[15] K. Aoyama, Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda. Faster online elastic degenerate string matching. In *29th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 105 of *LIPIcs*, pages 9:1–9:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICS.CPM.2018.9.

[16] E. Arrighi, H. Fernau, S. Hoffmann, M. Holzer, I. Jecker, M. de Oliveira Oliveira, and P. Wolf. On the complexity of intersection non-emptiness for star-free language classes. In *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 213 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.FSTTCS.2021.34.

[17] R. Ascone, G. Bernardini, A. Conte, M. Equi, E. Gabory, R. Grossi, and N. Pisanti. A unifying taxonomy of pattern matching in degenerate strings and founder graphs. In *24th International Workshop on Algorithms in Bioinformatics, (WABI)*, volume 312 of *LIPIcs*, pages 14:1–14:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.WABI.2024.14.

[18] R. Ascone, G. Bernardini, E. Formenti, F. Leiter, and L. Manzoni. Pure reaction automata. *Nat. Comput.*, 23(2):189–204, 2024. doi:10.1007/S11047-024-09980-7.

[19] R. Ascone, G. Bernardini, and L. Manzoni. Fixed points and attractors of additive reaction systems. *Nat. Comput.*, 23(2):205–215, 2024. doi:10.1007/S11047-024-09977-2.

[20] R. Ascone, G. Bernardini, and L. Manzoni. Fixed points and attractors of reactantless and inhibitorless reaction systems. *Theor. Comput. Sci.*, 984:114322, 2024. doi:10.1016/J.TCS.2023.114322.

[21] R. Ascone, G. Bernardini, A. Conte, V. Guerrini, and G. Punzi. Are depth-2 regular expressions hard to intersect? *arXiv preprint*, 2025. doi:10.48550/arXiv.2507.03593.

[22] R. Ascone, G. Bernardini, F. Leiter, and L. Manzoni. Chemical pure reaction automata in maximally parallel manner. *J. Membr. Comput.*, pages 1–10, 2025. doi:10.1007/s41965-024-00176-7.

[23] R. Ascone, G. Bernardini, and L. Manzoni. Cycles and global attractors of reactantless and inhibitorless reaction systems. *Theor. Comput. Sci.*, 1045:115300, 2025. doi:10.1016/J.TCS.2025.115300.

[24] R. Ascone, L. Mariot, L. Manzoni, and G. Pietropolli. Evolving cryptographic boolean functions with reaction systems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '25 Companion, page 195–198, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714641. doi:10.1145/3712255.3726685. URL https://doi.org/10.1145/3712255.3726685.

[25] R. Asthana, N. Verma, and R. Ratan. Generation of boolean functions using genetic algorithm for cryptographic applications. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 1361–1366. IEEE, 2014. doi:10.1109/IAdCC.2014.6779525.

[26] S. Azimi. Steady states of constrained reaction systems. *Theor. Comput. Sci.*, 701:20–26, 2017. doi:10.1016/j.tcs.2017.03.047.

[27] S. Azimi, B. Iancu, and I. Petre. Reaction system models for the heat shock response. *Fundam. Informaticae*, 131(3-4):299–312, 2014. doi:10.3233/FI-2014-1016.

[28] S. Azimi, C. Gratie, S. Ivanov, L. Manzoni, I. Petre, and A. E. Porreca. Complexity of model checking for reaction systems. *Theor. Comput. Sci.*, 623:103–113, 2016. doi:10.1016/j.tcs.2015.11.040.

[29] J. A. Baaijens, P. Bonizzoni, C. Boucher, G. D. Vedova, Y. Pirola, R. Rizzi, and J. Sirén. Computational graph pangenomics: a tutorial on data structures and their applications. *Nat. Comput.*, 21(1):81–108, 2022. doi:10.1007/s11047-022-09882-6.

[30] A. Backurs and P. Indyk. Which regular expression patterns are hard to match? *CoRR*, abs/1511.07070, 2015.

[31] A. Backurs and P. Indyk. Which regular expression patterns are hard to match? In *57th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 457–466, 2016. doi:10.1109/FOCS.2016.56.

[32] S. Bala. Intersection of regular languages and star hierarchy. In *29th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 2380 of *Lecture Notes in Computer Science*, pages 159–169. Springer, 2002. doi:10.1007/3-540-45465-9_15.

[33] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar. cvc5: A versatile and industrial-strength SMT solver. In *28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022. doi:10.1007/978-3-030-99524-9_24.

[34] R. Barbuti, R. Gori, F. Levi, and P. Milazzo. Investigating dynamic causalities in reaction systems. *Theor. Comput. Sci.*, 623:114–145, 2016. doi:10.1016/j.tcs.2015.11.041.

[35] R. Barbuti, A. Bernasconi, R. Gori, and P. Milazzo. Computing preimages and ancestors in reaction systems. In D. Fagan, C. Martín-Vide, M. O'Neill, and M. A. Vega-Rodríguez, editors, *Theory and Practice of Natural Computing - 7th International Conference, TPNC 2018, Dublin, Ireland, December 12-14, 2018, Proceedings*, volume 11324 of *Lecture Notes in Computer Science*, pages 23–35. Springer, 2018. doi:10.1007/978-3-030-04070-3_2.

[36] R. Barbuti, R. Gori, F. Levi, and P. Milazzo. Generalized contexts for reaction systems: definition and study of dynamic causalities. *Acta Informatica*, 55(3):227–267, 2018. doi:10.1007/S00236-017-0296-3.

[37] R. Barbuti, A. Bernasconi, R. Gori, and P. Milazzo. Characterization and computation of ancestors in reaction systems. *Soft Comput.*, 25(3):1683–1698, 2021. doi:10.1007/S00500-020-05300-0.

[38] R. Barbuti, P. Bove, R. Gori, D. P. Gruska, F. Levi, and P. Milazzo. Encoding threshold boolean networks into reaction systems for the analysis of gene regulatory networks. *Fundam. Informaticae*, 179(2):205–225, 2021. doi:10.3233/FI-2021-2021.

[39] R. Barbuti, R. Gori, and P. Milazzo. Encoding boolean networks into reaction systems for investigating causal dependencies in gene regulation. *Theor. Comput. Sci.*, 881:3–24, 2021. doi:10.1016/J.TCS.2020.07.031.

[40] C. Barton. On the average-case complexity of pattern matching with wildcards. *Theor. Comput. Sci.*, 922:37–45, 2022. doi:10.1016/J.TCS.2022.04.009.

[41] G. Bathie, P. Charalampopoulos, and T. Starikovskaya. Pattern matching with mismatches and wildcards. In T. M. Chan, J. Fischer, J. Iacono, and G. Herman, editors, *32nd Annual European Symposium on Algorithms, ESA 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 308 of *LIPIcs*, pages 20:1–20:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.ESA.2024.20.

[42] G. Battaglia, D. Cangelosi, R. Grossi, and N. Pisanti. Masking patterns in sequences: A new class of motif discovery with don't cares. *Theor. Comput. Sci.*, 410(43):4327–4340, 2009. doi:10.1016/J.TCS.2009.07.014.

[43] G. T. Becker. The gap between promise and reality: On the insecurity of XOR arbiter pufs. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 535–555. Springer, 2015. doi:10.1007/978-3-662-48324-4_27.

[44] P. K. Behera and S. Gangopadhyay. An improved hybrid genetic algorithm to construct balanced boolean function with optimal cryptographic properties. *Evol. Intell.*, 15(1): 639–653, 2022. doi:10.1007/S12065-020-00538-X.

[45] G. Bernardini, N. Pisanti, S. P. Pissis, and G. Rosone. Pattern matching on elastic-degenerate text with errors. In *24th International Symposium on String Processing and Information Retrieval (SPIRE)*, volume 10508 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 2017. doi:10.1007/978-3-319-67428-5_7.

