



# UNIVERSITÀ DI PISA

DEPARTMENT OF COMPUTER SCIENCE

MASTER DEGREE

## ENHANCING PREDICTOR ANALYSIS FOR REACTION SYSTEMS

Candidates

*Valeria Montagna*

*Pasquale Pulieri*

Supervisors

*Roberta Gori*

*Roberto Bruni*

Examiner

*Paolo Milazzo*

A.Y. 2021-2022



# Abstract

Natural Computing draws inspiration from biological mechanisms and processes for designing innovative computational models.

Reaction systems are a formal framework conceived to model how chemical reactions work. Using this formalism we are able to investigate practical problems by reasoning on a theoretical model.

An interesting aspect to look into is understanding the dynamic behaviour of a biochemical system. Reaction systems run in presence of an environment that provides new molecules inside the system at each step. A challenging question is which set of molecules must be observed in the environment to determine whether or not a substance  $s$  is produced by the system at step  $n$ .

A variant of this problem consists in investigating the  $n$ -th ancestors of a state where the substance  $s$  is introduced, namely the initial sets of reactants leading to the production of the target substance in  $n$  steps. Despite the deterministic nature of reaction system, many computational problems in this setting are proven to be intractable.

The objective of this thesis is to study and explore possible over-approximations, in order to increase the computational efficiency of  $n$ -th ancestors. Firstly, we define over-approximations. In particular, we propose an abstraction based on the idea of entities that have to be present and entities that may be present. Then a reinterpretation of the reaction system formalism will be proposed, arising from the idea of finding a common treatment for inhibitors and reactants, in order to keep only positive terms in the system. This allows us to consider reaction systems without inhibitors, that we call "positive". We apply the previously defined over approximation to standard reaction systems and to the "positive" reaction systems. Next, we develop a tool called MuMa Predictor, designed to compute the above transformation and approximations. In the end, some final results from appropriately constructed benchmarks will demonstrate the improvements we obtained with the proposed solutions.



# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Main results . . . . .	8
1.2	Structure of the thesis . . . . .	9
<b>2</b>	<b>Background</b>	<b>11</b>
2.1	Over Approximation . . . . .	11
2.2	Reaction Systems . . . . .	15
2.3	Dynamic causalities . . . . .	19
2.3.1	Causality in Reaction Systems . . . . .	19
2.3.2	Predictors . . . . .	19
2.3.3	Formula Based Predictors . . . . .	22
2.3.4	Ancestors and preimages . . . . .	25
<b>3</b>	<b>Over-Approximation</b>	<b>31</b>
3.1	Set of sets of literals over-approximation . . . . .	32
3.2	Predictor abstraction . . . . .	34
3.3	MuMa predictor concretization . . . . .	39
3.4	Formulae verification . . . . .	40
3.4.1	Cycles and fixed points . . . . .	43
3.5	Summarizing the process . . . . .	45
<b>4</b>	<b>Positive Reaction Systems</b>	<b>47</b>
4.1	Converting a Reaction System . . . . .	47
4.2	Positive reaction system analysis . . . . .	52
4.2.1	Space complexity . . . . .	52
4.2.2	Predictor computation . . . . .	52
<b>5</b>	<b>A tool for dealing with predictor</b>	<b>61</b>
5.1	General overview . . . . .	61
5.2	Implementation . . . . .	62
5.2.1	Must/Maybe set generation . . . . .	63
5.2.2	Generate all formulas and verification . . . . .	67

5.3	Experimental . . . . .	70
5.3.1	Benchmarks . . . . .	71
5.3.2	Simulation results . . . . .	74
<b>6</b>	<b>Conclusions</b>	<b>83</b>
6.1	Future works . . . . .	84
	<b>Appendices</b>	<b>91</b>

# Chapter 1

## Introduction

Natural Computing is a multidisciplinary research area that draws connections between computer science and the natural sciences, such as biology, chemistry or genetics. This field enables the development of new computational tools inspired by the observation of natural phenomena.

Reaction systems (RSs, for short) define an expressive computational formalism inspired by the concept of chemical reaction. As such, a RS is seen as a set of entities and a set of rules, called reactions. Each reaction is composed by a set of reactants necessary for the reaction to happen, a set of inhibitors whose aim is blocking the reaction and a set of products produced as result. The two key mechanisms on which interactions are based are facilitation and inhibition. Facilitation means that a reaction can happen when all its reactants are present. Instead inhibition means that the reaction cannot occur if any of its inhibitors is present. A reaction that has all reactants present and no inhibitors present is enabled.

The state of a reaction system is a finite set of objects, which can change over discrete time steps, by considering the reactions. The next state is determined by current state of the system: the presence of entities in the current states triggers all enabled reactions to be applied simultaneously. In fact, the amount of entities in the current state is always assumed to be sufficient to support the occurrence of each of enabled reaction.

An interesting question in the field of reaction systems dynamics concerns the study of causal relationships between entities, given the reactions that can generate them. Specifically, the problem is how to determine all possible set of objects that must initially be present to engage a transitions sequence, driven by reactions applicability, that results in a certain set of products  $P$  after a certain number of steps  $n$ . This sets of objects is called a predictor.

Intuitively, the idea is to start with the set of entities  $P$  that we want to obtain in  $n$  steps and, by reasoning backwards, to find out which possible sets

of entities enable the reactions that produce  $P$ . By continuing this process long  $n$  steps backward, we obtain all the initial entities that form the predictor. The problem is that depending on how complex the RS turns out to be, this backward process can be computationally hard.

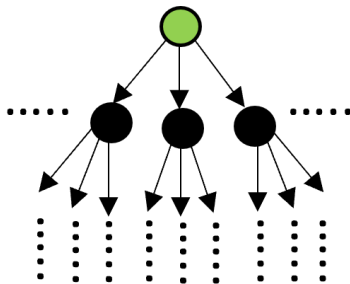


Figure 1.1: Computational tree for predictor backward process

The motivation lies in the growth, at each step, of the possible states from which one can start, as it can be seen from the Figure 1.1.

In cases where an exact solution to the problem is excessively time consuming, devising over-approximations can reduce the time complexity. To design an over-approximation we exploit abstract interpretation, a general framework originating from static program analysis techniques.

## 1.1 Main results

The main idea of over-approximation is based on the concept of synthesising all possibilities for one state at a specific execution step into two sets: the set of entities necessarily needed, called must set, and the set of entities possibly needed, called maybe set, for that state. We apply over-approximation to the concept of predictor, outlining a three-stage procedure comprising approximation, concretisation and formulas verification.

We then define a transformation on the reaction system, based on the idea that inhibitory entities can be considered as separate new entities. The new resulting reaction system will be called positive reaction system. The analysis of the characteristics of positive reaction systems allows us to define useful properties in order to speed up the verification phase of formulas.

We have developed a tool, called MuMa Predictor, that implements the above concepts. We also present two experiments from well-established studies in which we use the tool to study the behaviour of real complex systems and compare our results with those already known.



## 1.2 Structure of the thesis

Chapter 2 introduces all the concepts useful for understanding the techniques presented in the following chapters.

In chapter 3 we define the over-approximation, whose design is based on abstract interpretation methodologies.

Chapter 4 introduce a transformation of the reaction system structure into an inhibitor-free equivalent system, called positive reaction system.

The main implementation choices that enabled MuMa Predictor's development are explained in Chapter 5, together with performance analyses and comparative benchmarks that validate the conclusions drawn theoretically. In the final chapter we provide an overview of the conclusions of this work with a prospect of possible future directions.

We also point out that within the appendix we find a guide on how to use the tool.



# Chapter 2

## Background

In this chapter, we introduce some notions that will be useful in the future sections. Most of the notation adopted in subsequent chapters is presented here.

In particular we give a necessary introduction to over-approximations, the key feature of our proposal. We refer to a well known technique to design over-approximations: abstract interpretation.

Then reaction systems will be discussed: they are a novel formalism designed to model biochemical reactions behaviours.

We will introduce both the syntax and the semantics of reaction systems and we will move on explaining the concept of predictor: this idea comes from the study of the structural and dynamic causalities of a reaction system and is one way to represent the causal relationships between the entities involved in a biological process. In this context also the notion of preimage and  $n$ -th ancestor of a reaction system will be defined.

### 2.1 Over Approximation

Most natural optimization problems, including those coming from important application areas, are NP-hard and their exact solution is prohibitively time consuming. Approximation techniques are good for handling the limitations that come up when one has to manage a complex problem.

All the approximation algorithms have in common the aim to somewhat unraveling the relevant structure of the problem and finding techniques to exploit it.

Abstract interpretation [1, 2] computes approximations of variable domains over a relaxation of the initial problem. Originally, it was developed as a unifying framework for designing and then validating static program analysis, but today it is recognized as a general methodology for describing

and formalizing approximate computations in many different areas [4, 8, 5, 7, 6, 3].

The basic observation is that abstract interpretation defines an approximated semantics, obtained from the standard one, by substituting the actual domain of computation, namely concrete domains  $C$ , with an abstract domain  $A$ . Abstract domains will represent some relevant properties of interest extracted from concrete domains' values.

**EXAMPLE 2.1:** One might imagine treating the set of all motor vehicles as a concrete domain  $C$  and, for example, want to derive information about the number of wheels each concrete object has. Then the abstract domain resulting from this request is the set of integers from 0 to 10, i.e.  $A = \{0, \dots, 10\}$ .  $\blacklozenge$

Sets of elements of a concrete domain  $C$  can be considered as possible states of the system, while the approximate values are elements of an abstract domain  $A$ . Abstract and concrete domains are related to each other through a pair of total functions, defined below.

**Definition 2.1** (Abstraction and Concretization Function.). Let  $C$  and  $A$  be two domains, respectively called the concrete and the abstract domain.

- An abstraction function is a function  $\alpha : 2^C \rightarrow A$  that maps each set of concrete elements to its abstract representation.
- A concretization function is a function  $\gamma : A \rightarrow 2^C$  performs a mapping from abstract value to a set of concrete values.

**EXAMPLE 2.2:** Taking the Example 2.1, one might image to treat the following concrete domain:

$C = \{\text{motorized bicycle, dicycle, motorcycle, motorcycle with sidecar, automobile, go-cart, truck}\}$

Abstracting respect to the number of wheels means to design an abstraction function  $\alpha$  defined as following:

$$\alpha(c) = \begin{cases} 2 & \text{if } c \in \wp(\{\text{motorized bicycle, dicycle, motorcycle}\}) \\ 3 & \text{if } c \in \wp(\{\text{motorcycle with sidecar}\}) \\ 4 & \text{if } c \in \wp(\{\text{automobile, go-cart, truck}\}) \\ \text{undef} & \text{otherwise} \end{cases}$$

The concretization function for abstraction of number of wheel in

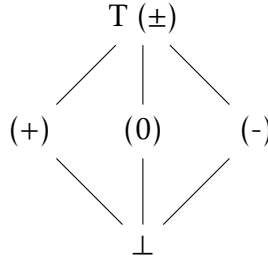
motorized vehicles will be:

$$\gamma(a) = \begin{cases} \{\text{motorized bicycle, dicycle, motorcycle}\} & \text{if } a = 2 \\ \{\text{motorcycle with sidecar}\} & \text{if } a = 3 \\ \{\text{automobile, go-cart, truck}\} & \text{if } a = 4 \\ C & \text{if } a = \text{undef} \end{cases}$$

◆

EXAMPLE 2.3: Let the abstraction be positive or negative numbers on set of real numbers  $\mathbb{R}$ .

Concrete domain will be  $\wp(\mathbb{R})$ , since we want to approximate sets of reals, while the abstract domain will be the signs domain  $\mathbb{D}_{+-} = \{\perp, +, -, \pm\}$  with the following lattice:



Abstraction function  $\alpha$  will be:

$$\alpha(C) = \begin{cases} \perp & \text{if } C = \emptyset \\ (0) & \text{if } C = \{0\} \\ (+) & \text{if } \forall c \in C : c > 0 \\ (-) & \text{if } \forall c \in C : c < 0 \\ (\pm) & \text{otherwise} \end{cases}$$

A few outputs of abstraction function are:

$$\alpha(\{-3.4, -1, -17\}) = (-)$$

$$\alpha(\{1, 7.35\}) = (+)$$

$$\alpha(\{-5, 2.7\}) = (\pm)$$

The concretization function for abstraction of positive and negative

numbers on  $\mathbb{R}$  returns such as displayed below:

$$\gamma(a) = \begin{cases} \emptyset & \text{if } a = \perp \\ \{0\} & \text{if } a = 0 \\ (0, \infty] & \text{if } a = (+) \\ (-\infty, 0) & \text{if } a = (-) \\ (-\infty, \infty) & \text{otherwise} \end{cases}$$

Starting from a set of reals, the application of  $\alpha$  followed by  $\gamma$  in this example don't return the set itself, but they return the set of all the values that fall into some intervals of either positive, negative numbers or not. Indeed, abstract interpretation will produce an over-approximation.  $\blacklozenge$

Moreover also operations on concrete values need to be translated into "native" counterparts which work on abstract values directly. An operation that has been transformed in this way is called abstract operation.

**EXAMPLE 2.4:** Continuing with the Example 2.3, the operation of product  $\times$  for concrete number can be translated into an abstract product  $\otimes$ , whose effects are resumed down here:

$$\begin{array}{llll} (+) & \otimes & (+) & = & (+) \\ (+) & \otimes & (-) & = & (-) \\ (-) & \otimes & (-) & = & (+) \\ (0) & \otimes & (+) & = & (0) \\ (0) & \otimes & (-) & = & (0) \\ (\pm) & \otimes & (+) & = & (\pm) \\ (\pm) & \otimes & (-) & = & (\pm) \\ (\pm) & \otimes & (0) & = & (0) \\ (\perp) & \otimes & a & = & (\perp) \quad \text{if } a = (+), (-), (\pm), (0) \end{array}$$

Abstract interpretation theory requires concrete and abstract semantics to be defined on domains that are partially ordered sets, in order to comparing two values.

**Definition 2.2** (Partially ordered set (poset)). A partially ordered set  $(S, \sqsubseteq)$  is a set  $S$  equipped with a partial order  $\sqsubseteq$ . A binary relation  $\sqsubseteq$  on  $S$  is a partial order if, for each  $x, y \in S$

- $x \sqsubseteq x$  (reflexivity)
- $x \sqsubseteq y, y \sqsubseteq x \implies x = y$  (antisymmetry)

- $x \sqsubseteq y, y \sqsubseteq z \implies x \sqsubseteq z$  (transitivity)

Intuitively, the pair of functions  $(\alpha, \gamma)$  defines the approximation as a path from sets of concrete values to an abstract values and viceversa. This concept is formalized as a Galois connection, which formally defines the approximation relation between  $\alpha$  and  $\gamma$ .

**Definition 2.3** (Galois connection). Let  $(C, \subseteq_C)$  and  $(A, \subseteq_A)$  be two posets. A pair of function  $(\alpha : 2^C \rightarrow A, \gamma : A \rightarrow 2^C)$  is a Galois insertion if and only if the following holds:

1.  $\alpha$  and  $\gamma$  are monotone, that is  $c_1 \subseteq_C c_2 \implies \alpha(c_1) \subseteq_A \alpha(c_2)$  and  $a_1 \subseteq_A a_2 \implies \gamma(a_1) \subseteq_C \gamma(a_2)$
2. For each  $c \in C$ ,  $c \subseteq_C \gamma(\alpha(c))$  and
3. For each  $a \in A$ ,  $\alpha(\gamma(a)) \subseteq_A a$

From an operator  $f$  on the concrete domain  $(C, \subseteq_C)$ , one can obtain the corresponding operator on the abstract domain  $(A, \subseteq_A)$  through the Galois connection.

**Definition 2.4.** Let  $(\alpha, \gamma)$  be a Galois connection of  $(C, \subseteq_C)$  into  $(A, \subseteq_A)$  and  $f : C \rightarrow C$ . A monotone abstract function  $\bar{f} : A \rightarrow A$  is a correct approximation of  $f$  if

$$\alpha(f(\gamma(d))) \subseteq_A \bar{f}(d)$$

Moreover,  $\bar{f}$  is an optimal correct approximation of  $f$  if, for each  $d \in C$ ,  $\alpha(f(\gamma(d))) = \bar{f}(d)$

## 2.2 Reaction Systems

Reaction systems [11, 17] are a formal framework inspired by natural phenomena for describing biochemical processes driven by interaction among reactions in living cells. The theory of RSs has been successfully applied in different fields, like computer science[23], theory of computing, mathematics, biology [20, 18, 25, 24], and molecular chemistry.

At the basis of the system mechanism there are biochemical reactions, which regulate the functioning of a living cell.

**Definition 2.5** (Reaction). Let  $S$  be a set of entities (or objects). A reaction over  $S$  is a triple  $a = (R, I, P)$ , with  $R, I, P \subseteq S$ ,  $R \neq \emptyset$  and  $R \cap I = \emptyset$ , composed by reactants  $R$ , inhibitors  $I$ , and products  $P$ .

Sometimes, sets  $R, I, P$  are also expressed with respect to the specific reaction to which they relate, namely  $R_a, I_a, P_a$  indicate respectively the reactant set, the inhibitor set, the product set of reaction  $a$ .

Reactants and inhibitors  $R \cup I$  of a reaction are collectively called resources of such a reaction. The set of reaction resources is denoted as  $M$ . As the definition points out, it is assumed that reactants and inhibitors of a reaction are disjointed sets, since otherwise the reaction would be inapplicable.

A reaction system is built in terms of a background set  $S$  of symbols, called entities, the molecular substances present in the states of a biochemical system, and a set of rewrite rules  $A$ , called reactions, representing biochemical reactions.