[46] G. Bernardini, P. Gawrychowski, N. Pisanti, S. P. Pissis, and G. Rosone. Even faster elastic-degenerate string matching via fast matrix multiplication. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *LIPIcs*, pages 21:1–21:15, 2019. doi:10.4230/LIPICS.ICALP.2019.21.

[47] G. Bernardini, N. Pisanti, S. P. Pissis, and G. Rosone. Approximate pattern matching on elastic-degenerate text. *Theor. Comput. Sci.*, 812:109–122, 2020. doi:10.1016/J.TCS.2019.08.012.

[48] G. Bernardini, E. Gabory, S. P. Pissis, L. Stougie, M. Sweering, and W. Zuba. Elastic-degenerate string matching with 1 error. In *15th Latin American Symposium on Theoretical Informatics (LATIN)*, volume 13568 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2022. doi:10.1007/978-3-031-20624-5_2.

[49] G. Bernardini, P. Gawrychowski, N. Pisanti, S. P. Pissis, and G. Rosone. Elastic-degenerate string matching via fast matrix multiplication. *SIAM J. Comput.*, 51(3): 549–576, 2022. doi:10.1137/20M1368033.

[50] P. Bille. New algorithms for regular expression matching. In *33rd International Colloquium on Automata, Languages and Programming (ICALP)*, volume 4051 of *Lecture Notes in Computer Science*, pages 643–654. Springer, 2006. doi:10.1007/11786986_56.

[51] P. Bille and M. Farach-Colton. Fast and compact regular expression matching. *Theor. Comput. Sci.*, 409(3):486–496, 2008. doi:10.1016/J.TCS.2008.08.042.

[52] P. Bille and I. L. Gørtz. Sparse regular expression matching. In *2024 ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 3354–3375. SIAM, 2024. doi:10.1137/1.9781611977912.120.

[53] P. Bille and M. Thorup. Faster regular expression matching. In *36th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5555 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2009. doi:10.1007/978-3-642-02927-1_16.

[54] P. Bille and M. Thorup. Regular expression matching with multi-strings and intervals. In *31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1297–1308. SIAM, 2010. doi:10.1137/1.9781611973075.104.

[55] P. Bille, I. L. Gørtz, and T. Stordalen. Rank and select on degenerate strings. In A. Bilgin, J. E. Fowler, J. Serra-Sagristà, Y. Ye, and J. A. Storer, editors, *Data Compression Conference, DCC 2024, Snowbird, UT, USA, March 19-22, 2024*, pages 283–292. IEEE, 2024. doi:10.1109/DCC58796.2024.00036.

[56] P. Bottoni, A. Labella, and G. Rozenberg. Reaction systems with influence on environment. *J. Membr. Comput.*, 1(1):3–19, 2019. doi:10.1007/S41965-018-00005-8.

[57] M. F. Brameier and W. Banzhaf. *Basic concepts of linear genetic programming*. Springer, 2007.

[58] R. Brijder, A. Ehrenfeucht, and G. Rozenberg. A note on causalities in reaction systems. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 30, 2010. doi:10.14279/TUJ.ECEASST.30.429.

[59] R. Brijder, A. Ehrenfeucht, M. G. Main, and G. Rozenberg. A tour of reaction systems. *Int. J. Found. Comput. Sci.*, 22(7):1499–1517, 2011. doi:10.1142/S0129054111008842.

[60] R. Brijder, A. Ehrenfeucht, and G. Rozenberg. Reaction systems with duration. In J. Kelemen and A. Kelemenová, editors, *Computation, Cooperation, and Life - Essays Dedicated to Gheorghe Paun on the Occasion of His 60th Birthday*, volume 6610 of *Lecture Notes in Computer Science*, pages 191–202. Springer, 2011. doi:10.1007/978-3-642-20000-7_16.

[61] K. Bringmann. Fine-Grained Complexity Theory. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2019. doi:10.4230/LIPIcs.STACS.2019.4.

[62] K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In V. Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.15.

[63] K. Bringmann, A. Grønlund, and K. G. Larsen. A dichotomy for regular expression membership testing. In *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 307–318. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.36.

[64] L. Brodo, R. Bruni, M. Falaschi, R. Gori, F. Levi, and P. Milazzo. Quantitative extensions of reaction systems based on SOS semantics. *Neural Comput. Appl.*, 35(9):6335–6359, 2023. doi:10.1007/S00521-022-07935-6.

[65] L. Brodo, R. Bruni, and M. Falaschi. A framework for monitored dynamic slicing of reaction systems. *Nat. Comput.*, 23(2):217–234, 2024. doi:10.1007/S11047-024-09976-3.

[66] R. Bruni, R. Gori, P. Milazzo, and H. Siboulet. Melding boolean networks and reaction systems under synchronous, asynchronous and most permissive semantics. *Nat. Comput.*, 23(2):235–267, 2024. doi:10.1007/S11047-024-09990-5.

[67] T. Büchler, J. Olbrich, and E. Ohlebusch. Efficient short read mapping to a pangenome that is represented by a graph of ED strings. *Bioinform.*, 39(5), 2023. doi:10.1093/BIOINFORMATICS/BTAD320.

[68] C. Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2020. ISBN 9781108606806. doi:10.1017/9781108606806.

[69] C. Carlet. A wide class of boolean functions generalizing the hidden weight bit function. *IEEE Trans. Inf. Theory*, 68(2):1355–1368, 2022. doi:10.1109/TIT.2021.3126684.

[70] C. Carlet, D. Jakobovic, and S. Picek. Evolutionary algorithms-assisted construction of cryptographic boolean functions. In *GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021*, pages 565–573. ACM, 2021. doi:10.1145/3449639.3459362.

[71] C. Carlet, M. Djurasevic, D. Jakobovic, L. Mariot, and S. Picek. Evolving constructions for balanced, highly nonlinear boolean functions. In *GECCO '22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022*, pages 1147–1155. ACM, 2022. doi:10.1145/3512290.3528871.

[72] H. Chen, D. Doty, and D. Soloveichik. Deterministic function computation with chemical reaction networks. *Nat. Comput.*, 13(4):517–534, 2014. doi:10.1007/S11047-013-9393-6.

[73] H. Chen, H. Huang, R. Li, C. Peng, and W. Su. Incremental algorithms for solving regular expression intersection non-emptiness. *Theor. Comput. Sci.*, 1043:115249, 2025. doi:10.1016/J.TCS.2025.115249.

[74] T. Chen, A. Flores-Lamas, M. Hague, Z. Han, D. Hu, S. Kan, A. W. Lin, P. Rümmer, and Z. Wu. Solving string constraints with regex-dependent functions through transducers with priorities and variables. *Proc. ACM Program. Lang.*, 6(POPL):1–31, 2022. doi:10.1145/3498707.

[75] A. Cicherski, A. Lisiecka, and N. Dojer. Alfapang: Alignment free algorithm for pangenome graph construction. In S. P. Pissis and W. Sung, editors, *24th International Workshop on Algorithms in Bioinformatics, WABI 2024, September 2-4, 2024, Royal Holloway, London, United Kingdom*, volume 312 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.WABI.2024.23.

[76] A. Cislak, S. Grabowski, and J. Holub. Sopang: online text searching over a pangenome. *Bioinform.*, 34(24):4290–4292, 2018. doi:10.1093/BIOINFORMATICS/BTY506.

[77] S. Clamons, L. Qian, and E. Winfree. Programming and simulating chemical reaction networks on a surface. *Journal of the Royal Society Interface*, 17(166):20190790, 2020. doi:10.1098/rsif.2019.0790.

[78] J. A. Clark and J. Jacob. Two-stage optimisation in the design of boolean functions. In *Information Security and Privacy, 5th Australasian Conference, ACISP 2000, Brisbane, Australia, July 10-12, 2000, Proceedings*, volume 1841 of *Lecture Notes in Computer Science*, pages 242–254. Springer, 2000. doi:10.1007/10718964_20.

[79] P. Clifford and R. Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007. doi:10.1016/J.IPL.2006.08.002.

[80] R. Cole and R. Hariharan. Tree pattern matching and subset matching in randomized o(n $\log^3$m) time. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing (STOC)*, pages 66–75. ACM, 1997. doi:10.1145/258533.258553.

[81] R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 592–601. ACM, 2002. doi:10.1145/509907.509992.

[82] R. Cole and R. Hariharan. Tree pattern matching to subset matching in linear time. *SIAM J. Comput.*, 32(4):1056–1066, 2003. doi:10.1137/S0097539700382704.

[83] C. P. Consortium. Computational pan-genomics: status, promises and challenges. *Briefings Bioinform.*, 19(1):118–135, 2018. doi:10.1093/BIB/BBW089.

[84] L. Corolli, C. Maj, F. Marini, D. Besozzi, and G. Mauri. An excursion in reaction systems: From computer science to biology. *Theor. Comput. Sci.*, 454:95–108, 2012. doi:10.1016/j.tcs.2012.04.003.

[85] D. A. Cox, J. Little, and D. O'Shea. *Gröbner Bases*, pages 49–119. Springer International Publishing, Cham, 2015. ISBN 978-3-319-16721-3. doi:10.1007/978-3-319-16721-3_2.

[86] Y. Crama and P. L. Hammer. *Boolean Functions - Theory, Algorithms, and Applications*, volume 142 of *Encyclopedia of mathematics and its applications*. Cambridge University Press, 2011. ISBN 978-0-521-84751-3.

[87] M. Crochemore, C. Hancart, and T. Lecroq. *Algorithms on strings*. Cambridge University Press, 2007. ISBN 978-0-521-84899-2.

[88] M. Crochemore, C. S. Iliopoulos, T. Kociumaka, J. Radoszewski, W. Rytter, and T. Walen. Covering problems for partial words and for indeterminate strings. *Theor. Comput. Sci.*, 698:25–39, 2017. doi:10.1016/J.TCS.2017.05.026.

[89] E. Csuhaj-Varjú and G. Vaszil. P automata or purely communicating accepting P systems. In G. Paun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing, International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002, Revised Papers*, volume 2597 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2002. doi:10.1007/3-540-36490-0_14.

[90] E. Csuhaj-Varjú and G. Vaszil. Variants of distributed reaction systems. *Nat. Comput.*, 23(2):269–284, 2024. doi:10.1007/S11047-024-09974-5.

[91] E. Csuhaj-Varjú, O. H. Ibarra, and G. Vaszil. On the computational complexity of P automata. *Nat. Comput.*, 5(2):109–126, 2006. doi:10.1007/S11047-005-4461-1.

[92] E. Csuhaj-Varjú, O. Marion, and G. Vaszil. *The Oxford handbook of membrane computing*, chapter P Automata. Oxford University Press, 2009.

[93] P. Danecek, A. Auton, G. R. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, G. Lunter, G. T. Marth, S. T. Sherry, G. McVean, and R. Durbin. The variant call format and vcftools. *Bioinform.*, 27(15):2156–2158, 2011. doi:10.1093/BIOINFORMATICS/BTR330.

[94] J. D. Day, M. Kosche, F. Manea, and M. L. Schmid. Subsequences with gap constraints: Complexity bounds for matching and analysis problems. In S. W. Bae and H. Park, editors, *33rd International Symposium on Algorithms and Computation, ISAAC 2022, December 19-21, 2022, Seoul, Korea*, volume 248 of *LIPIcs*, pages 64:1–64:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.ISAAC.2022.64.

[95] J. W. Daykin and B. W. Watson. Indeterminate string factorizations and degenerate text transformations. *Math. Comput. Sci.*, 11(2):209–218, 2017. doi:10.1007/S11786-016-0285-X.

[96] J. W. Daykin, R. Groult, Y. Guesnet, T. Lecroq, A. Lefebvre, M. Léonard, L. Mouchard, É. Prieur, and B. W. Watson. Efficient pattern matching in degenerate strings with the burrows-wheeler transform. *Inf. Process. Lett.*, 147:82–87, 2019. doi:10.1016/J.IPL.2019.03.003.

[97] J. W. Daykin, R. Groult, Y. Guesnet, T. Lecroq, A. Lefebvre, M. Léonard, L. Mouchard, É. Prieur, and B. W. Watson. Efficient pattern matching in degenerate strings with the burrows-wheeler transform. *Inf. Process. Lett.*, 147:82–87, 2019. doi:10.1016/J.IPL.2019.03.003.

[98] M. de Oliveira Oliveira and M. Wehar. On the fine grained complexity of finite automata non-emptiness of intersection. In N. Jonoska and D. Savchuk, editors, *Developments in Language Theory - 24th International Conference, DLT 2020, Tampa, FL, USA, May 11-15, 2020, Proceedings*, volume 12086 of *Lecture Notes in Computer Science*, pages 69–82. Springer, 2020. doi:10.1007/978-3-030-48516-0_6.

[99] A. Dennunzio, E. Formenti, L. Manzoni, and A. E. Porreca. Ancestors, descendants, and gardens of eden in reaction systems. *Theor. Comput. Sci.*, 608:16–26, 2015. doi:10.1016/J.TCS.2015.05.046.

[100] A. Dennunzio, E. Formenti, L. Manzoni, and A. E. Porreca. Reachability in resource-bounded reaction systems. In *Language and Automata Theory and Applications - 10th International Conference, LATA 2016*, volume 9618 of *Lecture Notes in Computer Science*, pages 592–602. Springer, 2016. doi:10.1007/978-3-319-30000-9_45.

[101] A. Dennunzio, E. Formenti, L. Manzoni, and A. E. Porreca. Complexity of the dynamics of reaction systems. *Inf. Comput.*, 267:96–109, 2019. doi:10.1016/j.ic.2019.03.006.

[102] M. Djurasevic, D. Jakobovic, L. Mariot, and S. Picek. A survey of metaheuristic algorithms for the design of cryptographic boolean functions. *Cryptogr. Commun.*, 15(6): 1171–1197, 2023. doi:10.1007/S12095-023-00662-2.

[103] D. Doerr, J. Stoye, S. Böcker, and K. Jahn. Identifying gene clusters by discovering common intervals in indeterminate strings. *BMC Genomics*, 15(Suppl 6):S2, 2014. doi:10.1186/1471-2164-15-S6-S2.

[104] D. Dorey-Robinson, G. Maccari, and J. A. Hammond. Igmat: immunoglobulin sequence multi-species annotation tool for any species including those with incomplete antibody annotation or unusual characteristics. *BMC Bioinform.*, 24(1):491, 2023. doi:10.1186/S12859-023-05624-2.

[105] B. Dudek, P. Gawrychowski, G. Gourdel, and T. Starikovskaya. Streaming regular expression membership and pattern matching. In *2022 ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 670–694. SIAM, 2022. doi:10.1137/1.9781611977073.30.

[106] S. Dutta, A. Jankowski, G. Rozenberg, and A. Skowron. Linking reaction systems with rough sets. *Fundam. Informaticae*, 165(3-4):283–302, 2019. doi:10.3233/FI-2019-1786.

[107] A. Ehrenfeucht and G. Rozenberg. Basic notions of reaction systems. In *Developments in Language Theory, 8th International Conference (DLT)*, volume 3340 of *Lecture Notes in Computer Science*, pages 27–29. Springer, 2004. doi:10.1007/978-3-540-30550-7_3.