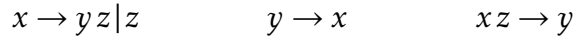
**Definition 2.6** (Reaction System (r.s. for short)). A reaction system is a pair  $\mathcal{A} = (S, A)$ , with  $S$  being a finite background set, and  $A$  being its set of reactions.

The set of reaction  $A$  is a subset of the set of all possible reactions obtained by  $S$ , denoted by  $rac(S)$ . Since  $S$  is finite, also  $rac(S)$  is finite and consequently  $A$  itself.

**EXAMPLE 2.5 (REACTION SYSTEM):** Suppose the background set is  $S = \{x, y, z\}$ . A biochemical reaction can be viewed in the form:

$$\{\text{Reactants}\} \rightarrow \{\text{Products}\} \mid \{\text{Inhibitors}\}$$

The example in question will have the following biochemical reactions:



These biochemical reactions will be transformed into the following set of reactions  $A$ :

$$A = \{a = (\{x\}, \{z\}, \{y, z\}), b = (\{y\}, \emptyset, \{x\}), c = (\{x, z\}, \emptyset, \{y\})\}$$

The ordered pair  $\mathcal{A} = (S, A)$  forms a reaction system. ♦

The applicability of the rules of a reaction system occurs exactly as it would in a biochemical reaction. The former may happen at a certain point if all of its reactants are present and none of its inhibitors is present. These two coexisting mechanisms are called "facilitation" and "inhibition", respectively. When this happens, a reaction creates its products.

**Definition 2.7** (Reaction Result). Let  $T$  be a finite set. Let  $a = (R_a, I_a, P_a)$  be a reaction. Then  $a$  is enabled by  $T$ , denoted by  $en_a(T)$ , if  $R_a \subseteq T$  and  $I_a \cap T = \emptyset$ .



The result of  $a$  on  $T$ , denoted by  $res_a(T)$ , is defined by:

$$res_a(T) = \begin{cases} P_a & \text{if } en_a(T) \\ \emptyset & \text{otherwise} \end{cases}$$

Expressing this notion for the entire reaction system, one obtains the following general definition:

**Definition 2.8** (Reaction System Result). Let  $T$  be a finite set. Let  $\mathcal{A} = (S, A)$  be a reaction system, with  $A$  a finite set of reactions. The result of  $A$  on  $T$ , denoted by  $res_A(T)$  is defined by:

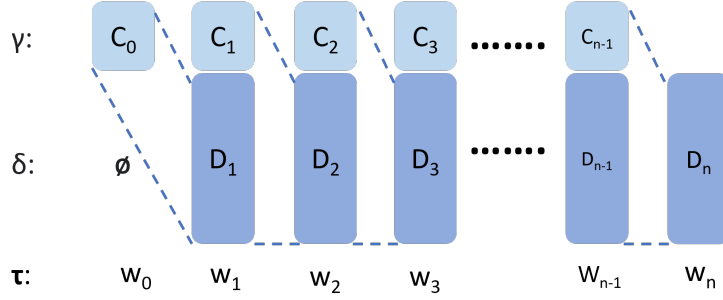
$$res_A(T) = \bigcup_{a \in A} res_a(T)$$

**EXAMPLE 2.6:** Consider the example presented in Example 2.5 and a finite set  $T = \{x, z\}$ . By analyzing reactions  $a$  and  $c$ , one can observe that  $R_a, R_c \subseteq T$ , but  $I_a \cap T = \{z\} \neq \emptyset$ , differently from  $I_c \cap T = \emptyset$ . Therefore, the only enabled reaction is  $c$ , that produces  $P_c = \{x\}$ . Instead, reaction  $b$  is not enabled because reactants set  $R_b$  is not contained in  $T$ . At the end,  $res_A(T) = res_a(T) \cup res_b(T) \cup res_c(T) = \emptyset \cup \emptyset \cup \{x\} = \{x\}$ . ♦

It should be noted that the concurrent applicability of multiple reactions occurs since no competition over the reagents is considered. If  $a, b \in A$  with both  $a$  and  $b$  enabled by  $T$ , then even if  $R_a \cap R_b \neq \emptyset$ , still both  $P_a \subseteq res_A(T)$  and  $P_b \subseteq res_A(T)$ . This reflects an assumption called "threshold supply" which means that if an object is present in the state, then it is guaranteed to be available in sufficient quantity for all enabled reactions to take place. Consequently, reaction systems are generally a qualitative model, not a quantitative one.

In terms of the reaction system dynamics, one other important feature concerns the "nonpermanence" of objects: objects present at any given moment, unless products of the same enabled reactions, merely vanish. This is typical of biology, namely that without chemical reactions to sustain it, a cell does not survive due to a lack of energy.

The dynamic behaviour of a reaction system is formalized through the notion of a discrete-time interactive process and context sequence. A context sequence, denoted by  $\gamma = C_0, \dots, C_n$  gives the possibility to add at each step contextual elements provided by the external environment. In other words, it models the interaction of reaction systems with other systems. Such elements, joined to the other elements resulting from the applicability of reactions on the elements the previous step, will constitute the new state of interacting objects.



**Definition 2.9** (Interactive Process). An  $(n\text{-step})$  interactive process in  $\mathcal{A}$  is a pair  $\pi = (\gamma, \delta)$  of finite sequences where  $\gamma$  is called context sequence and  $\delta$  is called result sequence.

Sequences  $\gamma = C_0, \dots, C_n$  and  $\delta = D_0, \dots, D_n$  are such that for some  $n \geq 1$ , with  $C_i, D_i \subseteq S$ ,  $D_0 = \emptyset$  and  $D_i = \text{res}_{\mathcal{A}}(D_{i-1} \cup C_{i-1})$  for all  $i \in \{1, \dots, n\}$ .

Of course, this notion defines a discrete-time dynamical system. The actual state  $k$  of a biochemical system is the set of biochemical entities present in the current biochemical environment  $C_k$ , together with the set of those entities naturally produced by the applicable reactions  $D_k = \text{res}_{\mathcal{A}}(C_{k-1} \cup D_{k-1})$ . Intuitively, substances present at step  $k$  rely on what was produced at step  $k-1$  that is  $D_{k-1}$ , namely the outcome of reactions enabled by substances existing at  $D_{k-2} \cup C_{k-2}$ , and those injected at context  $C_{k-1}$ .

The sequence given by the union, step by step, of the context sequence with result sequence is the sequence  $\tau = W_0, \dots, W_n$ , with  $W_i = C_i \cup D_i$  for all  $i \in \{0, \dots, n\}$ . Sequence  $\tau$  is referred to as state sequence of  $\pi$ , with  $W_0 = C_0$  called the initial state of  $\pi$ . Observe that it must be  $D_0 = \emptyset$ , because initially entities cannot come as a product of previously enabled reactions; the trigger is activated only by substances introduced from the context.

When  $C_i \subseteq D_i$  (or alternatively  $C_i = \emptyset$ ) for all  $i \in \{1, \dots, n\}$ , interactive process  $\pi$  is called context-independent, or closed.

**EXAMPLE 2.7:** Consider the reaction system  $\mathcal{R} = (\{A, B, C, D\}, R)$ , where  $R$  is the following set of six reactions:

$$(\{C\}, \{A, B\}, \{A, B, D\}), \quad (\{B\}, \{A\}, \{A\}), \quad (\{B\}, \{C\}, \{C\})$$

$$(\{A\}, \{C\}, \{B\}), \quad (\{D\}, \{C\}, \{A, B\}), \quad (\{A, C\}, \{B, D\}, \{B, C\})$$

The sequence  $\tau = \{A, B, D\}, \{A, B, C\}, \emptyset$  is a context-independent state sequence of  $\mathcal{R}$ . As an example, notice that  $W_1 = C_1 \cup D_1 = \emptyset \cup \text{res}_{\mathcal{R}}(\{A, B, D\}) = \{A, B, C\}$ .

Instead the sequence  $\tau = \{A, B, D\}, \{A, B, C, D\}, \{C\}$  is a context-

dependent state sequence, in fact:

$$\begin{aligned} W_1 &= C_1 \cup D_1 = \{D\} \cup \text{res}_{\mathcal{R}}(\{A, B, D\}) = \{A, B, C, D\} \\ W_2 &= C_2 \cup D_2 = \{C\} \cup \text{res}_{\mathcal{R}}(\{A, B, C, D\}) = \{C\} \end{aligned}$$

◆

Context-independent interactions define the semantics of a closed reaction system, the dynamics of which are dictated merely by what is in the initial state, and the process evolution is simplified to just the result sequence.

## 2.3 Dynamic causalities

### 2.3.1 Causality in Reaction Systems

Understanding which are the causal relationships between the actions in biology system is a crucial issue [16, 19, 21, 12, 10, 26]. The study of causality in reaction systems[15], that is, the ways in which different entities influence each other, is important for understanding how system entities work.

Some of these dependencies can be captured by looking at the reaction system specification. These types of causalities are called static causalities. For example, let  $x$  and  $y$  be two entities and let  $b$  be a reaction such that  $y \in M_b$  (meaning  $y$  is either an inhibitor or a reagent) and  $x \in P_b$ , then it can be inferred that the production of  $x$  depends on the presence/absence of  $y$  (and possibly on other entities). It is said that  $x$  is resource dependent on  $y$  and that  $y$  product-influences  $x$ . Through these easily deducible dependencies, for instance, one can derive a very convenient tool for the analysis of causalities, the influence graph[14].

Other types of causalities can be classified as dynamic, since they are defined in terms of relationships formed through the dynamic runs of a reaction system. Again, interesting technical results can be achieved, such as reaching to know the causal distance between two entities.

The concept of predictor captures dynamic causality: the idea is of knowing the ways of producing a particular set of entities at a certain step of the system's dynamical process.

### 2.3.2 Predictors

Brijder, Ehrenfeucht and Rozenberg[15] introduced the notion of predictor.

Let  $\mathcal{A} = (S, A)$  be a reaction system. We are interested in knowing whether or not, given a  $n \in \mathbb{Z}^+$ , a specific entity  $x \in S$  will be present in the final state after  $n$ -steps.

It may be the case, during the interactive process, contextual elements are introduced in certain steps: these entities are the only source of non-determinism in reaction systems. Knowing which elements will be received at each step permits to predict the creation of  $s \in S$  after  $n$  steps.

However, the point is that not all contextual elements are essential to determine whether  $s$  will be produced after  $n$  steps. It is sufficient to derive a subset  $Q$  of  $S$  which will cause  $s$  to be or not be produced after  $n$  steps.

Let  $\tau = W_0, \dots, W_n$  be a sequence of sets. For a set  $S$ , we say that  $\tau$  is an  $S$ -sequence if  $W_i \subseteq S$  for all  $i = 0, \dots, n$ . Moreover, for a set  $Q$ , the  $Q$ -projection of  $\tau$  is the  $Q$ -sequence of sets

$$proj_Q(\tau) = W_0 \cap Q, \dots, W_n \cap Q$$

We say that  $Q \subseteq S$   $n$ -predicts  $x \in S$  if for any two interactive process it holds that if the projections on  $Q$  of their context sequences is equal, then either  $x$  is in the  $n+1$ -th context state of both the interactive processes, or in none of them. More formally:

**Definition 2.10** ( $n$ -predicts). Let  $\mathcal{A} = (S, A)$  be a reaction system. For  $x \in S$ ,  $n \geq 1$  and  $Q \subseteq S$ ,  $Q$   $n$ -predicts  $x$  if for an arbitrary  $n$ -steps interactive process  $\pi_1 = (\gamma_1, \delta_1)$  and  $\pi_2 = (\gamma_2, \delta_2)$  the following holds:

$$proj_Q(\gamma_1) = proj_Q(\gamma_2) \implies (x \in W_{n+1}^1 \iff x \in W_{n+1}^2)$$

**Definition 2.11** ( $n$ -steps Predictor of  $s$ ). Let  $\mathcal{A} = (S, A)$  a reaction system, then a predictor of  $s \in S$  after  $n$  steps is defined as:

$$\mathcal{P}_{x,n} = \{Q \subseteq S \mid Q \text{ } n\text{-predicts } x\}$$

Following this idea, one could exploit the notion of predictor, in order to decide whether  $s$  will appear or not after  $n$  steps, without executing the reaction system.

Pushing forward the original idea Barbuti et al.[24] introduced the notions of formula based predictor.

In an  $n$ -step interactive process  $\pi = (\gamma, \delta)$  of a closed reaction system  $\mathcal{A}$ , since in  $\gamma$  the only initial context  $C_0$  is non-empty, while  $C_1 = \emptyset, \dots, C_n = \emptyset$ , given the predictor of  $x$  in the reaction system  $\mathcal{A}$   $prd_{\mathcal{A}}(x, n)$ , the problem of knowing if  $x$  will be produced after  $n$  steps is reduced to knowing which entities of  $prd_{\mathcal{A}}(x, n)$  are included in  $C_0$ .

We now introduce the concept of formula based predictors, after some preliminary definitions.

The set of propositional formulas on  $S$   $F_S$  is defined in a standard way:  $S \cup \{true, false\} \subseteq F_S$ ,  $\neg f_1, f_1 \wedge f_2, f_1 \vee f_2 \in F_S$  if  $f_1, f_2 \in F_S$ .

The propositional formulas  $F_S$  have to be interpreted with respect to subsets of objects  $O \subseteq S$ . The wording  $s \in O$  means intuitively that  $s$  is present and it is equivalent to the truth of the corresponding propositional symbol. The full definition of satisfiability of a propositional formula can be given:

**Definition 2.12.** Let  $O \subseteq S$  for a set of objects  $S$ . Given a propositional formula  $f \in F_S$ , the satisfaction relation  $O \models f$  is inductively defined as follows:

$$\begin{array}{ll} O \models \neg f' \text{ iff } O \not\models f' & \\ O \models true & \\ O \models s \text{ iff } s \in O & O \models f_1 \vee f_2 \text{ iff } O \models f_1 \text{ or } O \models f_2 \\ O \models (f') \text{ iff } O \models f' & O \models f_1 \wedge f_2 \text{ iff } O \models f_1 \text{ and } O \models f_2 \end{array}$$

In the following, it is indicated the logical equivalence on propositional formulas  $F_S$  as  $\equiv_l$ . Moreover, given a formula  $f \in F_S$ , with  $atom(f)$  it is indicated the set of propositional symbols that appear in  $f$ .

The requirements for the applicability of a reaction  $a$  are seen as the conjunction of all atomic formulas representing reactants  $R_a$  and the negations of all atomic formulas representing inhibitors  $I_a$ .

**Definition 2.13.** Let  $a = (R, I, P)$  be a reaction with  $R, I, P \subseteq S$  for a set of objects  $S$ . The applicability predicate of  $a$ , denoted by  $ap(a)$  is defined as:

$$ap(a) = \left( \bigwedge_{s_r \in R} s_r \right) \wedge \left( \bigwedge_{s_i \in I} \neg s_i \right)$$

EXAMPLE 2.8: Let  $\mathcal{A} = (\{A, \dots, G\}, \{a_1, a_2, a_3\})$ , be a reaction system with:

$$a_1 = (\{A, B\}, \emptyset, \{C\}), a_2 = (\{C, D\}, \emptyset, \{E, F\}), a_3 = (\{A\}, \{G\}, \{E\})$$

The applicability predicates of the reactions are:

$$\begin{array}{l} ap(a_1) = A \wedge B \\ ap(a_2) = C \wedge D \\ ap(a_3) = A \wedge \neg G \end{array}$$

◆

Similarly, the cause for a given object  $s$  is a propositional formula on  $S$  that represents the fact that it must be applicable at least one reaction having  $s$  as product. In other words, the cause of an object  $s$  in a reaction system  $\mathcal{A}$  is the disjunction of the applicability predicates related to individual reactions having  $s$  as a product.

**Definition 2.14.** Let  $\mathcal{A} = (S, A)$  be a reaction system and  $s \in S$ . The causal predicate of  $s$ , denoted by  $cause(s, \mathcal{A})$  (or  $cause(s)$  when  $\mathcal{A}$  is clear from the context), is defined as:

$$cause(s, \mathcal{A}) = \bigvee_{\{a \in A \mid s \in P\}} ap(a)$$

EXAMPLE 2.9: Reconsider reaction system  $\mathcal{A}$  in Example 2.8.

Then the causal predicates of the entities are:

$$\begin{aligned} cause(A) &= cause(B) = cause(D) = cause(G) = false \\ cause(C) &= A \wedge B, \quad cause(F) = C \wedge D, \quad cause(E) = (C \wedge D) \vee (A \wedge \neg G) \end{aligned}$$

Where the causal predicates give as result *false* it has to be interpreted as that entity cannot be produced by any reaction.  $\blacklozenge$

### 2.3.3 Formula Based Predictors

The notion of formula based predictor was originally presented in [24]. A formula based predictor for an object  $s$  at step  $n + 1$  is a propositional formula which must be satisfied exactly by the context sequences leading to the production of  $s$  at step  $n + 1$ .

Given a set of entities  $S$ , consider a corresponding set of labelled objects  $S \times \mathbb{N}$ . A labelled object  $(s, i) \in S \times \mathbb{N}$ , or more simply  $s_i$ , denotes the presence (or the absence, if negated) of object  $s$  in the element  $C_i$  of the context sequence  $\gamma = C_0, \dots, C_n$ .

EXAMPLE 2.10:  $B_3$  denotes in a context sequence the presence of entity  $B$  in the third element  $C_3$  of the context sequence  $\gamma = C_0, \dots, C_n$ .  $\blacklozenge$

The set of all possible entities for each execution step from 0 to  $n$  is denoted by  $S^n$ . Formally, it is  $S^n = \bigcup_{i=0}^n S_i$ , where  $S_i = \{s_i \mid s \in S\}$ . Then the set of propositional formulae having as literals objects in  $S^n$  will be indicated as  $F_{S^n}$ .