[108] A. Ehrenfeucht and G. Rozenberg. Reaction systems. *Fundam. Informaticae*, 75(1-4): 263–280, 2007.

[109] A. Ehrenfeucht and G. Rozenberg. Introducing time in reaction systems. *Theor. Comput. Sci.*, 410(4-5):310–322, 2009. doi:10.1016/J.TCS.2008.09.043.

[110] A. Ehrenfeucht, M. G. Main, and G. Rozenberg. Functions defined by reaction systems. *Int. J. Found. Comput. Sci.*, 22(1):167–178, 2011. doi:10.1142/S0129054111007927.

[111] A. Ehrenfeucht, J. Kleijn, M. Koutny, and G. Rozenberg. Evolving reaction systems. *Theor. Comput. Sci.*, 682:79–99, 2017. doi:10.1016/J.TCS.2016.12.031.

[112] J. M. Eizenga, A. M. Novak, E. Kobayashi, F. Villani, C. Cisar, S. Heumos, G. Hickey, V. Colonna, B. Paten, and E. Garrison. Efficient dynamic variation graphs. *Bioinform.*, 36(21):5139–5144, 2021. doi:10.1093/bioinformatics/btaa640.

[113] M. Equi, V. Mäkinen, and A. I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. In *47th International Conference on Current Trends in Theory and Practice of Computer Science, (SOFSEM)*, volume 12607 of *Lecture Notes in Computer Science*, pages 608–622. Springer, 2021. doi:10.1007/978-3-030-67731-2_44.

[114] M. Equi, T. Norri, J. Alanko, B. Cazaux, A. I. Tomescu, and V. Mäkinen. Algorithms and complexity on indexing elastic founder graphs. In *32nd International Symposium on Algorithms and Computation (ISAAC)*, volume 212 of *LIPIcs*, pages 20:1–20:18, 2021. doi:10.4230/LIPICS.ISAAC.2021.20.

[115] M. Equi, V. Mäkinen, and A. I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails. *Theor. Comput. Sci.*, 975:114128, 2023. doi:10.1016/J.TCS.2023.114128.

[116] M. Equi, V. Mäkinen, A. I. Tomescu, and R. Grossi. On the complexity of string matching for graphs. *ACM Trans. Algorithms*, 19(3):21:1–21:25, 2023. doi:10.1145/3588334.

[117] M. Equi, T. Norri, J. Alanko, B. Cazaux, A. I. Tomescu, and V. Mäkinen. Algorithms and complexity on indexing founder graphs. *Algorithmica*, 85(6):1586–1623, 2023. doi:10.1007/S00453-022-01007-W.

[118] L. et al. A draft human pangenome reference. *Nature*, 617(7960):312 – 324, 2023. doi:10.1038/s41586-023-05896-x.

[119] M. Farach. Optimal suffix tree construction with large alphabets. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*, pages 137–143, 1997. doi:10.1109/SFCS.1997.646102. URL `https://doi.org/10.1109/SFCS.1997.646102`.

[120] B. A. Farbey. Structural models: An introduction to the theory of directed graphs. *Journal of the Operational Research Society*, 17(2):202–203, 1966. doi:10.1057/jors.1966.32.

[121] M. Federico and N. Pisanti. Suffix tree characterization of maximal motifs in biological sequences. *Theor. Comput. Sci.*, 410(43):4391–4401, 2009. doi:10.1016/J.TCS.2009.07.020.

[122] H. Fernau and A. Krebs. Problems on finite automata and the exponential time hypothesis. *Algorithms*, 10(1):24, 2017. doi:10.3390/A10010024.

[123] E. Formenti, L. Manzoni, and A. E. Porreca. Fixed points and attractors of reaction systems. In *Language, Life, Limits - 10th Conference on Computability in Europe, CiE 2014*, volume 8493 of *Lecture Notes in Computer Science*, pages 194–203. Springer, 2014. doi:10.1007/978-3-319-08019-2_20.

[124] E. Formenti, L. Manzoni, and A. E. Porreca. Cycles and global attractors of reaction systems. In *Descriptional Complexity of Formal Systems - 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*, volume 8614 of *Lecture Notes in Computer Science*, pages 114–125. Springer, 2014. doi:10.1007/978-3-319-09704-6_11.

[125] E. Formenti, L. Manzoni, and A. E. Porreca. On the complexity of occurrence and convergence problems in reaction systems. *Nat. Comput.*, 14(1):185–191, 2015. doi:10.1007/s11047-014-9456-3.

[126] R. Freund and S. Verlan. A formal framework for static (tissue) P systems. In G. Eleftherakis, P. Kefalas, G. Paun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, 8th International Workshop, WMC 2007, Thessaloniki, Greece, June 25-28, 2007 Revised Selected and Invited Papers*, volume 4860 of *Lecture Notes in Computer Science*, pages 271–284. Springer, 2007. doi:10.1007/978-3-540-77312-2_17.

[127] R. Freund and S. Verlan. (tissue) P systems working in the k-restricted minimally or maximally parallel transition mode. *Nat. Comput.*, 10(2):821–833, 2011. doi:10.1007/S11047-010-9215-Z.

[128] R. Freund, C. Martín-Vide, A. Obtulowicz, and G. Paun. On three classes of automata-like P systems. In Z. Ésik and Z. Fülöp, editors, *Developments in Language Theory, 7th International Conference, DLT 2003, Szeged, Hungary, July 7-11, 2003, Proceedings*, volume 2710 of *Lecture Notes in Computer Science*, pages 292–303. Springer, 2003. doi:10.1007/3-540-45007-6_23.

[129] E. Gabory, N. M. Mwaniki, N. Pisanti, S. P. Pissis, J. Radoszewski, M. Sweering, and W. Zuba. Comparing elastic-degenerate strings: Algorithms, lower bounds, and applications. In L. Bulteau and Z. Lipták, editors, *34th Annual Symposium on Combinatorial Pattern Matching, CPM 2023, June 26-28, 2023, Marne-la-Vallée, France*, volume 259 of *LIPIcs*, pages 11:1–11:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.CPM.2023.11.

[130] E. Gabory, N. M. Mwaniki, N. Pisanti, S. P. Pissis, J. Radoszewski, M. Sweering, and W. Zuba. Pangenome comparison via ED strings. *Frontiers Bioinform.*, 4, 2024. doi:10.3389/FBINF.2024.1397036.

[131] E. Gabory, N. M. Mwaniki, N. Pisanti, S. P. Pissis, J. Radoszewski, M. Sweering, and W. Zuba. Elastic-degenerate string comparison. *Inf. Comput.*, 304:105296, 2025. doi:10.1016/J.IC.2025.105296.

[132] M. Gadouleau. Dynamical properties of disjunctive boolean networks (invited talk). In A. Castillo-Ramirez, P. Guillon, and K. Perrot, editors, *27th IFIP WG 1.5 International Workshop on Cellular Automata and Discrete Complex Systems, AUTOMATA 2021, July 12-14, 2021, Aix-Marseille University, France*, volume 90 of *OASIcs*, pages 1:1–1:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/OASICS.AUTOMATA.2021.1.

[133] G. Gamard, P. Guillon, K. Perrot, and G. Theyssier. Rice-like theorems for automata networks. In M. Bläser and B. Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 32:1–32:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.STACS.2021.32.

[134] E. Garrison, J. Sirén, A. M.Novak, G. Hickey, J. M.Eizenga, E. T.Dawson, W. Jones, S. Garg, C. Markello, M. F. MF, B. Paten, and R. Durbin. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature Biotechnology*, 36(9):875–879, 2018. doi:10.1038/nbt.4227.

[135] P. Gawrychowski, S. Ghazawi, and G. M. Landau. On indeterminate strings matching. In *31st Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 161 of *LIPIcs*, pages 14:1–14:14, 2020. doi:10.4230/LIPICS.CPM.2020.14.