Clearly, a logic formula on  $S^n$  describes the properties of  $n$ -step context sequences. The following definition describes how context sequences and propositional formulas are related.

**Definition 2.15.** Let  $\gamma = C_0, \dots, C_n$  a context sequence and  $f \in F_{S^n}$  a propositional formula. It is said that  $\gamma$  satisfies  $f$  ( $\gamma \models f$ ), where the satisfaction relation  $\models$  is inductively defined as follows:

$$\begin{aligned} \gamma \models \text{true} & & \gamma \models \neg f' \text{ iff } \gamma \not\models f' \\ \gamma \models s_i \text{ iff } s \in C_i & & \gamma \models f_1 \vee f_2 \text{ iff } \gamma \models f_1 \text{ or } \gamma \models f_2 \\ \gamma \models (f') \text{ iff } \gamma \models f' & & \gamma \models f_1 \wedge f_2 \text{ iff } \gamma \models f_1 \text{ and } \gamma \models f_2 \end{aligned}$$

EXAMPLE 2.11: Consider the context sequence  $\gamma = C_0, C_1$ , where  $C_0 = \{A, C\}$  and  $C_1 = \{B\}$ :  $\gamma$  satisfies the formula  $A_0 \wedge B_1$  (i.e.  $\gamma \models A_0 \wedge B_1$ ), while  $\gamma$  does not satisfy the formula  $A_0 \wedge (\neg B_1 \vee C_1)$  (i.e.  $\gamma \not\models A_0 \wedge (\neg B_1 \vee C_1)$ ).  $\blacklozenge$

The latter notion allows to define formula based predictor.

**Definition 2.16** (Formula-based predictor). Let  $\mathcal{A} = (S, A)$  be a reaction system,  $s \in S$  and  $f \in F_{S^n}$ . It is said that  $f$  f-predicts  $s$  in  $n+1$  steps if for any  $n$ -step context sequence  $\gamma = C_0, \dots, C_n$

$$\gamma \models f \iff s \in D_{n+1}$$

where  $\delta = D_0, \dots, D_n$  is the result sequence and  $D_{n+1} = \text{res}_{\mathcal{A}}(C_n \cup D_n)$ .

The satisfaction relation  $\models$  induces also the logical equivalence.

**Definition 2.17.** Given  $f, f' \in F_{S^n}$ ,  $f \equiv_l f'$  iff for all context sequences  $\gamma$ , it holds ( $\gamma \models f \iff \gamma \models f'$ )

Of course, if a formula  $f$  f-predicts  $s$  in  $n+1$  steps and if  $f' \equiv_l f$ , then also  $f'$  f-predicts  $s$  in  $n+1$  steps. Clearly, the aim is to find the formula with the minimum number of propositional symbols that would make it easier to check satisfiability. That is ensured by the following definition of order of approximation on  $F_{S^n}$ .

**Definition 2.18.** Given  $f_1, f_2 \in F_{S^n}$ ,  $f_1 \sqsubseteq f_2$  if and only if  $f_1 \equiv_l f_2$  and  $\text{atom}(f_1) \subseteq \text{atom}(f_2)$ .

The formula based predictor is computed through an operator fbp, defined below.

**Definition 2.19** (Formula-based predictor operator). Let  $\mathcal{A} = (S, A)$  be a reaction system and  $s \in S$ . Let fbp a function  $\text{fbp} : \mathbb{N} \rightarrow F_{S^n}$  defined as follows:

$$\text{fbp}(s, n) = \text{fbs}(\text{cause}(s), n)$$

where  $\text{fbs} : F_S \times \mathbb{N} \rightarrow F_{S^n}$  is an auxiliary function recursively defined as follows:

$$\begin{aligned}
& \text{fbs}(s, 0) = s_0 & \text{fbs}(\neg f', i) &= \neg \text{fbs}(f', i) \\
& \text{fbs}(s, i) = s_i \vee \text{fbs}(\text{cause}(s), i - 1) & \text{fbs}((f'), i) &= (\text{fbs}(f', i)) \\
& \text{fbs}(f_1 \wedge f_2, i) = \text{fbs}(f_1, i) \wedge \text{fbs}(f_2, i) & \text{fbs}(\text{true}, i) &= \text{true} \\
& \text{fbs}(f_1 \vee f_2, i) = \text{fbs}(f_1, i) \vee \text{fbs}(f_2, i) & \text{fbs}(\text{false}, i) &= \text{false}
\end{aligned}$$

EXAMPLE 2.12: Consider the reaction system  $\mathcal{B} = (\{A, B, C, D\}, \{a_1, a_2, a_3, a_4\})$ , with the following reactions:

$$\begin{aligned}
a_1 &= (\{A\}, \emptyset, \{B, D\}), a_2 = (\{C\}, \emptyset, \{B\}) \\
a_3 &= (\{D\}, \{B\}, \{A\}), a_4 = (\{D\}, \emptyset, \{C\})
\end{aligned}$$

One can compute the logic formula that f-predicts  $B$  in 3 steps, for example.

By applying the function fbp, the following result is obtained:

$$\begin{aligned}
\text{fbp}(B, 2) &= \text{fbs}(A \vee C, 2) \\
&= \text{fbs}(A, 2) \vee \text{fbs}(C, 2) \\
&= A_2 \vee \text{fbs}(D \wedge \neg B, 1) \vee C_2 \vee \text{fbs}(D, 1) \\
&= A_2 \vee (\text{fbs}(D, 1) \wedge \neg \text{fbs}(B, 1)) \vee C_2 \vee D_1 \vee \text{fbs}(A, 0) \\
&= A_2 \vee ((D_1 \vee \text{fbs}(A, 0)) \wedge \neg(B_1 \vee \text{fbs}(C, 0))) \vee C_2 \vee D_1 \vee \text{fbs}(A, 0) \\
&= A_2 \vee ((D_1 \vee A_0) \wedge \neg(B_1 \vee C_0)) \vee C_2 \vee D_1 \vee A_0 \\
&= A_2 \vee ((D_1 \vee A_0) \wedge \neg B_1 \wedge \neg C_0) \vee C_2 \vee D_1 \vee A_0
\end{aligned}$$

To see the formula more clearly, one could apply boolean distributive laws and obtain:

$$\text{fbp}(B, 2) = A_2 \vee (D_1 \wedge \neg B_1 \wedge \neg C_0) \vee (A_0 \wedge \neg B_1 \wedge \neg C_0) \vee C_2 \vee D_1 \vee A_0$$

The resulting formula is in Disjunctive Normal Form (DNF), since it is a disjunction (sequence of ORs) consisting of one or more disjuncts, each of which is a conjunction (AND).  $\blacklozenge$

Of course the main property of fbp consists on the fact that any  $n$ -step context sequence satisfies  $\text{fbp}(s, n)$  if and only if the object  $s$  will appear in the system after  $n + 1$  steps.

**Theorem 2.20.** *Let  $\mathcal{A} = (S, A)$  be a reaction system and  $s \in S$ . For any  $n$ -step context sequence  $\gamma = C_0, C_1, \dots, C_n$  it holds:*

$$s \in D_{n+1} \iff \gamma \models \text{fbp}(s, n)$$



where  $\delta = D_0, \dots, D_n$  is the result sequence and  $D_{n+1} = \text{res}_{\mathcal{A}}(C_n \cup D_n)$ .

**EXAMPLE 2.13:** Some examples of (minimal) context sequences satisfying the formula  $\text{fbp}(B, 2)$  computed in Example 2.12 are:  $\gamma_1 = \{A\}, \{\}, \{\}$ , or  $\gamma_2 = \{\}, \{\}, \{C\}$ , or else  $\gamma_3 = \{\}, \{D\}, \{\}$ . Other examples are  $\gamma_4 = \{A, B\}, \{D\}, \{\}$  and  $\gamma_5 = \{A\}, \{A, C\}, \{D\}$ .  $\blacklozenge$

**Lemma 2.21.** Let  $\mathcal{A} = (S, A)$  be a reaction system,  $s \in S$  and  $f \in F_{S^n}$  be a propositional logical formula. If for all  $n$ -step context sequences  $\gamma$  it holds ( $s \in D_{n+1} \iff \gamma \models f$ ), then  $f$   $f$ -predicts  $s$  in  $n + 1$  steps, where  $D_{n+1} = \text{res}_{\mathcal{A}}(C_n \cup D_n)$ .

**Corollary 2.22.** Let  $\mathcal{A} = (S, A)$  be a reaction system. For any object  $s \in S$ , the formula  $\text{fbp}(s, n)$   $f$ -predicts  $s$  in  $n + 1$  steps.

In general, the function given at Definition 2.19 computes a formula based predictor not minimal with respect to approximation order  $\sqsubseteq_f$  defined in 2.18. However, it is possible to obtain a minimal version of formula based predictor by applying a standard simplification procedure to the obtained logic formula.

**EXAMPLE 2.14:** Consider the final version of the formula  $\text{fbp}(B, 2)$  in Example 2.12. This is not minimal. Again, one more simplification that can be made stands out: conjunctions  $(D_1 \wedge \neg B_1 \wedge \neg C_0)$  and  $(A_0 \wedge \neg B_1 \wedge \neg C_0)$  can be discarded, since they are seen as overspecifications respectively of  $D_1$  and  $A_0$ , conjunctions also present in formula.

Finally formula that  $f$ -predicts  $B$  in 3 steps will be:

$$\text{fbp}(B, 2) = A_2 \vee C_2 \vee D_1 \vee A_0$$

This last formula is minimal with respect to  $\sqsubseteq_f$ , in fact it cannot further simplified and any literal cannot be canceled without obtaining a non equivalent formula.  $\blacklozenge$

### 2.3.4 Ancestors and preimages

The concept of preimage and  $n$ -th ancestor was introduced by Dennunzio, Formenti and Manzoni[22].

The formula-based predictor can be tailored to the specific case of closed reaction systems. Recall that a closed reaction system is a reaction system  $\mathcal{A} = (S, A)$ , whose dynamics is characterized simply by its result sequence  $\delta = D_1, \dots, D_n$ , where formally  $D_i = \text{res}_{\mathcal{A}}(D_{i-1})$  for all  $1 \leq i < n$ . What will

be produced at step  $n$  depends on the initial set  $D_0$ , after which the system will be left free to evolve without explicitly injecting new instances during the steps of execution.

**Definition 2.23** ( $n$ -th Ancestors for  $s$ ). Let  $\mathcal{A} = (S, A)$  be a reaction system and  $s \in S$ . A set  $D_0$  is an  $n$ -th ancestor of  $s$  if  $s \in res_{\mathcal{A}}^{(n)}(D_0)$ .  $D_0$  is a preimage of  $s$  if it is a 1-st ancestor of  $s$ .

It can be defined a formula characterizing all the initial sets  $D_0$  that after  $n$  steps output a given product  $s$ . As anticipated, the new definitions can be derived from the notions of formula based predictors.

**Definition 2.24** ( $n$ -th Ancestor Formula). Let  $\mathcal{A} = (S, A)$  be a reaction system,  $s \in S$  and  $f \in F_S$  a propositional formula. A formula  $f$  is a  $n$ -th ancestor formula of  $s$  if it holds that

$$D_0 \models f \iff s \in D_n$$

As before, there cannot exist two non-equivalent formulas  $f$  and  $f'$ , both satisfying  $n$ -th Ancestor Formula definition. But if  $f$  is a  $n$ -th ancestor formula of  $s$  and there exist a formula  $f'$  such that  $f' \equiv f$ , then also  $f'$  is an  $n$ -th ancestor formula of  $s$ . Also for this concept it can be formalized an approximation order on  $F_S$ .

The computation of  $n$ -th ancestor formula is allowed by an operator called Anc.

**Definition 2.25.** Let  $\mathcal{A} = (S, A)$  be a reaction system and  $s \in S$ . Let Anc a function  $\text{Anc}: S \times \mathbb{N} \rightarrow F_S$  defined as follows:

$$\text{Anc}(s, n) = \text{Anc}_a(\text{cause}(s), n - 1)$$

Let  $\text{Anc}_a: F_S \times \mathbb{N} \rightarrow F_S$  an auxiliary function recursively defined as follows:

$$\begin{aligned} \text{Anc}_a(\neg f', i) &= \neg \text{Anc}_a(f', i) \\ \text{Anc}_a(l, 0) &= l \text{ where } l = s \text{ or } l = \neg s & \text{Anc}_a((f'), i) &= (\text{Anc}_a(f', i)) \\ \text{Anc}_a(s, i + 1) &= \text{Anc}_a(\text{cause}(s), i) & \text{Anc}_a(\text{true}, i) &= \text{true} \\ \text{Anc}_a(f_1 \wedge f_2, i) &= \text{Anc}_a(f_1, i) \wedge \text{Anc}_a(f_2, i) & \text{Anc}_a(\text{false}, i) &= \text{false} \\ \text{Anc}_a(f_1 \vee f_2, i) &= \text{Anc}_a(f_1, i) \vee \text{Anc}_a(f_2, i) \end{aligned}$$

The  $n$ -th ancestor formula given by operator defined in 2.25 in general also in this case may not be minimal with respect to  $\sqsubseteq_f$ . However, one could apply heuristic techniques to produce prime and irredundant quasi minimal DNF (Disjunctive Normal Form), for guaranteeing that formula will be minimal with respect to  $\sqsubseteq_f$ .

**Theorem 2.26.** Let  $\mathcal{A} = (S, A)$  be a reaction system. For any object  $s \in S$ ,

- $\text{Anc}(s, n)$  is the  $n$ -th ancestor formula of  $s$
- $\min(\text{Anc}(s, n))$  is the  $n$ -th ancestor formula of  $s$  and is minimal w.r.t.  $\sqsubseteq_f$

### Space Complexity for $n$ -Ancestor Formula Computation

Let  $cp(A)$  be the maximum number of products and inhibitors in all reactions the reaction system. Moreover, denote the maximum number of rules sharing a product with  $mp(A)$ .

**Definition 2.27.** Given a reaction system  $\mathcal{A} = (S, A)$ , let:

- $cp(A) = \max\{|R| + |I| \mid (R, I, P) \in A\}$
- $mp(A) = \max\{|p(A, s)| \mid s \in S\}$ , where  $p(A, s) = \{(R, I, P) \in A \mid s \in P\}$

For each  $s \in S$ , the size of  $\text{cause}(s)$  in terms of number of literals is at most  $cp(A) \cdot mp(A)$ . To obtain the  $n$ -ancestors formula they are required  $n$  steps. At first step, the formula will have size  $cp(A) \cdot mp(A)$ . At the second step, each one of the  $cp(A) \cdot mp(A)$  literals will be substituted with its causes, so the new formula will have size  $(cp(A) \cdot mp(A))^2$ . Therefore, the size of  $\text{Anc}(s, n)$  is at most  $(cp(A) \cdot mp(A))^n$  for each  $s \in S$ . If one wanted to know the size of the  $n$ -ancestors formula for the set product  $\{s_1, s_2, \dots, s_m\} \subseteq S$ , it would be  $m \cdot (cp(A) \cdot mp(A))^n$ . In both cases, the size is polynomial to  $cp(A)$  and  $mp(A)$ .

**Definition 2.28.** Given a reaction system  $\mathcal{A} = (S, A)$ , let

$$c(A) = \begin{cases} 1 & \text{if } cp(A) > 1 \\ 0 & \text{otherwise} \end{cases} \quad p(A) = \begin{cases} 1 & \text{if } mp(A) > 1 \\ 0 & \text{otherwise} \end{cases}$$

For each  $s \in S$ , the nesting level of the formula  $\text{Anc}(s, n)$ , characterizing the  $n$ -th ancestors of  $s$  is at most  $n \cdot (c(A) + p(A))$ .

### Time Complexity Issues

One issue to be addressed concerns the transformation of the  $n$ -Ancestor Formula  $f$  into a DNF of it. This is fundamental for executing the logic minimization step, consisting of deriving a DNF minimal with respect to the number of conjunctions or to the number of literals occurring in it. This last form is convenient to have in order to consider a compact representation of  $f$  so that it can be more easily verified. In general, the following (exact) procedure is applied:

1. Put the negation next to the atomic objects using De Morgan's laws
2. Put the conjunctions within the disjunctions using the distributive law
3. Simplify the formula using idempotent, negation and domination laws

The problem is the application of the distributive laws, whose time complexity can be exponential. To overcome such a thing, it is possible to apply heuristic techniques, e.g. ESPRESSO heuristic minimizer, to produce near minimal prime and irredundant DNF formulas.

It is interesting to study the complexity of checking the existence of preimages and  $n$ -th ancestors, and of computing minimal preimages and  $n$ -th ancestors.

The first problem consists of checking if a  $n$ -th ancestor of a given product exists, while the second one is about finding a  $n$ -th ancestor with a minimal number of instances.

In some particular cases these problem can be solved in polynomial time. With  $n = 1$ , that is the case of preimages, both problems can be solved in polynomial time, because formula  $\text{Anc}(s, 1)$  is already in DNF.

One first condition that assure that existence and minimal size  $n$ -th ancestor, with  $n > 1$ , can be computed in polynomial time is linear dependency.

**Definition 2.29.** Let  $\mathcal{A} = (S, A)$  be a reaction system. The  $n$ -linear dependency of an instance  $y$  from an instance  $x$ , denoted as  $x \hookrightarrow^n y$ , is recursively defined as follows:

1.  $x \hookrightarrow^1 y$  iff  $p(A, s) = 1$  and either  $(\{x\}, \emptyset, \{y\}) \in A$  or  $(\emptyset, \{x\}, \{y\}) \in A$
2.  $x \hookrightarrow^n y$  with  $n > 1$  iff there exists  $k \in S$  such that  $x \hookrightarrow^1 k$  and  $k \hookrightarrow^{n-1} y$

In practice  $y$  is  $n$ -linearly dependent from  $x$  if there exists a unique way to produce  $y$  from  $x$ , by producing a single element at each step.

A second property states when an instance is  $n$ -linearly produced.

**Definition 2.30.** Let  $\mathcal{A} = (S, A)$  be a reaction system. An instance  $y$  is  $n$ -linearly produced in  $\mathcal{A}$  iff

- $|p(A, y)| = 0$ , or
- there exists  $x \in S$  such that  $x \hookrightarrow^1 y$  and  $x$  is  $(n - 1)$ -linearly produced

In general, an object is linearly produced when it is  $n$ -linearly produced for all  $n$ .

**Theorem 2.31.** Let  $\mathcal{A} = (S, A)$  be a reaction system. If  $\forall s \in S. |p(A, s)| \leq 1$ , and for every rule  $(R, I, P) \in A$  all the objects in  $I$  are  $n$ -linearly produced, then, for any  $s \in S$ , the existence and minimal size of the  $n$ -th ancestors of  $s$  can be solved in polynomial time.

EXAMPLE 2.15: Let  $\mathcal{A} = (\{A, \dots, E\}, \{a_1, \dots, a_5\})$  be a r.s. with the following reactions:

$$\begin{aligned} a_1 &= (\{B\}, \emptyset, \{C\}) & a_2 &= (\{D, E\}, \{B\}, \{A\}) & a_3 &= (\{C\}, \emptyset, \{B\}) \\ a_4 &= (\{E\}, \emptyset, \{E\}) & a_5 &= (\{A, C\}, \emptyset, \{D\}) \end{aligned}$$

Every object is produced by at most one rule, moreover the inhibitor  $B$  is linearly produced by a reaction with one reactant, which in turn is linearly produced by a reaction with a single reactant. This means that conditions introduced in Theorem 2.31 are both satisfied. All ancestors computed by  $\text{Anc}$  will be already in DNF. For example, if we compute  $\text{Anc}(A, 2)$ :

$$\begin{aligned} \text{Anc}(A, 2) &= \text{Anc}_a(D, 1) \wedge \text{Anc}_a(E, 1) \wedge \neg \text{Anc}_a(B, 1) \\ &= \text{Anc}_a(A, 0) \wedge \text{Anc}_a(C, 0) \wedge \text{Anc}_a(E, 0) \wedge \neg \text{Anc}_a(C, 0) \\ &= A \wedge C \wedge E \wedge \neg C \end{aligned}$$

The propositional formula obtained is already in DNF.  $\blacklozenge$

Under these syntactic conditions, the computation of the minimal ancestor of a given state is solvable in polynomial time, but these constraints cut out equally interesting classes of reaction systems.

The problem is that in a free, unrestricted reaction system, the size of the ancestor formula grows quickly. Suppose that to produce  $s$  there are  $k$  reactions to be enabled. Start by wanting to know what it would take to produce  $s$  at step  $n$ : it would require that either all reactants of reaction 1 to be present without any inhibitor of reaction 1, or else all reactants of reaction 2 to be present with no inhibitors of reaction 2 present, and so on until reaction  $k$ . At step  $n-1$ , the aim is to find out what factors must cause all reaction 1 reactants to be simultaneously present and all reaction 1 inhibitors to be absent, and the same applies to all reactions up to  $k$  considered for the current step. In particular, the absence of an inhibitor at a certain step will depend on the simultaneous non-applicability of all reactions that can produce it: consequently, for each reaction that can produce it, either one or more reactants will be missing, or there will be at least one instance that inhibits it. It is clear that when faced with a complex reaction system, even on calculating a predictor with a small  $n$ , there would be a state space explosion and consequently bringing a huge propositional formula into DNF would become time-consuming.

In this thesis, the focus is on the application of concepts typical of abstraction techniques to make computationally tractable predictor computation

even in cases of reaction systems consisting of resources that don't meet conditions described in Theorem 2.31.

## Chapter 3

# Over-Approximation

First let us discuss why a form of approximation can be useful in this context. The exact computation of an ancestor, and more in general of a formula based predictor can be computationally very costly, as anticipated in Section 2.3.4. The complexity of its calculation depends on the number of reactions that have  $s$  as product and in particular on the number of reactants and inhibitors participating in each reaction. Moreover, such complexity increases as the number of steps increases.

The result is that the complexity of the backward process that results from formula based predictor computation can burst.

In this regard, a possible approximation was studied in order to capture the dynamic behavior of the system. Specifically, we defined an over-approximation that outputs a superset of possible sets of initial reactants and inhibitor that could lead to obtaining  $s$  after  $n$  steps.

**Preliminaries.** The goal of the proposed approximation is to compute only the initial set of objects that leads to the production of  $s$  in  $n$  steps. So it is correct to say that the solution described below is designed to work in closed reaction systems, that is, those reaction systems in which only a starting set  $D_0$  is specified, after which the evolution will be described solely by the sequence of results  $\delta$ , without introducing anything external during the developmental steps. Formally, then:

$$\delta = D_0, D_1, \dots, D_n, \text{ where } D_i = \text{res}_{\mathcal{A}}(D_{i-1}) \text{ for all } 0 < i < n$$

So to be more precise, we aim to finding an approximation of the ancestor, even though the term predictor is used in the text.

### 3.1 Set of sets of literals over-approximation

The starting point for designing our over-approximation, according to the abstract interpretation framework, is the definition of the concrete domain. Specifically, the concrete domain is represented by a collection of sets of positive and negative literals  $\wp(\wp(\mathcal{L} \cup \neg\mathcal{L}))$ . This collection can synthesize all possible causes for which a certain effect on computation was obtained. To simplify the reading, from here on, let us denote by  $\mathcal{Q}$  the set  $\mathcal{L} \cup \neg\mathcal{L}$ .

Approximation aims at representing set of sets of literals with a pairs of sets of literals  $\wp(\mathcal{Q}) \times \wp(\mathcal{Q})$ . Precisely, given a set of sets of literals, the first set of the pair will contain all the literals common to all sets and the second those that appear in at least one set. We will refer to these sets as must set and maybe set respectively.

**EXAMPLE 3.1:** For example,  $K = \{\{a, b\}, \{a, c, \neg b\}\}$  can represent the formula  $(a \wedge b) \vee (a \wedge c \wedge \neg b)$  for explaining the causes of some given entity. We use the set  $\text{must}(K) = \{a\}$  and  $\text{maybe}(K) = \{c, b, \neg b\}$  to approximate the set  $K$ .  $\blacklozenge$

We now define the abstraction function on the set of sets literals.

**Definition 3.1** (Set of sets of literals abstraction function). Given the set of all positive and negative literals, indicated as  $\mathcal{Q} = \mathcal{L} \cup \neg\mathcal{L}$ , the abstraction function  $\alpha : \wp(\wp(\mathcal{Q})) \rightarrow \wp(\mathcal{Q}) \times \wp(\mathcal{Q})$  is defined as:

$$\alpha(K) = (\text{must}(K), \text{maybe}(K))$$

where  $\text{must} : \wp(\wp(\mathcal{Q})) \rightarrow \wp(\mathcal{Q})$  and  $\text{maybe} : \wp(\wp(\mathcal{Q})) \rightarrow \wp(\mathcal{Q})$  are two auxiliary functions such that:

$$\begin{aligned} \wp(\wp(\mathcal{Q})) \ni K &\xrightarrow{\text{must}} \bigcap_{K \in \mathcal{K}} K \\ \wp(\wp(\mathcal{Q})) \ni K &\xrightarrow{\text{maybe}} \bigcup_{K \in \mathcal{K}} K \setminus \text{must}(K) \end{aligned}$$

The concretization function from abstract to concrete domain will give a set of sets, where each set contains the must literals joined each set of the power set of literals appearing in the maybe.

**Definition 3.2** (Set of sets of literals concretization function). Given the set of all positive and negative literals, indicated as  $\mathcal{Q} = \mathcal{L} \cup \neg\mathcal{L}$ , the concretization function  $\gamma : \wp(\mathcal{Q}) \times \wp(\mathcal{Q}) \rightarrow \wp(\wp(\mathcal{Q}))$  is defined as:

$$\gamma(\text{MU}, \text{MA}) = \{K \mid \text{MU} \subseteq K \wedge K \setminus \text{MU} \in \wp(\text{MA})\}$$



EXAMPLE 3.2: Suppose to have the following set of sets of literals:

$$K = \{\{a, b\}, \{a, c, \neg b\}\}$$

This can be for example a convenient way to see a DNF formula: a set of sets of literals where two literals are in the same set when they are in conjunction in the formula.

We can approximate this set by applying the  $\alpha$  defined in Definition 3.1.

$$\alpha(K) = (\{a\}, \{b, c, \neg b\})$$

However, when one switches to concretizing it, not only the original set is found, but also something more.

$$\gamma(\{a\}, \{b, \neg b, c\}) : \begin{array}{cc} \{a\} & \{a, b\} \\ \{a, \neg b\} & \{a, c\} \\ \{a, \neg b, c\} & \{a, b, c\} \end{array}$$

◆

The example above confirms that the defined approximation is indeed an over-approximation, since when an element of the abstract set is concretised, a super-set of the exact element of the concrete set is obtained.

For the concrete set, we can define a concrete operation  $pred : \wp(\wp(\mathcal{Q})) \rightarrow \wp(\wp(\mathcal{Q}))$  that, given an initial set of literals sets, returns a set of sets of literals which explain how the set given as input was obtained. The abstract counterpart of  $f$  will have to be defined in terms of the elements of the abstract domain. It will be a function  $\overline{pred} : \wp(\mathcal{Q}) \times \wp(\mathcal{Q}) \rightarrow \wp(\mathcal{Q}) \times \wp(\mathcal{Q})$  defined in terms of the must and maybe sets.

EXAMPLE 3.3: Let's picture in the Figure 3.1 all possible ways to generate a state colored in orange. To see how we could arrive to the orange state, we have to go backward from the orange state to all the possible ways to create it. This backward process can be abstracted in a process that, from an abstract state containing the orange state in the must set, gives a representation of all the ways we could obtain it. For building the resulting abstract state we need to have a state containing two sets: the first will contain the coloured balls that are common to all states, the second the balls that appear in at least one state of the ancestors of the orange state. ◆

For compactness, in addition to the *must* and *maybe* operators, previously introduced in the Definition 3.1, a third operator will be used in the

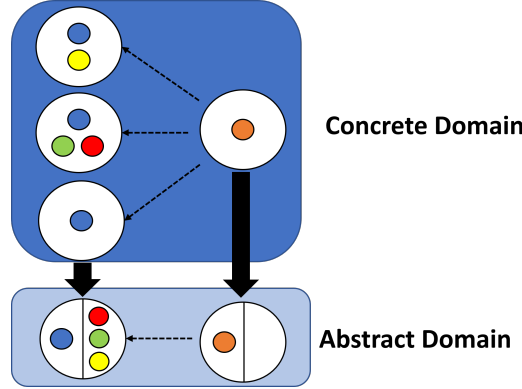


Figure 3.1: Abstract operation for computing set of sets of literals modeling colours state

next definition, referred to as this:

$$all(\mathcal{K}) = maybe(\mathcal{K}) \cup must(\mathcal{K}), \text{ with } \mathcal{K} \in \wp(\wp(\mathcal{Q}))$$

We define  $\overline{pred}$  as follow:

**Definition 3.3** (set of sets of literals abstract operator). Given the abstract domain  $\wp(\mathcal{Q}) \times \wp(\mathcal{Q})$ , the abstract operator  $\overline{pred} : \wp(\mathcal{Q}) \times \wp(\mathcal{Q}) \rightarrow \wp(\mathcal{Q}) \times \wp(\mathcal{Q})$  is defined as follows:

$$\overline{pred}(MU, MA) = \left( \bigcap_{mu \in MU} must(\mathcal{K}_{mu}), \left( \bigcup_{\substack{ma \in MA \\ mu \in MU}} all(\mathcal{K}_{ma}) \cup all(\mathcal{K}_{mu}) \right) \setminus \bigcap_{mu \in MU} must(\mathcal{K}_{mu}) \right)$$

Where  $\mathcal{K}_a \in \wp(\wp(\mathcal{Q}))$  is a set of sets of literals that represent ancestors of literal  $a$  and that will be formally defined in the next section (see Definition 3.4).

## 3.2 Predictor abstraction

As mentioned in Section 3.1, we can attribute to each set in  $\wp(\wp(\mathcal{Q}))$  the meaning of all possible causes for the production of a entity in a reaction system. In the context of reaction systems, the literals  $\ell$  in  $\wp(\wp(\mathcal{Q}))$  will mean the presence or absence (according to when positive and when negative) of the homonymous entity they represent.

Thus, we can use the over-approximation defined in Section 3.1 for classifying the presence or absence of the set of instances as being strictly or possibly necessary for obtaining a product.

Given a reaction system  $\mathcal{A} = (S, A)$ , the pair of propositional symbol sets returned by the  $\alpha$  defined in Definition 3.1 will have the following meaning:

- **MUST set:** Set of objects necessary to ensure the presence/absence of an object  $p \in S$ .
- **MAYBE set:** Set of objects whose requirement to ensure the making of  $p \in S$  is doubted.

EXAMPLE 3.4: Suppose to have a reaction system  $\mathcal{E} = (\{A, \dots, F\}, \{a_1, \dots, a_6\})$ , with the following reactions:

$$\begin{aligned} a_1 &= (\{A, B\}, \{F\}, \{C\}) & a_2 &= (\{A, B, C\}, \emptyset, \{C\}) \\ a_3 &= (\{A, B, C, D\}, \emptyset, \{A, E\}) & a_4 &= (\{E\}, \{A\}, \{F\}) \\ a_5 &= (\{A\}, \emptyset, \{B\}) & a_6 &= (\{A, C, F\}, \emptyset, \{A\}) \end{aligned}$$

Suppose that one want to approximate causalities for producing instance  $C$ . The concrete object to approximate will be:

$$cause(C) = (A \wedge B \wedge \neg F) \vee (A \wedge B \wedge C)$$

It is represented as  $\{\{A, B, \neg F\}, \{A, B, C\}\}$ . Its approximation will be:

$$\begin{aligned} must(cause(C)) &= \{A, B, \neg F\} \cap \{A, B, C\} \\ &= \{A, B\} \end{aligned}$$

$$\begin{aligned} maybe(cause(C)) &= (\{A, B, \neg F\} \cup \{A, B, C\}) \setminus \{A, B\} \\ &= \{A, B, C, \neg F\} \setminus \{A, B\} \\ &= \{C, \neg F\} \end{aligned}$$

$$\begin{aligned} \alpha(cause(C)) &= (must(cause(C)), maybe(cause(C))) \\ &= (\{A, B\}, \{C, \neg F\}) \end{aligned}$$

This also reflects what can be observed: Production of  $C$  depends on the applicability of reactions  $a_1$  and  $a_2$ . By observing them, one can conclude that:

- it is necessary to have  $A$  and  $B$  in both cases
- however, whether at least one reaction is successful may depend on either the absence of  $F$ , or the presence of  $C$

Therefore, the presence of  $C$  can be said to rely on the (certain) presence of  $A$  and  $B$  and possibly on the presence of  $C$  and the absence of  $F$ .

It is worth to notice that  $\alpha$  works also to compute causes for non-production of  $C$ . If we define  $nocause(lit) = \neg cause(lit)$ , we obtain:

$$\begin{aligned}
nocauses(C) &= \neg((A \wedge B \wedge \neg F) \vee (A \wedge B \wedge C)) \\
&= (\neg A \vee \neg B \vee F) \wedge (\neg A \vee \neg B \vee \neg C) \\
&= \neg A \vee \neg B \vee (\neg A \wedge \neg B) \vee (\neg A \wedge \neg C) \vee \\
&\quad (\neg B \wedge \neg C) \vee (F \wedge \neg A) \vee (F \wedge \neg B) \vee (F \wedge \neg C) \\
&= \{\neg A\}, \{\neg B\}, \{\neg A, \neg B\}, \{\neg A, \neg C\}, \\
&\quad \{\neg B, \neg C\}, \{F, \neg A\}, \{F, \neg B\}, \{F, \neg C\}
\end{aligned}$$

$$\begin{aligned}
\alpha(nocauses(C)) &= (must(nocauses(C)), maybe(nocauses(C))) \\
&= (\{\}, \{\neg A, \neg B, \neg C, F\})
\end{aligned}$$

By concretizing the abstraction of  $cause(C)$  we obtain:

$$\begin{aligned}
\gamma(\{\{A, B\}, \{C, \neg F\}\}) &= \{A, B\} \cup \{\{\}, \{C\}, \{\neg F\}, \{C, \neg F\}\} \\
&= \{\{A, B\}, \{A, B, C\}, \{A, B, \neg F\}, \{A, B, C, \neg F\}\}
\end{aligned}$$

Note again the nature of the approximation found: by concretizing one finds that they can be included conjunctions that were not in the original causal proposition. The additional sets could still lead to the production or non-production of an object because they overspecify the original formulas contained in the exact solution. Therefore, the conclusion is that the approximation formed is an overapproximation.  $\blacklozenge$

Furthermore, all possible sets of predictors can be seen as an iterative process of calculating the possible causes for obtaining a particular product in a certain number of steps. Assuming to be at step  $n$ , when coming back to step  $n-1$ , there may be various causes that lead the instances to be produced in  $n$  steps, and for each of these causes there can be many reasons that in step  $n-1$  caused them to occur, and so on. By iteratively visit this production process backward, we may end up in defining a set of entities that need to be present in all ancestors, while some other can be present.