[136] D. Genova, H. J. Hoogeboom, and Z. G. Prodanoff. Extracting reaction systems from function behavior. *J. Membr. Comput.*, 2(3):194–206, 2020. doi:10.1007/S41965-020-00045-Z.

[137] D. Gibney. An efficient elastic-degenerate text index? not likely. In *27th International Symposium on String Processing and Information Retrieval*, volume 12303 of *Lecture Notes in Computer Science*, pages 76–88. Springer, 2020. doi:10.1007/978-3-030-59212-7_6.

[138] D. Gibney, G. Hoppenworth, and S. V. Thankachan. Simple reductions from formula-sat to pattern matching on labeled graphs and subtree isomorphism. In *4th SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 232–242, 2021. doi:10.1137/1.9781611976496.26.

[139] A. Granas and J. Dugundji. *Elementary Fixed Point Theorems*, pages 9–84. Springer New York, New York, NY, 2003. doi:10.1007/978-0-387-21593-8_2.

[140] R. Groot Koerkamp. A*PA2: Up to 19× faster exact global alignment. In *24th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 312 of *LIPIcs*, pages 17:1–17:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. doi:10.4230/LIPICS.WABI.2024.17.

[141] R. Grossi, A. Pietracaprina, N. Pisanti, G. Pucci, E. Upfal, and F. Vandin. MADMX: A novel strategy for maximal dense motif extraction. In S. Salzberg and T. J. Warnow, editors, *Algorithms in Bioinformatics, 9th International Workshop, WABI 2009, Philadelphia, PA, USA, September 12-13, 2009. Proceedings*, volume 5724 of *Lecture Notes in Computer Science*, pages 362–374. Springer, 2009. doi:10.1007/978-3-642-04241-6_30.

[142] R. Grossi, A. Pietracaprina, N. Pisanti, G. Pucci, E. Upfal, and F. Vandin. MADMX: A strategy for maximal dense motif extraction. *J. Comput. Biol.*, 18(4):535–545, 2011. doi:10.1089/CMB.2010.0177.

[143] R. Grossi, C. S. Iliopoulos, C. Liu, N. Pisanti, S. P. Pissis, A. Retha, G. Rosone, F. Vayani, and L. Versari. On-line pattern matching on similar texts. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 78 of *LIPIcs*, pages 9:1–9:14, 2017. doi:10.4230/LIPICS.CPM.2017.9.

[144] M. Gu, M. Farach, and R. Beigel. An efficient algorithm for dynamic text indexing. In D. D. Sleator, editor, *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. 23-25 January 1994, Arlington, Virginia, USA*, pages 697–704. ACM/SIAM, 1994.

[145] R. Guanciale, D. Gurov, and P. Laud. Private intersection of regular languages. In *12th Annual International Conference on Privacy, Security and Trust*, pages 112–120. IEEE Computer Society, 2014. doi:10.1109/PST.2014.6890930.

[146] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In *1st International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110. Springer, 1995. doi:10.1007/3-540-60630-0_5.

[147] J. Holub, W. F. Smyth, and S. Wang. Fast pattern-matching on indeterminate strings. *J. Discrete Algorithms*, 6(1):37–50, 2008. doi:10.1016/J.JDA.2006.10.003.

[148] M. Holzer and C. Rauch. On the computational complexity of reaction systems, revisited. In R. Santhanam and D. Musatov, editors, *Computer Science - Theory and Applications - 16th International Computer Science Symposium in Russia, CSR 2021, Sochi, Russia, June 28 - July 2, 2021, Proceedings*, volume 12730 of *Lecture Notes in Computer Science*, pages 170–185. Springer, 2021. doi:10.1007/978-3-030-79416-3_10.

[149] R. Hrbacek and V. Dvorak. Bent function synthesis by means of cartesian genetic programming. In *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings*, volume 8672 of *Lecture Notes in Computer Science*, pages 414–423. Springer, 2014. doi:10.1007/978-3-319-10762-2_41.

[150] J. Husa. Comparison of genetic programming methods on design of cryptographic boolean functions. In *Genetic Programming - 22nd European Conference, EuroGP 2019, Held as Part of EvoStar 2019, Leipzig, Germany, April 24-26, 2019, Proceedings*, volume 11451 of *Lecture Notes in Computer Science*, pages 228–244. Springer, 2019. doi:10.1007/978-3-030-16670-0_15.

[151] J. Husa and R. Dobai. Designing bent boolean functions with parallelized linear genetic programming. In *Genetic and Evolutionary Computation Conference, Berlin, Germany, July 15-19, 2017, Companion Material Proceedings*, pages 1825–1832. ACM, 2017. doi:10.1145/3067695.3084220.

[152] J. Husa and L. Sekanina. Semantic mutation operator for a fast and efficient design of bent boolean functions. *Genet. Program. Evolvable Mach.*, 25(1):3, 2024. doi:10.1007/S10710-023-09476-W.

[153] C. S. Iliopoulos and J. Radoszewski. Truly subquadratic-time extension queries and periodicity detection in strings with uncertainties. In *27th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 54 of *LIPIcs*, pages 8:1–8:12, 2016. doi:10.4230/LIPICS.CPM.2016.8.

[154] C. S. Iliopoulos, L. Mouchard, and M. S. Rahman. A new approach to pattern matching in degenerate DNA/RNA sequences and distributed pattern matching. *Math. Comput. Sci.*, 1(4):557–569, 2008. doi:10.1007/S11786-007-0029-Z.

[155] C. S. Iliopoulos, R. Kundu, and S. P. Pissis. Efficient pattern matching in elastic-degenerate strings. *Information and Computation*, 279:104616, 2021. doi:10.1016/j.ic.2020.104616.

[156] N. Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999. ISBN 978-1-4612-6809-3. doi:10.1007/978-1-4612-0539-5.

[157] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2): 367–375, 2001. doi:10.1006/JCSS.2000.1727.

[158] H. Intekhab and W. C. Teh. Ranks of functions specified by minimal reaction systems and induced by images of singletons. *Nat. Comput.*, 23(2):285–293, 2024. doi:10.1007/S11047-024-09973-6.

[159] IUPAC-IUB Commission on Biochemical Nomenclature. Abbreviations and symbols for nucleic acids, polynucleotides, and their constituents. *Biochemistry*, 9(20):4022–4027, 1970. doi:10.1016/0022-2836(71)90319-6.

[160] S. Ivanov and I. Petre. Controllability of reaction systems. *J. Membr. Comput.*, 2(4): 290–302, 2020. doi:10.1007/s41965-020-00055-x.

[161] A. John and J. Jose. *Cellular Automata and Cryptography*, pages 285–306. Springer Nature Switzerland, Cham, 2025. ISBN 978-3-031-81097-8. doi:10.1007/978-3-031-81097-8_11.

[162] H. R. Jr. *Theory of recursive functions and effective computability (Reprint from 1967)*. MIT Press, 1987. ISBN 978-0-262-68052-3.

[163] M. Kaniecki and L. Mikulski. On categorical approach to reaction systems. *Nat. Comput.*, 23(2):295–307, 2024. doi:10.1007/S11047-024-09978-1.

[164] J. Kari. Rice's theorem for the limit sets of cellular automata. *Theor. Comput. Sci.*, 127 (2):229–254, 1994. ISSN 0304-3975. doi:https://doi.org/10.1016/0304-3975(94)90041-8.

[165] S. Kavut, S. Maitra, and M. D. Yücel. Search for boolean functions with excellent profiles in the rotation symmetric class. *IEEE Trans. Inf. Theory*, 53(5):1743–1751, 2007. doi:10.1109/TIT.2007.894696.

[166] J. Kleijn, M. Koutny, and G. Rozenberg. Modelling reaction systems with petri nets. In *BioPPN-2011, 2nd International Workshop on Biological Processes & Petri Nets*. Newcastle University, 2011.

[167] D. E. Knuth, J. H. M. Jr., and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977. doi:10.1137/0206024.

[168] D. Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 254–266. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.16.

[169] O. Kramer. Genetic algorithms. In *Genetic algorithm essentials*, pages 11–19. Springer, 2017.

[170] H. Kreowski and A. Lye. Modeling np-problems with families of extended graph-based reaction systems. *Nat. Comput.*, 23(2):309–322, 2024. doi:10.1007/S11047-024-09984-3.

[171] W. B. Langdon and R. Poli. *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[172] T. Lemane, R. Chikhi, and P. Peterlongo. KMDIFF: large-scale and user-friendly differential *k*-mer analyses. *Bioinform.*, 38(24):5443–5445, 2022. doi:10.1093/BIOINFORMATICS/BTAC689.

[173] A. Leporati, L. Manzoni, G. Mauri, A. E. Porreca, and C. Zandron. Simulating elementary active membranes - with an application to the P conjecture. In M. Gheorghe, G. Rozenberg, A. Salomaa, P. Sosík, and C. Zandron, editors, *Membrane Computing - 15th International Conference, CMC 2014, Prague, Czech Republic, August 20-22, 2014, Revised Selected Papers*, volume 8961 of *Lecture Notes in Computer Science*, pages 284–299. Springer, 2014. doi:10.1007/978-3-319-14370-5_18.

[174] A. Leporati, L. Manzoni, G. Mauri, A. E. Porreca, and C. Zandron. Membrane division, oracles, and the counting hierarchy. *Fundam. Informaticae*, 138(1-2):97–111, 2015. doi:10.3233/FI-2015-1201.

[175] Y. Li, Z. Chen, J. Cao, Z. Xu, Q. Peng, H. Chen, L. Chen, and S. Cheung. Redoshunter: A combined static and dynamic approach for regular expression dos detection. In *30th USENIX Security Symposium*, pages 3847–3864. USENIX Association, 2021.

[176] T. Liang, N. Tsiskaridze, A. Reynolds, C. Tinelli, and C. W. Barrett. A decision procedure for regular membership and length constraints over unbounded strings. In *10th International Symposium on Frontiers of Combining Systems (FroCoS)*, volume 9322 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 2015. doi:10.1007/978-3-319-24246-0_9.

[177] Y. E. Lien. Periodic properties of strings. *SIGACT News*, 7(1):21–25, 1975. doi:10.1145/990518.990520.

[178] F. A. Louza, N. Mhaskar, and W. F. Smyth. A new approach to regular & indeterminate strings. *Theor. Comput. Sci.*, 854:105–115, 2021. doi:10.1016/J.TCS.2020.12.007.

[179] V. Mäkinen, B. Cazaux, M. Equi, T. Norri, and A. I. Tomescu. Linear time construction of indexable founder block graphs. In *20th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 172 of *LIPIcs*, pages 7:1–7:18, 2020. doi:10.4230/LIPICS.WABI.2020.7.

[180] L. Manzoni, M. Castelli, and L. Vanneschi. Evolutionary reaction systems. In *Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics - 10th European Conference, EvoBIO 2012*, volume 7246 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 2012. doi:10.1007/978-3-642-29066-4_2.

[181] L. Manzoni, D. Poças, and A. E. Porreca. Simple reaction systems and their classification. *Int. J. Found. Comput. Sci.*, 25(4):441–458, 2014. doi:10.1142/S012905411440005X.

[182] L. Manzoni, L. Mariot, and E. Tuba. Balanced crossover operators in genetic algorithms. *Swarm Evol. Comput.*, 54:100646, 2020. doi:10.1016/J.SWEVO.2020.100646.

[183] L. Manzoni, A. E. Porreca, and G. Rozenberg. Facilitation in reaction systems. *J. Membr. Comput.*, 2(3):149–161, 2020. doi:10.1007/S41965-020-00044-0.

[184] L. Manzoni, L. Mariot, and G. Menara. Combinatorial designs and cellular automata: A survey. *arXiv preprint arXiv:2503.10320*, 2025. doi:10.48550/arXiv.2503.10320.

[185] F. Marini and B. Walczak. Particle swarm optimization (pso). a tutorial. *Chemometrics and Intelligent Laboratory Systems*, 149:153–165, 2015. ISSN 0169-7439. doi:10.1016/j.chemolab.2015.08.020.

[186] L. Mariot. Insights gained after a decade of cellular automata-based cryptography. In M. Gadouleau and A. Castillo-Ramirez, editors, *Cellular Automata and Discrete Complex Systems*, pages 35–54, Cham, 2024. Springer Nature Switzerland. ISBN 978-3-031-65887-7.

[187] L. Mariot and A. Leporati. Heuristic search by particle swarm optimization of boolean functions for cryptographic applications. In *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*, pages 1425–1426. ACM, 2015. doi:10.1145/2739482.2764674.

[188] L. Mariot and A. Leporati. A genetic algorithm for evolving plateaued cryptographic boolean functions. In *Theory and Practice of Natural Computing - Fourth International Conference, TPNC 2015, Mieres, Spain, December 15-16, 2015. Proceedings*, volume 9477 of *Lecture Notes in Computer Science*, pages 33–45. Springer, 2015. doi:10.1007/978-3-319-26841-5_3.

[189] L. Mariot, S. Picek, D. Jakobovic, and A. Leporati. Evolutionary algorithms for the design of orthogonal latin squares based on cellular automata. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, page 306–313, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349208. doi:10.1145/3071178.3071284.

[190] L. Mariot, S. Picek, D. Jakobovic, M. Djurasevic, and A. Leporati. Evolutionary construction of perfectly balanced boolean functions. In *IEEE Congress on Evolutionary Computation, CEC 2022, Padua, Italy, July 18-23, 2022*, pages 1–8. IEEE, 2022. doi:10.1109/CEC55065.2022.9870427.

[191] L. Mariot, M. Saletta, A. Leporati, and L. Manzoni. Heuristic search of (semi-)bent functions based on cellular automata. *Nat. Comput.*, 21(3):377–391, 2022. doi:10.1007/S11047-022-09885-3.

[192] L. Mariot, M. Saletta, A. Leporati, and L. Manzoni. Heuristic search of (semi-)bent functions based on cellular automata. *Natural Computing*, 21(3):377–391, Sep 2022. ISSN 1572-9796. doi:10.1007/s11047-022-09885-3.

[193] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers (extended abstract). In *Advances in Cryptology - EUROCRYPT '88, Workshop on the Theory and Application of of Cryptographic Techniques, Davos, Switzerland, May 25-27, 1988, Proceedings*, volume 330 of *Lecture Notes in Computer Science*, pages 301–314. Springer, 1988. doi:10.1007/3-540-45961-8_28.

[194] A. Meski, M. Koutny, L. Mikulski, and W. Penczek. Reaction mining for reaction systems. *Nat. Comput.*, 23(2):323–343, 2024. doi:10.1007/S11047-024-09989-Y.

[195] W. Millan, A. J. Clark, and E. Dawson. An effective genetic algorithm for finding highly nonlinear boolean functions. In *Information and Communication Security, First International Conference, ICICS'97, Beijing, China, November 11-14, 1997, Proceedings*, volume 1334 of *Lecture Notes in Computer Science*, pages 149–158. Springer, 1997. doi:10.1007/BFB0028471.

[196] W. Millan, A. J. Clark, and E. Dawson. Heuristic design of cryptographically strong balanced boolean functions. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 489–499. Springer, 1998. doi:10.1007/BFB0054148.