**EXAMPLE 3.5:** Suppose we have a reaction system consisting of instances  $S = \{a, b, c\}$ . Somehow, an ancestor was derived that has the following formula:

$$Anc(a, 2) : \{\{a, b\}, \{a, \neg b, c\}\}$$

With marking all traces derived from all initial contexts that satisfy this formula, one could obtain a trace tree like the one in Figure 3.2.

The formula expressed by  $Anc(a, 2)$  can be summarized by two facts that must happen to the initial context:

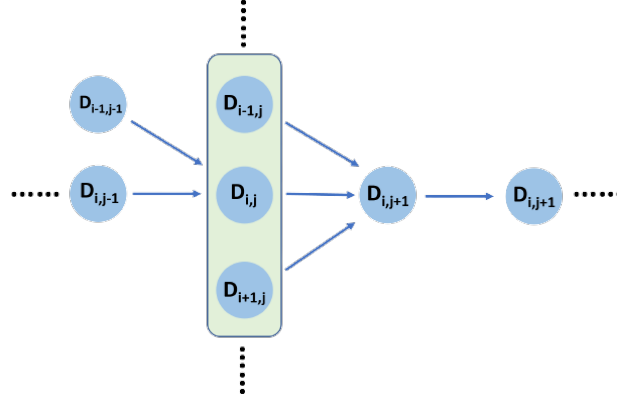


Figure 3.2: Predictor abstraction operation graphically

- $a$  must be present
- $b$  and  $c$  may be present,  $b$  may be absent

Translated with must and maybe sets, the approximate version of the formula becomes:

$$\text{MUST}(\{a\}), \text{MAYBE}(\{b, \neg b, c\})$$

However, when one switches to concretizing it into a formula in a way similar to Example 3.4, not only the original formula is obtained, but also several other sets.

$$\text{MUST}(\{a\}), \text{MAYBE}(\{b, \neg b, c\}) : \left\{ \begin{array}{ll} \{a\} & \{a, b\} \\ \{a, \neg b\} & \{a, c\} \\ \{a, \neg b, c\} & \{a, b, c\} \end{array} \right\}$$

The other set are either superset of the formulas in Anc, or formulas that do not generate  $a$  in 2 steps. ♦

Intuitively, the approximate predictor can be conceived as a flattening of the possible configurations with which we can start a computation leading to  $s$  in  $n + 1$ -steps long leading to  $s$ , exactly as showed in Figure 3.2.

A Must/Maybe based predictor (or MuMa predictor) is modeled as a set pair (MU, MA), composed by two subsets of atomic literals, containing respectively the objects that must be necessarily included/excluded and the objects that can be optionally included/excluded for leading to the production of  $s$  after  $n$  steps.

Let us define an operator MuMa that allows the maybe/must set based predictor to effectively be derived. The operator is seen as a recursive ver-

sion of the abstract operator defined for a generic set of sets of literals in Definition 3.3.

**Definition 3.4** (MuMa operator). Let  $\mathcal{A} = (S, A)$  be a reaction system and  $s \in S$ . It is defined as Must/Maybe set predictor operator the function  $\text{MuMa} : S \times \mathbb{N} \rightarrow \mathcal{Q} \times \mathcal{Q}$  as follows:

$$\text{MuMa}(s, n) = \text{MuMas}(\{s\}, \{\}, n) \text{ [or } \text{MuMas}^n(\{s\}, \{\})]$$

Where  $\text{MuMas} : \mathcal{Q} \times \mathcal{Q} \times \mathbb{N} \rightarrow \wp(\mathcal{Q}) \times \wp(\mathcal{Q})$  is an auxiliary iterative function defined as follows:

$$\text{MuMas}^n(MU, MA) = \begin{cases} \text{MuMas}^{n-1} \left( \bigcap_{mu \in MU} \text{must}(\mathcal{K}_{mu}), \right. & \text{if } n > 1 \\ \left. \left( \bigcup_{\substack{ma \in MA \\ mu \in MU}} \text{all}(\mathcal{K}_{ma}) \cup \text{all}(\mathcal{K}_{mu}) \right) \setminus \bigcap_{mu \in MU} \text{must}(\mathcal{K}_{mu}) \right) \\ (MU, MA) & \text{if } n = 1 \end{cases}$$

Where  $\mathcal{K}_a \in \wp(\wp(\mathcal{Q}))$  is  $\text{cause}(a)$  if  $a$  is a positive literal,  $\text{nocause}(a)$  otherwise.

**EXAMPLE 3.6:** Let  $\mathcal{A} = \{A, \dots, F\}$  be the reaction system of Example 3.4 with the following reactions:

$$\begin{aligned} a_1 &= (\{A, B\}, \{F\}, \{C\}) & a_2 &= (\{A, B, C\}, \emptyset, \{C, D\}) \\ a_3 &= (\{A, B, C, D\}, \emptyset, \{A\}) & a_4 &= (\{E\}, \{A\}, \{F\}) \\ a_5 &= (\{A\}, \emptyset, \{B, E\}) & a_6 &= (\{A, C, F\}, \emptyset, \{A\}) \end{aligned}$$

We first compute the approximation of the causes of each literal, by considering the abstraction of the  $\text{cause}(\mathcal{L})$ , that is  $\alpha(\text{cause}(\mathcal{L}))$ . For shortness, instead of writing  $\alpha(\text{cause}(\text{lit}))$  or  $\alpha(\text{nocause}(\text{lit}))$ , we write directly  $\alpha(\text{lit})$  or  $\alpha(\neg(\text{lit}))$ .

$$\begin{aligned} \alpha(A) &= \{A, C\}, \{B, D, F\} & \alpha(\neg A) &= \{\}, \{\neg A, \neg B, \neg C, \neg D, \neg F\} \\ \alpha(B) &= \{A\}, \{\} & \alpha(\neg B) &= \{\neg A\}, \{\} \\ \alpha(C) &= \{A, B\}, \{C, \neg F\} & \alpha(\neg C) &= \{\}, \{\neg A, \neg B, \neg C, F\} \\ \alpha(D) &= \{A, B, C\}, \{\} & \alpha(\neg D) &= \{\}, \{\neg A, \neg B, \neg C\} \\ \alpha(E) &= \{A\}, \{\} & \alpha(\neg E) &= \{\neg A\}, \{\} \\ \alpha(F) &= \{E, \neg A\}, \{\} & \alpha(\neg F) &= \{\}, \{\neg E, A\} \end{aligned}$$

Suppose you want here infer notion of the possible sets that can lead you to the production of  $C$  in 4 steps. We compute  $\text{MuMa}(C, 4)$ . Negative literals will be abbreviated with a bar above the corresponding literal.

$$\text{MuMa}(C, 4) = \text{MuMas}^4(\{C\}, \{\})$$

$$\begin{aligned}
&= \text{MuMas}^3(\{A, B\}, \{C, \bar{F}\}) \\
&= \text{MuMas}^2\left(\{A, C\} \cap \{A\}, \right. \\
&\quad \left. (\{B, D, F\} \cup \{A, C\} \cup \{A\} \cup \{A, B\} \cup \{C, \bar{F}\}, \{\bar{E}, A\}) \setminus \{A\}\right) \\
&= \text{MuMas}^2(\{A\}, \{B, C, D, F, \bar{E}, \bar{F}\}) \\
&= \text{MuMas}^1\left(\{A, C\}, \right. \\
&\quad \left. (\{A, C\} \cup \{B, D, F\} \cup \{A\} \cup \{A, B\} \cup \{C, \bar{F}\} \cup \{A, B, C\} \cup \{E, \bar{A}\} \cup \{\bar{A}\} \cup \{\bar{E}, A\}) \setminus \{A, C\}\right) \\
&= \text{MuMas}^1(\{A, C\}, \{B, D, E, F, \bar{A}, \bar{E}, \bar{F}\}) \\
&= (\{A, C\}, \{B, D, E, F, \bar{A}, \bar{E}, \bar{F}\})
\end{aligned}$$

If the exact solution, that is, the ancestor of  $C$  in 4 steps, were computed, the following formula would be obtained:

$$\text{Anc}(C, 4) = \{\{A, B, C, D\}, \{A, B, C, F\}\}$$

Observe that if you concretize  $\text{MuMa}(C, 4)$ , you would obtain an overapproximation of  $\text{Anc}(C, 4)$ , namely:

$$\text{Anc}(C, 4) \subseteq \gamma(\text{MuMa}(C, 4))$$

♦

### 3.3 MuMa predictor concretization

After MuMa predictor computation we computed the reverse transition from abstract domain to concrete domain must take place.

This will correspond to applying the concretization function  $\gamma$ , delineated in Definition 3.2, to the maybe/must sets obtained through  $\text{MuMa}(s, n)$ .

$$\mathcal{F} = \gamma(\text{MuMa}(s, n)) = \{F_1, \dots, F_n\}$$

The overall result can be interpreted as a boolean formulae set. Each formula  $F_i$  will be obtained by considering all literals belonging to the same set in conjunction, while the formulae obtained from different set have to be considered in disjunction.

It is possible to simplify  $\mathcal{F}$  by removing all the formulas  $F$  such that:

$$\mathcal{F} = \{F \in \gamma(\text{MuMa}(s, n)) \mid \nexists s \in S \text{ s. t. } \{s, \neg s\} \subseteq F\}$$

The explanation for this choice is evident: if the  $F \in \mathcal{F}$  represents a conjunction of literals, a literal for the same instance  $s \in S$  that is both positive and negative means specifying in the formula that  $s$  is both present and absent is contradictory, consequently all  $F$  of this type should be rejected regardless.

### 3.4 Formulae verification

The concrete set  $\mathcal{F}$  is the result of an above approximate computation. This means that not all  $F_i$  elements could represent correct directives formulas expressing set leading to  $s$  after  $n$  steps. Therefore, it is crucial to build a verification process that filters out all formulas that cannot lead to the production on the desired entity.

Given a boolean formula  $F \in \mathcal{F}$ , it is required to verify, for all possible initial contexts  $D_0$  satisfying  $F$ , whether it produce  $s$  after  $n$  steps or not. By following the predictor's definition of satisfiability 2.15, easily adaptable to the case of ancestor, we need to derive the initial sets. Assumed the formula is  $F = \bigcup_{\ell=s \text{ or } \ell=\neg s, s \in S} \ell$ , every possible  $D_0$  will have to be equipped with these fixed attributes:

- If  $\ell$  is positive, then insert it in  $D_0$
- If  $\ell$  is negative, i.e.  $\neg s$ , then omit it in  $D_0$

From these instructions, it can be noted the choice that the choice we made is: every not mentioned element in  $D_0$  and in general in every other state  $D_i$  will be translated into the total absence of entity. Another possible choice could have been to interpret the omission of an entity in  $D_0$  as a under specification, meaning either the absence or presence.

The set  $D_0$  constructed from  $F$  following the previous guidelines will be called the minimal set and will be denoted from here on by  $\mathcal{M}(F)$ . Intuitively, this is the minimum configuration that satisfies  $F$ .

**Definition 3.5** (Minimal configuration). Given a formula  $F = \bigcup \ell$ , the minimal configuration  $\mathcal{M}(F)$  is  $\mathcal{M}(F) = \bigcup_{\ell \text{ s.t. } \ell=s \in S} \ell$ .

**EXAMPLE 3.7:** Take the previous Example 3.6 and consider  $\text{MuMa}(C, 4)$ , which result is:

$$\text{MuMa}(C, 4) = \{\{A, C\}, \{B, D, E, F, \bar{A}, \bar{E}, \bar{F}\}\}$$

One possible formula  $F$  belonging to its concretisation can be  $\{A, C, \bar{F}\} \in \gamma(\text{MuMa}(C, 4))$ . Then the minimal configuration satisfying  $F$  will be  $\mathcal{M}(F) = \{A, C\}$ .



To form all other initial sets, one could add to the minimal configuration  $\mathcal{M}(F)$  all elements belonging to the powerset of all entities not appearing in  $F$ . The reason lies again in the general choice of considering missing elements in states as absent, either because they were not initially present or because of inapplicability of the rules that produce them. When an instance is not present in the current formula, it means that its presence or absence is not specified. That is as if for the formula it is irrelevant to specify what happens to this object. So one will have to proceed with checking the correctness of the formula by specifying both an initial context in which that particular object is not there and an initial context in which it is absent. We define an operation of joining a set with each set in a set. This will be indicated by  $\cup$ .

$$A \cup \{B_1, \dots, B_n\} = \{B_1 \cup A, \dots, B_n \cup A\}$$

We can now define how to generate all possible configurations that satisfy a formula.

**Definition 3.6.** Given a reaction system  $\mathcal{A} = (S, A)$  and a formula  $F \in \mathcal{F}$ , the set of all initial states  $\mathcal{D}_0^F$  satisfying  $F$  will be defined as follows:

$$\mathcal{D}_0^F = \mathcal{M}(F) \cup \mathcal{P}(\mathbb{A})$$

With  $\mathbb{A} = \{s \in S \mid \nexists \ell \in F \text{ s.t. } \ell = s \text{ or } \ell = \neg s\}$

**EXAMPLE 3.8:** From the previous Example 3.5, it has been computed  $\text{MuMa}(C, 4)$ .

$$\text{MuMa}(C, 4) = (\{A, C\}, \{B, D, E, F, \bar{A}, \bar{E}, \bar{F}\})$$

Assume that  $\mathcal{F} = \gamma(\text{MuMa}(C, 4))$  and take one formula from  $\mathcal{F}$ , for example  $F_1 = \{A, B, C, D\}$ . Considering that  $S = \{A, \dots, F\}$ , all possible initial contexts satisfying  $F_1$  will be:

$$\mathcal{D}_0^{F_1} = \{\{A, B, C, D\}, \{A, B, C, D, E\}, \{A, B, C, D, F\}, \{A, B, C, D, E, F\}\}$$

By taking another formula from  $\mathcal{F}$ , for example  $F_2 = \{A, B, C, \neg F\}$ , all initial contexts satisfying  $F_2$  will be:

$$\mathcal{D}_0^{F_2} = \{\{A, B, C\}, \{A, B, C, D\}, \{A, B, C, D, E\}\}$$

Observe that  $\mathcal{M}(F_1) = \{A, B, C, D\}$ , while  $\mathcal{M}(F_2) = \{A, B, C\}$ . ♦

Once obtained  $\mathcal{D}_0^F$  for a given formula  $F$ , we need to check if after  $n + 1$  steps we obtain  $S$ .

Let a dashed arrow with a number  $n$  at its apex  $A \dashrightarrow^n B$  indicate the  $n$ -long trace starting from  $A$  and ending to  $B$ .

**Definition 3.7.** Given a reaction system  $\mathcal{A} = (S, A)$  and  $\mathcal{F} = \gamma(\text{MuMa}(s, n))$ , consider a formula  $F \in \mathcal{F}$  and indicate with  $\mathcal{D}_0^F = \bigcup D_0^F$  all initial contexts satisfying  $F$ . If for all  $D_0^F \in \mathcal{D}_0^F$  we have that:

$$\mathcal{D}_0^F \dashrightarrow^{n+1} D_{n+1} \text{ such that } s \in D_{n+1}$$

Then  $F$  is said to be verified.

**EXAMPLE 3.9:** Take the formula  $F_1$  from the previous Example 3.8, whose contexts satisfying it were:

$$\mathcal{D}_0^{F_1} = \{\{A, B, C, D\}, \{A, B, C, D, E\}, \{A, B, C, D, F\}, \{A, B, C, D, E, F\}\}$$

Starting from each set in  $\mathcal{D}_0^{F_1}$ , generate the 5-long trace and see if there is  $C$  in the fifth state.

$$\begin{aligned} \{A, B, C, D\} &\rightarrow \{A, B, C, D, E\} \dashrightarrow^4 \{A, B, C, D, E\} \\ \{A, B, C, D, E\} &\dashrightarrow^5 \{A, B, C, D, E\} \\ \{A, B, C, D, F\} &\rightarrow \{A, B, C, D, E\} \dashrightarrow^4 \{A, B, C, D, E\} \\ \{A, B, C, D, E, F\} &\rightarrow \{A, B, C, D, E\} \dashrightarrow^4 \{A, B, C, D, E\} \end{aligned}$$

Since  $C$  is present in all fifth states,  $F_1$  is verified.

Now take the formula  $F_2$ , whose contexts satisfying it were:

$$\mathcal{D}_0^{F_2} = \{\{A, B, C\}, \{A, B, C, D\}, \{A, B, C, D, E\}\}$$

It has already been found that the second and third set leads to  $C$  after 5 steps. It remains to check the first one.

$$\{A, B, C\} \rightarrow \{B, C, D\} \rightarrow \{\}$$

Having arrived at an empty state, it is not possible to go further, so none of the rules of the system are applicable anymore. It goes without saying that  $C$  will not be obtained after 5 steps. That renders the formula  $F_2$  not verified.  $\blacklozenge$

In the example, the reader can see a typical behavior during the verification step. Given two formulas  $F_1$  and  $F_2$ , the set of contexts of a formula  $\mathcal{D}_0^{F_1}$  contains some elements of the set of contexts  $\mathcal{D}_0^{F_2}$ , since formulas are such that  $F_1 \cap F_2 \neq \emptyset$ .

The case of inclusion, that is,  $\mathcal{D}_0^{F_2} \subset \mathcal{D}_0^{F_1}$ , correspond to  $F_2 \supset F_1$ . In this particular circumstance, it would be convenient to try to verify first the less specific formula  $F_1$ , since if it is verified it wouldn't be necessary to verify also  $F_2$ .

**Proposition 3.8.** Given two formulas  $F_1, F_2 \in \mathcal{F}$  such that  $F_2 \supset F_1$ , then:

$$F_1 \text{ verified} \implies F_2 \text{ verified}$$

*Proof.* If  $F_2 \supset F_1$ , it means that  $\mathcal{D}_0^{F_2} \subset \mathcal{D}_0^{F_1}$ . If it were not true, then there would exist an initial context  $c$ , belonging to  $\mathcal{D}_0^{F_2}$ , but not to  $\mathcal{D}_0^{F_1}$ . This means that formula  $F_2$  will specify absence or presence of another objects different from all elements in  $F_1$ . In other words, it is inferred that  $F_2 \not\supset F_1$ , that is a contradiction.

If  $F_1$  is verified, from Definition 3.7 all sets in  $\mathcal{D}_0^{F_2}$  lead to  $s$  after  $n$  steps. Since the fact that  $\mathcal{D}_0^{F_2} \subset \mathcal{D}_0^{F_1}$ , it includes already all the initial contexts of  $F_2$ , and for all of them the validity of the track has already been checked. So  $F_2$  is verified.  $\square$

Note that the reverse implication does not hold, because if  $F_2$  has been verified,  $F_1$  being less specific will include contexts for which traces have not been checked. It may be the case that for some of these traces results  $s$  after  $n$  steps will not be produced.

### 3.4.1 Cycles and fixed points

The process of verifying a formula  $F$  builds, for each possible initial context satisfying that formula  $\mathcal{D}_0^F$ , a path that, after  $n$  steps, reaches the product  $s$  we are looking for.

By following these paths it may happen that cycles are encountered. Cycles can be divided into attractors and fixed points.

**Definition 3.9.** Given a reaction system  $\mathcal{A} = (S, A)$ , an attractor in the result sequence  $\delta$  is a cycle starting from  $h$  and having period  $k$  such that for all  $t \in \mathbb{N}$ :

$$res_{\mathcal{A}}^{h+kt}(T) = res_{\mathcal{A}}^{h+k}(T)$$

In particular, if  $k = 1$ , the resulting attractor is a fixed point.

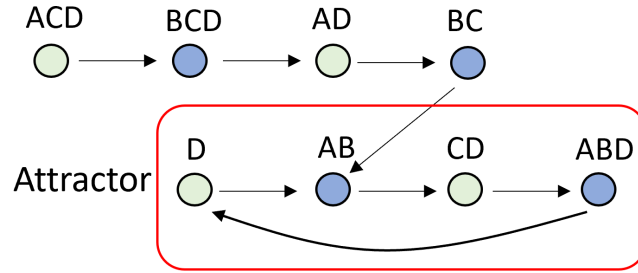
An example of what attractors and fixed points actually are can be seen below.

EXAMPLE 3.10: Given a reaction system  $\mathcal{A} = (S, A)$ , with the follow-

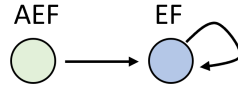
ing reactions:

$$\begin{aligned} a_1 = (\{A, B\}, \{D\}, \{C\}) \quad a_2 = (\{C\}, \{\}, \{D\}) \quad a_3 = (\{B\}, \{\}, \{D\}) \quad a_4 = (\{E\}, \{\}, \{F\}) \\ a_5 = (\{A, D\}, \{B\}, \{C\}) \quad a_6 = (\{D\}, \{B\}, \{B\}) \quad a_7 = (\{D\}, \{A\}, \{A\}) \quad a_8 = (\{F\}, \{\}, \{E\}) \end{aligned}$$

The figure below is obtained by computing all traces from the states satisfying the formula  $D \wedge \neg B$  (marked in light green). After at most 4 steps there is the state  $AB$  of an attractor of period 4. Once entered in the attractor, there's no way to escape from it.



An example of a fixed point could be the trace found starting from  $AEF$  in the following image, where  $EF$  is the only state belonging to the attractor.



◆

In general, since the set of any possible result in  $D_i$  is finite, every state sequence is ultimately periodic. Of course  $n$  can be much smaller than the steps necessary to reach the attractor cycle but as  $n$  increases, it becomes very likely to run into an attractor. This is because we are dealing with closed reaction systems, so the process is deterministic and it is ensured that, once a cycle is met, it will never be left. So the identification of these particular cycles is crucial for simplifying the formula verification phase. Cycles can be exploited to speed up the process of formulae verification. When aiming to compute the  $n + 1$ -th state of a trace, assuming it arrives in a cycle after  $t$  steps and knowing the period  $k$  of the attractor, as well as the states from which it is composed, one can jump directly to the state of interest by computing:

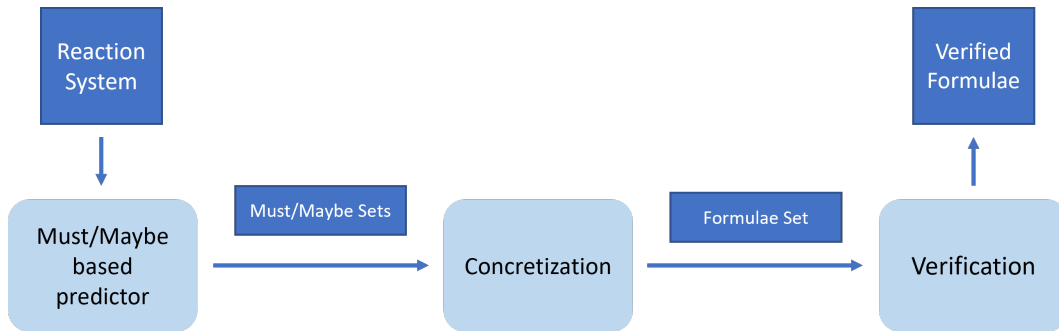
$$\text{ind}_{\text{jump}} = t + ((n + 1 - t) \bmod k)$$

With this simple mathematical formula, one avoids calculating by hand the

result of the remaining  $n + 1 - t$  steps.

### 3.5 Summarizing the process

Given a reaction system  $\mathcal{A} = (S, A)$ , the search for a predictor for  $s \in S$  in  $n$  steps basically comprises three steps:



1. The abstraction phase

It computes the Must/Maybe predictor for  $s$  in  $n$  steps through the operator  $\text{MuMa}(s, n)$ . This brings up a pair of sets  $\text{MU}, \text{MA}$ , respectively the set of objects that should be present/absent and the set of objects that will probably need to be present/absent in the initial context.

2. The concretization phase

It settles the transition from abstract domain to concrete domain, interpreting the sets  $\text{MU}, \text{MA}$  as a set of formulas  $\mathcal{F}$ .

3. The verification phase

It filters the formulas in  $\mathcal{F}$  that effectively generate initial contexts out of which traces depart that after  $n$  steps lead to  $s$ .



# Chapter 4

## Positive Reaction Systems

In this chapter, a method to translate reaction system will be introduced that allows us to completely remove the inhibitors. The idea is to have only positive resources available, by merging properly reactants and inhibitors in a unique set. Since inhibitors can be conceived as stand-alone positive instances, new reactions with those new entities as product can be derived. The result of this transformation will be a new reaction system, that we will call reaction system with no inhibitors, or "positive" reaction system. In the last part of this chapter, we will discuss advantages and disadvantages of the proposed transformation.

### 4.1 Converting a Reaction System

Suppose we have a reaction system  $\mathcal{A} = (S, A)$ , having entities  $s_1, \dots, s_n \in S$  and reactions  $a \in A$  of the form  $a = (R_a, I_a, P_a)$ .

The corresponding reaction system without inhibitors, or positive reaction system, referred to as  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$ , will have twice as many entities as the original one, i.e.,  $s_1, \dots, s_n, i_1, \dots, i_n \in S_{AwI}$ . This comes from the idea to represent absence of entities with separate instances. During this chapter, the new entities  $i_1, \dots, i_n$  will be referred to as negative instances.

First of all, we defined a (bijective) function that maps instances present in a reaction as inhibitors, into new elements  $i_k$  that are part of the set of reagents of the reaction.

$$inh(s_k) = i_k, \text{ with } inh : S \rightarrow \{i_1, \dots, i_n\}$$

We can lift the previous operator to one that works on sets in the usual way.

$$Inh(S) = \bigcup_{s \in S} inh(s)$$

Reactions in  $\mathcal{A}$ , producing elements  $s_1, \dots, s_n$ , will be transformed by moving inhibitory entities in reactants set  $S_{AwI}$ .

EXAMPLE 4.1: Suppose we have  $S = \{A, B, C\}$ , and assume the following function  $inh$  that will map entities appearing as inhibitor into the new entities  $a, b, c$ .

$$inh(A) = a, \quad inh(B) = b, \quad inh(C) = c \implies inh(S) = \{a, b, c\}$$

Then, the corresponding positive reaction system will have the set of entities  $S_{AwI} = \{A, B, C, a, b, c\}$ .  $\blacklozenge$

Each reaction  $a \in A$  having the form  $(R_a, I_a, P_a)$  will be converted into a reaction  $(R_a \cup Inh(I_a), \emptyset, P_a)$ .

Moreover,  $A_{AwI}$  will be characterized by the set of positive reactions  $A_{pos} = \{(R_a \cup Inh(I_a), \emptyset, P_a) \mid a \in A\}$  plus a new set of reactions that will characterize the new introduced entities.

EXAMPLE 4.2: Suppose to have a classic reaction system  $\mathcal{A} = (S, A)$ , with  $S = \{A, B, C, D\}$ , and  $A$  characterized by the following reactions:

$$\begin{aligned} a_1 &= (\{A, B\}, \{D\}, \{C\}) & a_2 &= (\{C\}, \{\}, \{D\}) & a_3 &= (\{B\}, \{\}, \{D\}) \\ a_4 &= (\{A, D\}, \{B\}, \{C\}) & a_5 &= (\{D\}, \{B\}, \{B\}) & a_6 &= (\{D\}, \{A\}, \{A\}) \end{aligned}$$

In this simple example (and in subsequent ones), it can be hypothesized of mapping the negative resource counterparts to their low-ercase versions.

$$inh(A) = a, \quad inh(B) = b, \quad inh(C) = c, \quad inh(D) = d$$

Then, reactions that will populate  $A_{pos}$  will be:

$$\begin{aligned} a'_1 &= (\{A, B, d\}, \{\}, \{C\}) & a'_2 &= (\{C\}, \{\}, \{D\}) & a'_3 &= (\{B\}, \{\}, \{D\}) \\ a'_4 &= (\{A, D, b\}, \{\}, \{C\}) & a'_5 &= (\{D, b\}, \{\}, \{B\}) & a'_6 &= (\{D, a\}, \{\}, \{A\}) \end{aligned}$$

$\blacklozenge$

For the production of the new entities in  $\mathcal{AwI}$ , we need to add new rules to create them. Since  $i_1, \dots, i_n$  represent entities that must not be generated, the idea is to negate the rules that produce the corresponding entities  $s_1, \dots, s_n$ . This means that all rules related with the same production of the positive instance must be enabled. That is, for every entity  $i_k$  we need to



generate as many new rules as the number of ways to disable all the rules that can produce the positive instance  $s_k$ .

For each reaction that has  $s_k$  as a product, there are two possible ways to prevent  $s_k$  from being created:

- Either an inhibitor is present
- Or a reactant is absent

In our new setting of positive reaction system the presence of an inhibitor is represented by the presence of the negative corresponding resource. Mathematically speaking, possible causes are declinated as follows. If the product  $s_k$  in question can be generated by only one reaction, let us say  $(R, \{\}, P) \in A_{pos}$ , then we need to introduce  $p$  reactions producing  $i_k$ , each one having a  $i_r = inh(i_r)$  as reactant, to compute as reactants sets. If instead  $s_k$  can be generated by more than one reactions, this corresponds, given the mapping  $inh(s_k) = i_k$ , to take all the reactant sets of reactions in  $A_{pos}$  having  $s_k$  as product, inverting the positive elements into negative ones and vice versa, and then making the Cartesian product of the resulting sets.

For convenience, a *rev* function is constructed, which takes a set of objects and converts any set positive/negative element into its negative/positive counterpart.

$$rev(R) = \bigcup_{r \in R} \hat{r}, \quad \text{where } \hat{r} = \begin{cases} inh(r) & \text{if } r \in S \\ inh^{-1}(r) & \text{if } r \in Inh(S) \end{cases}$$

Since the result of a Cartesian product  $A_1 \times A_2 \cdots \times A_l$  is the list of all possible tuples that can be formed by selecting one element from each set  $A_1, \dots, A_l$ , it is also useful to define a function  $\# : \text{list} \rightarrow \text{set}$ , which converts an ordered sequence of elements into a sequence of unordered distinct elements.

Therefore, we add to  $A_{AwI}$  the set of reactions  $A'_{neg}$  producing the negative elements, created by applying the above reasoning for every negative instance  $i_k$ , for  $k \in 1, \dots, n$ . During the construction of  $A'_{neg}$ , it is possible to obtain situations in which an instance occurs in the same reaction positively and negatively. Since in this circumstance, the rule would be logically inapplicable, we remove such rules from  $A'_{neg}$ .

$$A'_{neg} = \left\{ (T^\#, \emptyset, i_k) \mid inh(s_k) = i_k \wedge T = \prod_{\forall a \in p(A_{pos}, s_k)} rev(R_a) \wedge T^\# \text{ not contradictory} \right\}$$

where  $p(A_{pos}, s_k) = \{(R, \{\}, P) \in A_{pos} \mid s_k \in P\}$

Another thing to notice is that  $A'_{neg}$  can be simplified by forcing it to contain only dominant rules, naming that given two reactions  $(R_a, \emptyset, i_k)$  and  $(R_b, \emptyset, i_k)$

in  $A'_{neg}$  such that  $R_a \subset R_b$ , only the simpler rule  $(R_a, \emptyset, i_k)$  can be kept, because this represents among the two the minimal case in which  $i_k$  will be generated anyway. The other reaction  $(R_b, \emptyset, i_k)$  is redundant. Because of these facts, one can further enrich the characterization of  $A'_{neg}$  by defining  $A_{neg}$  in the following way:

$$A_{neg} = \left\{ (R, \emptyset, i_k) \in A'_{neg} \mid \nexists a \in A'_{neg} : R_a \subset R \wedge i_k \in P_a \right\}$$

EXAMPLE 4.3: Knowing  $A_{pos}$ , compute  $A_{neg}$  of the system from the previous Example 4.2.

For computing all rules for  $c$ , given the rules:

$$a'_1 = (\{A, B, d\}, \{\}, \{C\}) \quad a'_4 = (\{A, D, b\}, \{\}, \{C\})$$

First of all, compute all reactant sets of  $c$   $\mathcal{R}_c$  through the cartesian product of  $rev(R_{a'_1})$  and  $rev(R_{a'_4})$ .

$$rev(R_{a'_1}) = \{a, b, D\} \quad rev(R_{a'_4}) = \{a, d, B\}$$

$$\begin{aligned} \mathcal{R}_c &= \left\{ \{a\}, \{a, b\}, \{a, d\}, \{a, B\}, \{a, D\}, \{b, d\}, \{b, B\}, \{d, D\}, \{B, D\} \right\} \\ &\quad (\{b, B\}, \{d, D\} \text{ contradictory}, \{a\} \subseteq \{a, b\}, \{a, d\}, \{a, B\}, \{a, D\}) \\ &= \left\{ \{a\}, \{b, d\}, \{B, D\} \right\} \end{aligned}$$

And so,  $c$  will be generated by one of these new reactions:

$$a_7 = (\{a\}, \{\}, \{c\}) \quad a_8 = (\{b, d\}, \{\}, \{c\}) \quad a_9 = (\{B, D\}, \{\}, \{c\})$$

For  $d$ , given the rules:

$$a'_2 = (\{C\}, \{\}, \{D\}) \quad a'_3 = (\{B\}, \{\}, \{D\})$$

The following rule will be generated:

$$a_{10} = (\{c, b\}, \{\}, \{d\})$$

For  $b$ , given the (only) rule:

$$a'_5 = (\{D, b\}, \{\}, \{B\})$$

Since  $B$  has one reaction, they will be generated as many rules as the number of reactant sets outputted by  $single(R_{a'_5}) = \{d\}, \{B\}$ :

$$a_{11} = (\{d\}, \{\}, \{b\}) \quad a_{12} = (\{B\}, \{\}, \{b\})$$

In the end, for  $a$ , given the (only) rule:

$$a'_6 = (\{D, a\}, \{\}, \{A\})$$

The following rules will be generated:

$$a_{13} = (\{d\}, \{\}, \{a\}) \quad a_{14} = (\{A\}, \{\}, \{a\})$$

All together,  $A_{neg} = \{a_7, \dots, a_{14}\}$ . ♦

At this point, it can be summarized what needs to be done to convert a reaction system to a positive reaction system in a single definition.