[197] W. Millan, J. Fuller, and E. Dawson. New concepts in evolutionary search for boolean functions in cryptology. *Comput. Intell.*, 20(3):463–474, 2004. doi:10.1111/J.0824-7935.2004.00246.X.

[198] J. F. Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, page 1135–1142, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1558606114. doi:10.5555/2934046.2934074.

[199] J. F. Miller and A. J. Turner. Cartesian genetic programming. In *Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015, Companion Material Proceedings*, pages 179–198. ACM, 2015. doi:10.1145/2739482.2756571.

[200] N. M. Mwaniki and N. Pisanti. Optimal sequence alignment to ED-strings. In *18th International Symposium Bioinformatics Research and Applications (ISBRA)*, volume 13760 of *Lecture Notes in Computer Science*, pages 204–216. Springer, 2022. doi:10.1007/978-3-031-23198-8_19.

[201] N. M. Mwaniki, E. Garrison, and N. Pisanti. Fast exact string to D-texts alignments. In *16th International Joint Conference on Biomedical Engineering Systems and Technologies (BIOSTEC)*, pages 70–79. SCITEPRESS, 2023. doi:10.5220/0011666900003414.

[202] E. W. Myers. A four russians algorithm for regular expression pattern matching. *J. ACM*, 39(2):430–448, 1992. doi:10.1145/128749.128755.

[203] J. V. Neumann and A. W. Burks. *Theory of Self-Reproducing Automata*. University of Illinois Press, USA, 1966.

[204] M. S. Nobile, A. E. Porreca, S. Spolaor, L. Manzoni, P. Cazzaniga, G. Mauri, and D. Besozzi. Efficient simulation of reaction systems on graphics processing units. *Fundam. Informaticae*, 154(1-4):307–321, 2017. doi:10.3233/FI-2017-1568. URL https://doi.org/10.3233/FI-2017-1568.

[205] K. Nyberg. Perfect nonlinear s-boxes. In *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, volume 547 of *Lecture Notes in Computer Science*, pages 378–386. Springer, 1991. doi:10.1007/3-540-46416-6_32.

[206] R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. ISBN 978-1-10-703832-5.

[207] F. Okubo. Reaction automata working in sequential manner. *RAIRO Theor. Informatics Appl.*, 48(1):23–38, 2014. doi:10.1051/ITA/2013047.

[208] F. Okubo and T. Yokomori. *Recent Developments on Reaction Automata Theory: A Survey*, pages 1–22. Springer Japan, Tokyo, 2015. ISBN 978-4-431-55105-8. doi:10.1007/978-4-431-55105-8_1.

[209] F. Okubo and T. Yokomori. The computational capability of chemical reaction automata. *Nat. Comput.*, 15(2):215–224, 2016. doi:10.1007/S11047-015-9504-7.

[210] F. Okubo and T. Yokomori. The computing power of determinism and reversibility in chemical reaction automata. In A. Adamatzky, editor, *Reversibility and Universality, Essays Presented to Kenichi Morita on the Occasion of his 70th Birthday*, volume 30 of *Emergence, Complexity and Computation*, pages 279–298. Springer, 2018. doi:10.1007/978-3-319-73216-9_13.

[211] F. Okubo, S. Kobayashi, and T. Yokomori. Reaction automata. *Theor. Comput. Sci.*, 429: 247–257, 2012. doi:10.1016/J.TCS.2011.12.045.

[212] F. Okubo, S. Kobayashi, and T. Yokomori. On the properties of language classes defined by bounded reaction automata. *Theor. Comput. Sci.*, 454:206–221, 2012. doi:10.1016/J.TCS.2012.03.024.

[213] F. Okubo, K. Fujioka, and T. Yokomori. Chemical reaction regular grammars. *New Gener. Comput.*, 40(2):659–680, 2022. doi:10.1007/S00354-022-00160-8.

[214] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994. ISBN 978-0-201-53082-7.

[215] B. Paten, A. M. Novak, J. M. Eizenga, and E. Garrison. Genome graphs and the evolution of genome inference. *Genome Res*, 27(5):665–676, 2017. doi:10.1101/gr.214155.116.

[216] G. Paun. *Membrane Computing: An Introduction*. Natural computing series. Springer, 2002. ISBN 978-3-540-43601-0. doi:10.1007/978-3-642-56196-2.

[217] G. Paun and G. Rozenberg. A guide to membrane computing. *Theor. Comput. Sci.*, 287 (1):73–100, 2002. doi:10.1016/S0304-3975(02)00136-6.

[218] G. Paun, M. J. Pérez-Jiménez, and G. Rozenberg. Bridging membrane and reaction systems - further results and research topics. *Fundam. Informaticae*, 127(1-4):99–114, 2013. doi:10.3233/FI-2013-898.

[219] P. Peterlongo, N. Pisanti, F. Boyer, and M. Sagot. Lossless filter for finding long multiple approximate repetitions using a new data structure, the bi-factor array. In M. P. Consens and G. Navarro, editors, *String Processing and Information Retrieval, 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, November 2-4, 2005, Proceedings*, volume 3772 of *Lecture Notes in Computer Science*, pages 179–190. Springer, 2005. doi:10.1007/11575832_20.

[220] P. Peterlongo, N. Pisanti, F. Boyer, A. P. do Lago, and M. Sagot. Lossless filter for multiple repetitions with hamming distance. *J. Discrete Algorithms*, 6(3):497–509, 2008. doi:10.1016/J.JDA.2007.03.003.

[221] P. Peterlongo, G. A. T. Sacomoto, A. P. do Lago, N. Pisanti, and M. Sagot. Lossless filter for multiple repeats with bounded edit distance. *Algorithms Mol. Biol.*, 4, 2009. doi:10.1186/1748-7188-4-3.

[222] S. Picek and D. Jakobovic. Evolving algebraic constructions for designing bent boolean functions. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, pages 781–788. ACM, 2016. doi:10.1145/2908812.2908915.

[223] S. Picek, D. Jakobovic, and M. Golub. Evolving cryptographically sound boolean functions. In *Genetic and Evolutionary Computation Conference, GECCO '13*, pages 191–192. ACM, 2013. doi:10.1145/2464576.2464671.

[224] S. Picek, D. Jakobovic, J. F. Miller, E. Marchiori, and L. Batina. Evolutionary methods for the construction of cryptographic boolean functions. In *Genetic Programming - 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings*, volume 9025 of *Lecture Notes in Computer Science*, pages 192–204. Springer, 2015. doi:10.1007/978-3-319-16501-1_16.

[225] S. Picek, D. Sisejkovic, and D. Jakobovic. Immunological algorithms paradigm for construction of boolean functions with good cryptographic properties. *Eng. Appl. Artif. Intell.*, 62:320–330, 2017. doi:10.1016/J.ENGAPPAI.2016.11.002.

[226] S. Picek, K. Knezevic, L. Mariot, D. Jakobovic, and A. Leporati. Evolving bent quaternary functions. In *2018 IEEE Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 8-13, 2018*, pages 1–8. IEEE, 2018. doi:10.1109/CEC.2018.8477677.

[227] N. Pisanti, H. Soldano, and M. Carpentier. Incremental inference of relational motifs with a degenerate alphabet. In *16th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 3537 of *Lecture Notes in Computer Science*, pages 229–240. Springer, 2005. doi:10.1007/11496656_20.

[228] N. Pisanti, H. Soldano, M. Carpentier, and J. Pothier. A relational extension of the notion of motifs: Application to the common 3d protein substructures searching problem. *Journal of Compututational Biology*, 16(12):1635–1660, 2009. doi:10.1089/CMB.2008.0019.

[229] S. P. Pissis. Optimal prefix-suffix queries with applications. In *2025 Symposium on Simplicity in Algorithms (SOSA)*, pages 166–171. SIAM, 2025. doi:10.1137/1.9781611978315.13.

[230] S. P. Pissis and A. Retha. Dictionary matching in elastic-degenerate texts with applications in searching VCF files on-line. In *17th International Symposium on Experimental Algorithms (SEA)*, volume 103 of *LIPIcs*, pages 16:1–16:14, 2018. doi:10.4230/LIPICS.SEA.2018.16.

[231] P. Procházka, O. Cvacho, L. Krcál, and J. Holub. Backward pattern matching on elastic-degenerate strings. *SN Comput. Sci.*, 4(5):442, 2023. doi:10.1007/S42979-023-01760-X.

[232] G. Rakocevic, V. Semenyuk, W.-P. Lee, J. Spencer, J. Browning, I. J. Johnson, V. Arsenijevic, J. Nadj, K. Ghose, M. C. Suciu, S.-G. Ji, G. Demir, L. Li, B. Ç. Toptaş, A. Dolgoborodov, B. Pollex, I. Spulber, I. Glotova, P. Kómár, A. L. Stachyra, Y. Li, M. Popovic, M. Källberg, A. Jain, and D. Kural. Fast and accurate genomic analyses using genome graphs. *Nature Genetics*, 51:354–362, 2019. doi:10.1038/s41588-018-0316-4.

[233] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953. ISSN 00029947, 10886850.

[234] N. Rizzo and V. Mäkinen. Indexable elastic founder graphs of minimum height. In *33rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 223 of *LIPIcs*, pages 19:1–19:19, 2022. doi:10.4230/LIPICS.CPM.2022.19.

[235] N. Rizzo and V. Mäkinen. Linear time construction of indexable elastic founder graphs. In *33rd International Workshop on Combinatorial Algorithms (IWOCA)*, volume 13270 of *Lecture Notes in Computer Science*, pages 480–493. Springer, 2022. doi:10.1007/978-3-031-06678-8_35.

[236] N. Rizzo, M. Equi, T. Norri, and V. Mäkinen. Elastic founder graphs improved and enhanced. *Theor. Comput. Sci.*, 982:114269, 2024. doi:10.1016/J.TCS.2023.114269.

[237] L. Rovito, A. D. Lorenzo, and L. Manzoni. Evolution of walsh transforms with genetic programming. In *Companion Proceedings of the Conference on Genetic and Evolutionary Computation, GECCO 2023, Companion Volume, Lisbon, Portugal, July 15-19, 2023*, pages 2386–2389. ACM, 2023. doi:10.1145/3583133.3596317.

[238] M. Sagot, A. Viari, and H. Soldano. Multiple sequence comparison: A peptide matching approach. In *6th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 937 of *Lecture Notes in Computer Science*, pages 366–385. Springer, 1995. doi:10.1007/3-540-60044-2_55.

[239] M. Sagot, A. Viari, and H. Soldano. Multiple sequence comparison - A peptide matching approach. *Theor. Comput. Sci.*, 180(1-2):115–137, 1997. doi:10.1016/S0304-3975(96)00137-5.

[240] S. Saha, R. S. Chakraborty, S. S. Nuthakki, Anshul, and D. Mukhopadhyay. Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 577–596. Springer, 2015. doi:10.1007/978-3-662-48324-4_29.

[241] A. Salomaa. Minimal reaction systems: Duration and blips. *Theor. Comput. Sci.*, 682:208–216, 2017. doi:10.1016/J.TCS.2017.01.032.

[242] P. Schepper. Fine-grained complexity of regular expression pattern matching and membership. In F. Grandoni, G. Herman, and P. Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 80:1–80:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.80.

[243] A. Shiftan and E. Porat. Set intersection and sequence matching with mismatch counting. *Theor. Comput. Sci.*, 638:3–10, 2016. doi:10.1016/J.TCS.2016.01.003.

[244] J. Sirén, E. Garrison, A. M. Novak, B. Paten, and R. Durbin. Haplotype-aware graph indexes. In *18th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 113 of *LIPIcs*, pages 4:1–4:13, 2018. doi:10.4230/LIPICS.WABI.2018.4.

[245] H. Soldano, A. Viari, and M. Champesme. Searching for flexible repeated patterns using a non-transitive similarity relation. *Pattern Recognit. Lett.*, 16(3):233–246, 1995. doi:10.1016/0167-8655(94)00095-K.

[246] P. Sosík. The computational power of cell division in P systems: Beating down parallel computers? *Nat. Comput.*, 2(3):287–298, 2003. doi:10.1023/A:1025401325428.

[247] P. Sosík. P systems attacking hard problems beyond NP: a survey. *J. Membr. Comput.*, 1(3):198–208, 2019. doi:10.1007/S41965-019-00017-Y.

[248] C. Stanford, M. Veanes, and N. S. Bjørner. Symbolic boolean derivatives for efficiently solving extended regular expression constraints. In *42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*, pages 620–635. ACM, 2021. doi:10.1145/3453483.3454066.

[249] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976. doi:10.1016/0304-3975(76)90061-X.

[250] W. Su, R. Li, C. Peng, and H. Chen. Algorithms for checking intersection non-emptiness of regular expressions. In *20th International Colloquium on Theoretical Aspects of Computing (ICTAC)*, volume 14446 of *Lecture Notes in Computer Science*, pages 216–235. Springer, 2023. doi:10.1007/978-3-031-47963-2_14.

[251] W. C. Teh and A. Atanasiu. Irreducible reaction systems and reaction system rank. *Theor. Comput. Sci.*, 666:12–20, 2017. doi:10.1016/J.TCS.2016.08.021.

[252] W. C. Teh and A. Atanasiu. Simulation of reaction systems by the strictly minimal ones. *J. Membr. Comput.*, 2(3):162–170, 2020. doi:10.1007/S41965-020-00042-2.

[253] W. C. Teh and J. Lim. Evolvability of reaction systems and the invisibility theorem. *Theor. Comput. Sci.*, 924:17–33, 2022. doi:10.1016/J.TCS.2022.03.039.

[254] C. Thachuk. Indexing hypertext. *J. Discrete Algorithms*, 18:113–122, 2013. doi:10.1016/J.JDA.2012.10.001.

[255] C. Umans. The minimum equivalent DNF problem and shortest implicants. *J. Comput. Syst. Sci.*, 63(4):597–611, 2001. doi:10.1006/JCSS.2001.1775.

[256] X. Wang, C. Zhang, Y. Li, Z. Xu, S. Huang, Y. Liu, Y. Yao, Y. Xiao, Y. Zou, Y. Liu, and W. Huo. Effective redos detection by principled vulnerability modeling and exploit generation. In *44th IEEE Symposium on Security and Privacy(SP)*, pages 2427–2443. IEEE, 2023. doi:10.1109/SP46215.2023.10179328.

[257] I. Wegener. *The complexity of Boolean functions*. Wiley-Teubner, 1987.

[258] P. Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory, Iowa City, Iowa, USA, October 15-17, 1973*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13. URL `https://doi.org/10.1109/SWAT.1973.13`.

[259] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/J.TCS.2005.09.023.

[260] A. Wombacher, P. Fankhauser, B. Mahleko, and E. J. Neuhold. Matchmaking for business processes based on choreographies. *Int. J. Web Serv. Res.*, 1(4):14–32, 2004. doi:10.4018/JWSR.2004100102.

[261] T. Yokomori and F. Okubo. Theory of reaction automata: a survey. *J. Membr. Comput.*, 3(1):63–85, 2021. doi:10.1007/S41965-021-00070-6.

[262] Z. Zhang, L. Wu, A. Wang, Z. Mu, and X. Zhang. A novel bit scalable leakage model based on genetic algorithm. *Secur. Commun. Networks*, 8(18):3896–3905, 2015. doi:10.1002/SEC.1308.

Trieste, Italia
Santiago, Chile

October MMXXV