**Definition 4.1** (Reaction System without Inhibitors, or Positive Reaction System). A reaction system  $\mathcal{A} = (S, A)$  can be transformed into a reaction system without inhibitors  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$ , with:

$$S_{AwI} = S \cup Inh(S)$$

$$A_{AwI} = A_{pos} \cup A_{neg}$$

EXAMPLE 4.4: Consider again reaction system  $\mathcal{A} = (S, A)$  of Example 4.2, with:

$$\begin{aligned} S &= \{A, B, C, D\} \\ A &= \left\{ (\{A, B\}, \{D\}, \{C\}), (\{C, \{\}, \{D\}\}), (\{B\}, \{\}, \{D\}), \right. \\ &\quad \left. (\{A, D\}, \{B\}, \{C\}), (\{D\}, \{B\}, \{B\}), (\{D\}, \{A\}, \{A\}) \right\} \end{aligned}$$

It will be transformed into a reaction system without inhibitors  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$ , with:

$$\begin{aligned} S_{AwI} &= \{A, B, C, D, a, b, c, d\} \\ A_{AwI} &= \left\{ (\{A, B, d\}, \{\}, \{C\}), (\{C, \{\}, \{D\}\}), (\{B\}, \{\}, \{D\}), (\{D, b\}, \{\}, \{B\}), \right. \\ &\quad (\{A, D, b\}, \{\}, \{C\}), (\{D, a\}, \{\}, \{A\}), (\{a\}, \{\}, \{c\}), (\{b, d\}, \{\}, \{c\}), \\ &\quad (\{B, D\}, \{\}, \{c\}), (\{A\}, \{\}, \{a\}), (\{c, b\}, \{\}, \{B\}), (\{d\}, \{\}, \{b\}), \\ &\quad \left. (\{B\}, \{\}, \{b\}), (\{d\}, \{\}, \{a\}) \right\} \end{aligned}$$
♦

## 4.2 Positive reaction system analysis

From the above examples, it can be seen that the transformation complicates the system. In fact, the number of reactions to be added can be large. Moreover, the dynamic behaviour will be in some ways more restrictive than the starting system, since the absence of the inhibitors is transformed into the presence of other new instances. This means that they are added new explicit conditions on formulas to be verified.

Despite these drawbacks, this translation makes it possible to simplify the verification phase of the predictor, keeping the predictor's approximate computation the same as in the reaction system with inhibitors.

### 4.2.1 Space complexity

By converting a classical reaction system into its corresponding one without inhibitors  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$ , there are more reactions and more entities populating the system.

Regarding  $S_{AwI}$ , under the assumption that each entity  $s \in S$  has at least one reaction that produces it, it results that the number of new entities will be exactly  $|S|$ , so  $|S_{AwI}| = 2 \cdot |S|$ .

For  $|A_{AwI}|$ , one must consider for each (positive) entity  $s \in S$  all the reactions that produce it  $p(A_{pos}, s)$ . The resources  $M$  of each reaction in  $p(A_{pos}, s)$  will have to be combined with each other using the cartesian product. To this, it must be added the cardinality of the transformed reactions of the starting system  $|A_{pos}| = |A|$ . Overall, the result will be:

$$|A_{AwI}| = |A| - k_{simp} + \sum_{s \in S} \prod_{a \in p(A_{pos}, s)} |M_a|$$

There's a  $k_{simp}$  factor derived from the fact that reactions that comes from  $A_{neg}$  can be simplified, either by removing reactions whose resources are contradictory, or by omitting redundant reactions. The more elements are in common between  $M$  sets, the more the  $k_{simpl}$  parameter weighs on  $|A_{AwI}|$ . However, this factor is strongly dependent from how are built sets  $M_a$  for  $a \in p(A_{pos}, s)$ , so it can be hard to quantify more precisely.

### 4.2.2 Predictor computation

It is assumed in this study that the entities whose predictor is to be computed are exclusively the positive ones, i.e. those objects  $s$  that belong to both  $S_{AwI}$  and  $S$ .

To obtain the Must/Maybe set based predictor on a reaction system  $\mathcal{AwI} =$

( $S_{AwI}, A_{AwI}$ ) the procedure to be followed is the same as that performed on a classical reaction system.

In particular, the abstraction and concretization stage does not change, while for the last phase, that is, the verification phase, there are some new advantages resulting from this transformation.

### Abstraction and concretization phases

In order to realize that the must/maybe predictor computation does not change, observe that the set  $A_{neg}$  of reactions that produce the negative instances  $i_k \in Inh(S)$  are transcripts of the causes for which the corresponding instance  $s_k \in S$  such that  $i_k = inh(s_k)$  is absent. In other words, it is as if  $cause(i_k) = nocause(s_k)$  and consequently the  $\alpha$  applied to them gives the same result.

The consequence of this should be that the relative concretization with  $\gamma$  of the Must/Maybe predictor will generate the same set  $\mathcal{F}$ . Although in  $AwI$  it would be permissible to have both  $s$  and its dual negative  $i = inh(s)$  in the same formula, it was decided to exclude this type of formula in order to standardise at least the abstraction and concretisation phases, since one is obliged to recognise differences in the verification phase.

It can then be seen that any formula in a classic reaction system  $F^A$  can be converted into a formula in its correspondent reaction system without inhibitor  $F^{AwI}$  by simply transforming any literals  $\ell = \neg s$  into their counterpart  $i = inh(s)$ .

EXAMPLE 4.5: Suppose to have  $F = \{A, B, \neg C, \neg D\}$  in  $\mathcal{A} = (S, A)$ , then in  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$  it will be converted into  $F = \{A, B, c, d\}$ . ♦

Let us devise, a new operator  $com : \wp(S_{AwI}) \rightarrow \wp(S)$ , which takes a set of objects  $K$  in  $\mathcal{AwI}$  and gives only objects in common with  $S$  in  $\mathcal{A} = (S, A)$ , so  $com(K) = K \cap S$ .

EXAMPLE 4.6: From the previous example, from  $F = \{A, B, c, d\}$  it is obtained  $com(F) = \{A, B\}$ . ♦

### Verification phase

When proceeding with the verification phase, since  $\mathcal{F}$  is an identical set to that obtained with the reaction system  $\mathcal{A}$ , the procedure to be followed consists of the same steps: for each formula  $F \in \mathcal{F}$  it is checked that each trace obtained from each initial context gives  $s$  after  $n$  steps. The difference to be highlighted is on the outcomes of controlling the traces of every set

$D_0 \in \mathcal{D}_0^F$  such that  $D_0 \models F$ .

Before discussing the differences between the systems, it is appropriate to focus on the similarities. Following the Definition 3.6 on the set of initial contexts satisfying a certain formula  $F \in \mathcal{F}$ , clearly in the sphere  $\mathcal{AwI}$ , where there are no formulae containing literals with the  $\neg$ , this will result in generating every possible set consisting of a fixed part  $\mathcal{M}(F)$  and a variable part given by all possible combinations of the instances not mentioned in the minimal configuration. From these sets, it was chosen to exclude those containing both  $s$  and its  $i = inh(s)$ , in order to maintain consistency with the same choice made in the concretisation phase.

A first piece of similarity compares traces that can be generated by a reaction system  $\mathcal{A}$  and its conversion into  $\mathcal{AwI}$ . Knowing that  $D = res^{\mathcal{A}}(C)$  and  $N = res^{\mathcal{AwI}}(M)$ , if  $M$  and  $C$  are such that by selecting in  $M$  only objects in common with  $S$  in  $\mathcal{A} = (S, A)$  we obtain a subset of  $C$ , and moreover in  $M$  there are no other entities that in  $C$  are inhibitors, then by selecting in the resulting set  $N$  only objects in common with  $S$ , we will have a subset of  $D$ .

**Lemma 4.2.** Given a reaction system  $\mathcal{A} = (S, A)$  and its conversion  $\mathcal{AwI} = (S_{\mathcal{AwI}}, A_{\mathcal{AwI}})$ , two entities sets  $C, D$  consisting of items  $s \in S$  and two other entities sets  $M, N$  consisting of items  $s \in A_{\mathcal{AwI}}$ , it is valid that if  $M \rightarrow^{\mathcal{AwI}} N$  and  $C \rightarrow^{\mathcal{A}} D$  then:

$$com(M) \subseteq C \wedge rev(M \setminus com(M)) \cap C = \emptyset \implies com(N) \subseteq D$$

*Proof.* Cases  $C \rightarrow^{\mathcal{A}} D$  and  $M \rightarrow^{\mathcal{AwI}} N$  can be rewritten as  $D = res(C)$  and  $N = res(M)$ . The result obtained under a system without inhibitors is the consequence of a series of reactions belonging to  $A_{\mathcal{AwI}}$  enabled. The positive instances  $s \in N$  are such that  $s \in S$  and simultaneously  $s \in S_{pos}$  and, by construction, are only generable by the rules in  $A_{pos}$ . At the same time, these instances will be the only ones maintained by the  $com$  function.

The main observation to do is the following: the common basis that has remained doing  $com(M)$  is the same reactants that are needed by  $C$  in the classical system to enable the very same equivalent reactions of  $A_{pos}$ . The positive products  $N$  in  $\mathcal{AwI}$  in the worst case are the result of one or more reactions having negative instances that were certainly present in  $M$ , but for hypothesis they are also absent in  $C$ , since  $rev(M \setminus com(M)) \cap C = \emptyset$ . So those same reactions are still enabled in the case of  $\mathcal{A}$ . Therefore products  $D$  will have in common all products retained by  $com(N)$ .  $\square$

While seemingly obvious, this lemma conceals some important consequences. First, given a trace in  $\mathcal{AwI}$ , known the initial context  $D_0^{\mathcal{AwI}}$ , beginning in  $\mathcal{A}$  a trace with an initial context  $D_0^{\mathcal{A}}$  such that  $com(D_0^{\mathcal{AwI}}) \subseteq D_0^{\mathcal{A}}$  and no inhibitors in common with  $D_0^{\mathcal{AwI}}$ , it will be obtained a trace so that all states  $D_i^{\mathcal{A}}$  will be such that  $com(D_i^{\mathcal{AwI}}) \subseteq D_i^{\mathcal{A}}$ .

**Corollary 4.3.** Given a reaction system  $\mathcal{A} = (S, A)$  and its conversion  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$ , a set  $\mathcal{F}$ , one formula  $F^A \in \mathcal{F}$  with its counterpart  $F^{AwI}$ , it is valid that:

$$\exists D_0^A \in \mathcal{D}_0^{F^A}, D_0^{AwI} \in \mathcal{D}_0^{F^{AwI}} : com(D_0^{AwI}) \subseteq D_0^A \implies \forall i > 0. com(D_i^{AwI}) \subseteq D_i^A$$

*Proof.* This claim is proved by induction on  $i \in \mathbb{N}$ .

For  $i = 1$ , take two initial contexts such that  $rev(D_0^{AwI} \setminus com(D_0^{AwI})) \cap D_0^A$ . They certainly exist, for example you can take  $\mathcal{M}(F^A)$  and  $\mathcal{M}(F^{AwI})$ . For Lemma 4.2 it is assured that  $com(D_1^{AwI}) \subseteq D_1^A$ .

For  $i > 0$ , suppose that  $com(D_{i-1}^{AwI}) \subseteq D_{i-1}^A$ . Prove that  $com(D_i^{AwI}) \subseteq D_i^A$ . Let us assume by absurdity that  $com(D_i^{AwI}) \not\subseteq D_i^A$  therefore there exists a product  $p \notin D_i^A$  and  $p \in com(D_i^{AwI})$ . If this is true,  $p$  will be a positive literal, thus generated with  $res(D_{i-1}^{AwI})$  by some reaction of  $A_{pos}$ . The reasons why the corresponding reaction was not activated in  $\mathcal{A}$  will reside solely in the fact that at least one reactant  $r$  was not present at the previous step in  $\mathcal{A}$ , but was present in  $\mathcal{AwI}$ , so it is concluded that  $com(D_{i-1}^{AwI}) \not\subseteq D_{i-1}^A$ . So it only remains to recognise that  $com(D_i^{AwI}) \subseteq D_i^A$ .  $\square$

Another important consequence is the following. When a formula taken from the concretisation of the same predictor is verified in  $\mathcal{AwI}$ , it is also verified in  $\mathcal{A}$ .

**Corollary 4.4.** Given a system  $\mathcal{A} = (S, A)$ , a corresponding system  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$ , an instance  $s \in S$  for which it is computed  $\text{MuMa}(s, n)$  and a formula  $F \in \mathcal{F}$ :

$$F \text{ verified in } \mathcal{AwI} \implies F \text{ verified in } \mathcal{A}$$

*Proof.* Assuming the same formula  $F$  is always referred to, let it be omitted for readability as a superscript on all sets symbolizing states.

When  $F$  is verified in  $\mathcal{AwI}$ , it means that:

$$\forall D_0^{AwI} \models F^{AwI}. \quad D_0^{AwI} \xrightarrow{n+1} D_{n+1}^{AwI} \ni s \in S_{AwI}$$

From the previous results at Corollary 4.3, it is known that for the same formula  $F^A$  in  $\mathcal{A}$  and corresponding formula  $F^{AwI}$  in  $\mathcal{AwI}$ , it can be found one initial context  $D_0^{AwI} \in \mathcal{D}_0^{F^{AwI}}$  such that  $com(D_0^{AwI}) \subseteq D_0^A$  implies  $com(D_i^{AwI}) \subseteq D_i^A$  for all  $i$ . Since one is always looking for an element  $s \in S$ , regardless of the view of the system, it is assured that  $s \in com(D_{n+1}^{AwI})$  and consequently will also belong to  $D_{n+1}^A$ .

All that remains is to show that all possible initial contexts  $D_0^A$  satisfying  $F^A$  always find among the contexts satisfying  $F^{AwI}$  an element  $D_0^{AwI}$  such

that  $\text{com}(D_0^{\mathcal{AwI}}) = D_0^{\mathcal{A}}$ . Noting that:

$$\mathcal{D}_0^{\mathcal{AwI}} = \mathcal{M}(F^{\mathcal{AwI}}) \cup \mathcal{P}(\mathcal{A}^{\mathcal{AwI}})$$

Intuitively it is clear that  $\text{com}(\mathcal{M}(F^{\mathcal{AwI}})) = \mathcal{M}(F^{\mathcal{A}})$  and  $\mathcal{P}(\text{com}(\mathcal{A}^{\mathcal{AwI}})) \supseteq \mathcal{P}(\mathcal{A}^{\mathcal{A}})$ , so all the elements  $D_0^{\mathcal{AwI}}$  transformed by  $\text{com}$  will cover all elements in  $\mathcal{D}_0^{\mathcal{A}}$ .  $\square$

Thanks to the previous proposition, there can never be a formula  $F \in \mathcal{F}$  that is true in  $\mathcal{AwI}$  and false in  $\mathcal{A}$ .

At the same time, however, the opposite implication is certainly not proven. Formally, given a reaction system  $\mathcal{A} = (S, A)$ , whose positive reaction system will be  $\mathcal{AwI} = (S_{\mathcal{AwI}}, A_{\mathcal{AwI}})$ , assuming that a formula  $F_i \in \mathcal{F}$  is occurring, where  $\mathcal{F} = \gamma(\text{MuMa}(s, n))$ , it is possible that  $F \in \mathcal{F}$  in the system with inhibitors is verified, while in the system without inhibitors the opposite happens. The motivation is to be found in the inspection of the traces generated by the  $D_0^{\mathcal{AwI}} \models F$  contexts: not all traces lead to  $s$  after  $n$  steps, as opposed to the case under  $\mathcal{A}$ .

Traces that do not result in  $s$ , starting from a certain  $D_0$ , do not produce the instances that are needed over the course of the various  $D_i$  states to achieve the same production that we would get if it were with  $\mathcal{A}$ . The lack of needed instances is surely attributable to inhibitors. Assuming that in order to produce something necessary at step  $k$  one needs a reaction (speculates the only one in the whole system) involving some  $s \in S$  as an inhibitor:

- in the case of  $\mathcal{A}$ , a context in which  $s$  is missing correctly results in the applicability of that rule
- on the other hand, with  $A_{\mathcal{AwI}}$ ,  $s$  becomes a new reactant  $i = \text{inh}(s)$ , whose presence is necessary, so a context in which there is no  $i$  results in the non-applicability of the rule and thus in the impossibility of moving forward

EXAMPLE 4.7: Take again the previous Example 4.4. From a reaction system  $\mathcal{A} = (S, A)$  done so:

$$\begin{aligned} S &= \{A, B, C, D\} \\ A &= \left\{ (\{A, B\}, \{D\}, \{C\}), (\{C, \{\}, \{D\}\}), (\{B\}, \{\}, \{D\}), \right. \\ &\quad \left. (\{A, D\}, \{B\}, \{C\}), (\{D\}, \{B\}, \{B\}), (\{D\}, \{A\}, \{A\}) \right\} \end{aligned}$$



We get  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$  represented by:

$$S_{AwI} = \{A, B, C, D, a, b, c, d\}$$

$$A_{AwI} = \left\{ (\{A, B, d\}, \{\}, \{C\}), (\{C, \{\}, \{D\}\}), (\{B, \{\}, \{D\}\}), (\{D, b\}, \{\}, \{B\}), \right. \\ (\{A, D, b\}, \{\}, \{C\}), (\{D, a\}, \{\}, \{A\}), (\{a\}, \{\}, \{c\}), (\{b, d\}, \{\}, \{c\}), \\ (\{B, D\}, \{\}, \{c\}), (\{A\}, \{\}, \{a\}), (\{c, b\}, \{\}, \{B\}), (\{d\}, \{\}, \{b\}), \\ \left. (\{B\}, \{\}, \{b\}), (\{d\}, \{\}, \{a\}) \right\}$$

Suppose to compute  $\text{MuMa}(C, 2)$  under the positive reaction system, that is  $(\{D, a\}, \{B, C, b, c\})$ .

- Now, consider the formula  $F^{\mathcal{AwI}} = \{D, a, c\} \in \mathcal{F}$ , whose minimal initial context satisfying it will be  $\mathcal{M}(F^{\mathcal{AwI}}) = \{D, a, c\}$ . Start to verify this context:

$$D, a, c \rightarrow c, A \rightarrow a$$

$C \notin D_3$ , so  $\{D, a, c\}$  isn't verified.

- The corresponding formula in  $\mathcal{A}$  will be  $F^{\mathcal{A}} = \{D, \neg A, \neg C\}$ , all initial contexts satisfying it will be  $D_0^{F^{\mathcal{A}}} = \{\{D\}, \{B, D\}\}$ , for which the traces are:

$$D \rightarrow B, A \rightarrow C, D$$

$$B, D \rightarrow D, A \rightarrow C, B$$

For all configurations, there is  $C$  in  $D_3$  and this means that  $D \neg A \neg C$  is verified.

The point is that in  $\mathcal{A}$  in either case, the following reactions may initially be triggered:

$$(\{D\}, \{A\}, \{A\}) \quad (\{D\}, \{B\}, \{B\})$$

This rules in  $\mathcal{AwI}$  will be traduced as:

$$(\{D, a\}, \{\}, \{A\}) \quad (\{D, b\}, \{\}, \{B\})$$

So under  $\mathcal{AwI}$  at context 0 one should have at least  $\{D, a, b\}$  in order to be able to have  $C$  at step 3.  $\blacklozenge$

The example should have explained that any discrepancy between the two results of the verification trial in the two systems originates in the under-specificity of formulas in  $\mathcal{AwI}$ . Under the system without inhibitors, verification phase has some additional terms that could render some formulas verified in  $\mathcal{A}$  not verified in  $\mathcal{AwI}$ .

Now let proceed to a second important result, which concerns a substantial simplification that can be made on the verification phase when under  $\mathcal{AwI}$ . Analysing the dynamics of the system, despite having to deal with generally more restrictive traces, it can be seen that the system without inhibitors adopts additive behaviour. The term additive means that, given any execution step that from a set results in another, if the source set is enriched with new elements, however, the result will be a super-set of the previous outcome.

**Lemma 4.5.** Given  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$ , and given two sets  $D$  and  $E$ , such that  $D \rightarrow E$ , then:

$$\forall M, N : D \cup M \rightarrow N \implies E \subseteq N$$

*Proof.* If  $D \cup M = D$ , then  $M$  is composed only by elements already present in  $D$ . Given that  $D \rightarrow E$  for hypothesis, then this fact it still applies in  $D \cup M$ . If  $D \cup M \subset D$ , then  $M$  is such that there are also items not belonging to  $D$ . Due to the fact that we are in a system without inhibitors, reactions previously enabled with  $D$  remain enabled with  $D \cup M$ , since there will never be elements in  $M$  that block such reactions. A set  $D \cup M$  will eventually enable other reactions in  $A_{AwI}$ , due to the presence of the elements in  $M$ . This is sufficient to conclude that  $N = \text{res}(D \cup M)$  will contain  $E = \text{res}(D)$  and that therefore  $E \subseteq N$ .  $\square$

Because of this behaviour introduced by the inhibitor transformation, it will be possible to proceed with the verification of a formula by directly checking that the minimum initial context satisfying a formula generates a trace leading to  $s$ .

**Corollary 4.6.** Given a system  $\mathcal{AwI} = (S_{AwI}, A_{AwI})$ ,  $\mathcal{F} = \gamma(\text{MuMa}(s, n))$  and a formula  $F \in \mathcal{F}$ , if  $\mathcal{M}(F)$  is such that  $\mathcal{M}(F) \dashrightarrow^{n+1} D_{n+1} \ni s$  then:

$$\forall D_0^F : D_0^F \models F \implies D_0^F \dashrightarrow D_{n+1} \ni s$$

*Proof.* If  $F = \bigcup_{i=1}^k \ell$ , then  $\mathcal{M}(F) = \{\ell_1, \dots, \ell_k\}$ . This means that from  $\{\ell_1, \dots, \ell_k\}$  it comes to some state  $D_{n+1}$  containing  $s$ , namely:

$$\{\ell_1, \dots, \ell_k\} \rightarrow D_1 \rightarrow \dots \rightarrow D_{n+1} \ni s$$

Any other  $D_0^F$  will be such that  $\mathcal{M}(F) \subseteq D_0^F$ , for definition of initial context satisfying  $F$ . Then in general one will start from an initial context  $\{\ell_1, \dots, \ell_k, m_1, \dots, m_o\}$ , with  $o = 0, \dots, |S_{AwI}| - k$ .

For the Lemma 4.5, a reaction system without inhibitors follows additive behaviour. The additional instances  $\{m_1, \dots, m_o\}$  will eventually enable other reactions, from which a subset of products  $res(\{m_1, \dots, m_o\})$  will be derived. So the general trace will roughly be as follows.

$$\mathcal{M}(F) \cup \{m_1, \dots, m_o\} \rightarrow D_1 \cup res(\{m_1, \dots, m_o\}) \rightarrow \dots \rightarrow D_n \cup M \rightarrow D_{n+1} \cup res(M)$$

Since  $s \in D_{n+1}$  by hypothesis, then it is proved.  $\square$



# Chapter 5

## A tool for dealing with predictor

In this chapter, we will present the MuMa Predictor tool[28], an application with a graphical user interface whose main purpose is to perform the predictor computation for a reaction system, given a set of products and the number of steps.

### 5.1 General overview

MuMa Predictor supports the definition of a reaction system either in a progressive manner, adding one reaction at a time, or by uploading a suitably formed file containing the entire set of reactions.

Once the reaction system is uploaded, by specifying one or more products and a certain number of steps, the tool can output an approximate version of the predictor for those parameters. The computation is based on the Must/Maybe set concepts, so the user will be able to give a first interpretation of what is needed in the initial context to arrive at the desired products in the required number of steps.

A second functionality of MuMa Predictor will allow a more precise information, by computing effectively for each possible set of formulas the ones that will lead to the defined products. A convenient table will allow a compact view of that result, representing all the verified formulas by the the smallest set of verified sub-formulas. There is also a convenient filtering system: formulas can be filtered through a search bar that allows you to specify which instances should be there, or to filter only verified formulas, or only unverified formulas.

The system adopts an internal mechanism capable of detecting and recording possible system attractors during formula verification. When attractors have been found, it is possible to view them in the form of a clear and ordered list.

Finally, speaking of optional features, MuMa Predictor has a small menu from which settings can be chosen quickly. Moreover it is possible to switch from the version of reaction system with inhibitors to the one without inhibitors and vice versa. In case of switching, the system automatically repeat the steps performed in the previous version. Some operations are performed with multithreading, and from the menu it is possible to adjust the number of threads either manually or automatically, taking information on how many cores the machine hosting the tool actually has.

## 5.2 Implementation

Java was chosen as the language for the implementation of MuMa Predictor. In Java, there are many libraries and frameworks that could help to quickly and properly develop all the components needed to implement a tool with the previous described functionalities.

First to mention is a very intuitive-to-use framework called Java Swing,

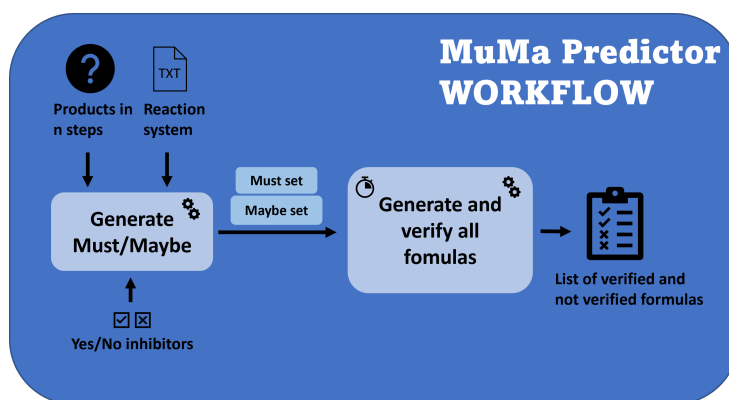


Figure 5.1: General workflow for MuMa Predictor

which is aimed at the development of graphical user interfaces. Other very useful Java libraries were Guava and BitSet. With the former it was possible to implement an initial computational logic, both for approximation and for set-based verification of formulas. With the second it was possible to improve some aspects of the logic adopted with Guava, shifting the view to the use of bit masks, thus introducing improvements on memory management as well. The implementation presented below uses precisely the latter view.

A workflow of how the tool works is shown in the Figure 5.1. As we can see, it exactly follows the three theoretical steps discussed in Chapter 3, section 3.5. Below it is reported the pseudocode describing the main code of the

tool.

---

```

1 function main (rs, req, steps);
  Input : rs - a reaction system
          req - requested products
          steps - number of steps
2 must_set, maybe_set  $\leftarrow$  computeMuMaPredictor(rs, req, steps);
3 all_formulas  $\leftarrow$  generateAllSets(must_set, maybe_set);
4 table  $\leftarrow$  {};
5 foreach f  $\in$  all_formulas do
6   | bb  $\leftarrow$  checkFormula(f, req, steps, table);
7   | table.insert(f, bb);

```

---

The aspects we will focus on will mainly be how Must/Maybe sets are generated and how formula generation and verification take place. These steps are contained in the functions underlined in main.

### 5.2.1 Must/Maybe set generation

In general, the must/maybe sets obtainable from the predictor calculation are computed as shown in Figure 5.2.

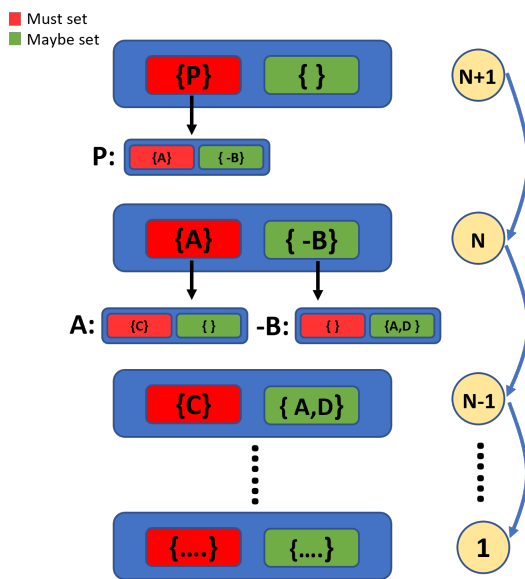


Figure 5.2: Implementation diagram of predictor computation.

At each step, the code considers what is in must and what is in maybe. For each instance that is part of one of these sets, we consult a table containing what is needed in terms of the must and maybe sets to ensure the presence or absence (here marked with a - in front of the instance) of that object.

Combining the information, we thus derive the must and maybe set at a certain execution step. Before discussing the Must/Maybe sets obtainable as a result of the predictor approximation, we need to initialize a table to see which Must/Maybe sets will account for the presence or absence of a single instance during the execution steps. We will use + and - to respectively refer to the operation of set union and set difference. To index elements that are part of a pair, we use a tuple-style notation: [1] for the first, [2] for the second.

The following is the pseudocode of what needs to be done for this procedure.

---

```

1 function initializeMustMaybe (rs);
  Input : rs - a reaction system
  Output: Four maps containing for all instances the Must/Maybe set
           synthesizing its presence/absence
2 must, maybe  $\leftarrow \{\}, \{\}$ ;
3 mustN, maybeN  $\leftarrow \{\}, \{\}$ ;
4 foreach  $p \in rs.allProducts()$  do
5   must.insert( $p$ , rs.commonR( $p$ ), rs.commonI( $p$ ));
6   maybe.insert( $p$ , rs.allR( $p$ ) - must[p][1], rs.allI( $p$ ) - must[p][2]);
7   if must.size == 1  $\wedge$  maybe.isEmpty() then
8     mustN.insert( $p$ , rs.commonR( $p$ ), rs.commonI( $p$ ));
9     maybeN.insert( $p$ ,  $\emptyset$ ,  $\emptyset$ );
10  else
11    mustN.insert( $p$ ,  $\emptyset$ ,  $\emptyset$ );
12    maybeN.insert( $p$ , rs.allI( $p$ ), rs.allR( $p$ ));
13 return must, maybe, mustN, maybeN

```

---

In must and maybe are stored respectively the set of must and maybe resources that allow the presence of  $p$ , in mustN and maybeN the corresponding resources to be associated with the absence of  $p$ . In general, the function `map.insert( $p$ , lit, lit_not)` creates in map a new entry with  $p$  as key, and ( $lit$ ,  $lit\_not$ ) as value. They are an ordered pair of sets, the former consisting of present instances and the latter of absent instances.

Intuitively, in musts to produce  $p$  go all the reactants and inhibitors common to all reactions, in maybe all the rest of the resources needed in the rules, except what is must.

For  $p$  to be absent, however, at least one reagent must be missing or at least one inhibitor must be present. So it all goes in maybeN, swapping in insert inhibitors for reagents and vice versa. The only case in which something becomes obligatory for the absence of  $p$  is when  $p$  can only be produced by one reaction and that reaction has only one reactant.

The way in which the must and maybe set for predictor are updated



in the actual step is based on when an instance is at the previous step in maybe or must; this fact is summarised in the procedures `updateMust` and `updateMaybe`.

The `updateMust` function updates sets when the  $p$  instance is in must. When this happens, only what is must for the presence/absence of  $p$  survives in `tmp_must`, everything else goes into `tmp_maybe`.

---

```
1 function updateMust ( $p, mu, ma, tmp\_must, tmp\_maybe$ );
```

**Input** :  $p$  - instance to be justified

$mu$  - must for presence/absence of  $p$

$ma$  - maybe for presence/absence of  $p$

$tmp\_must$  - actual must set

$tmp\_maybe$  - actual maybe set

**Output:** must and maybe updated after analyzing  $p$

```
2 tmp_must[1] += mu[p][1];
```

```
3 tmp_must[2] += mu[p][2];
```

```
4 tmp_maybe[1] += ma[p][1];
```

```
5 tmp_maybe[2] += ma[p][2];
```

```
6 return tmp_must, tmp_maybe
```

---

For `updateMaybe`, the idea is that when an instance  $p$  is in a maybe set, since its presence is doubtful, then none of the resources that produce it become necessary to ensure that it is present.

---

```
1 function updateMaybe ( $p, mu, ma, tmp\_must, tmp\_maybe$ );
```

**Input** :  $p$  - instance to be justified

$mu$  - must for presence/absence of  $p$

$ma$  - maybe for presence/absence of  $p$

$tmp\_must$  - actual must set

$tmp\_maybe$  - actual maybe set

**Output:** must and maybe updated after analyzing  $p$

```
2 tmp_maybe[1] += mu[p][1] + ma[p][1];
```

```
3 tmp_maybe[2] += mu[p][2] + ma[p][2];
```

```
4 return tmp_must, tmp_maybe
```

---

The code for determining the MuMa predictor is as follows. Given a *req* requested set of products and a number of steps *steps*, this code outputs the must and maybe set needed at initial context to obtain *req* after *steps* number of steps.

---

```

1 function computeMuMaPredictor (rs, req, steps);
   Input : rs - a reaction system
           req - the set of requested products
           steps - the number of steps
   Output: must and maybe set for obtaining req after steps steps
2 must, maybe, mustN, maybeN  $\leftarrow$  initializeMustMaybe(rs);
3 pred_must  $\leftarrow$  ({req}, {});
4 pred_maybe  $\leftarrow$  ({}, {});
5 while steps > 1 do
6   tmp_must  $\leftarrow$  ({}, {});
7   tmp_maybe  $\leftarrow$  ({}, {});
8   foreach mu  $\in$  pred_must[1] do
9     tmp_must, tmp_maybe  $\leftarrow$ 
       updateMust(mu, must, maybe, tmp_must, tmp_maybe);
10  foreach mu_n  $\in$  pred_must[2] do
11    tmp_must, tmp_maybe  $\leftarrow$ 
       updateMust(mu_n, mustN, maybeN, tmp_must, tmp_maybe);
12  foreach ma  $\in$  pred_maybe[1] do
13    tmp_must, tmp_maybe  $\leftarrow$ 
       updateMaybe(ma, must, maybe, tmp_must, tmp_maybe);
14  foreach ma_n  $\in$  pred_maybe[2] do
15    tmp_must, tmp_maybe  $\leftarrow$ 
       updateMaybe(ma_n, mustN, maybeN, tmp_must, tmp_maybe);
16  pred_must, pred_maybe  $\leftarrow$  tmp_must, tmp_maybe;
17  steps  $\leftarrow$  steps - 1;
18 return pred_must, pred_maybe

```

---

Initially, the product *req* whose predictor is to be worked out is put into *must*. The procedure *computeMuMaPredictor* at each step updates two temporary *tmp\_must* and *tmp\_maybe* sets taking into account the *pred\_must* and *pred\_maybe* at the previous step. The update is done by calling *updateMust* or *updateMaybe* depending on whether one is dealing with an instance in *pred\_must* or in *pred\_maybe*. The loop stops when step 1 is reached and the procedure return *pred\_must* and *pred\_maybe*.

### 5.2.2 Generate all formulas and verification

Generating all possible formulas, given the must and maybe calculated with `computeMuMaPredictor`, results in generating the power set of all maybe members (both present and absent) and then adding the elements in must to each of the obtained set. If a resulting formula contains the same element both present and absent (a check that is done with `isContradictory()`), it will be removed from the all formulas list.

---

```

1 function generateAllSets (must, maybe);
  In  : maybe - pair of maybe and maybe not list
        must - pair of must and must not list
  Out: all formulas respecting must/maybe semantics
  /* Suppose to add a  $\neg$  symbol both to must[2] and maybe[2] items */
2 all_maybe  $\leftarrow$  maybe[1] + maybe[2];
3 all_must  $\leftarrow$  must[1] + must[2];
4 all_formulas  $\leftarrow$  generatePowerset(all_maybe);
5 foreach f  $\in$  all_formulas do
6   | f  $\leftarrow$  all_must + f;
7   | if isContradictory(f) then
8     |   all_formulas.remove(f);
9 return all_formulas

```

---

The code for checking a formula in `all_formulas` in a system with inhibitors is as follows:

---

```

1 function checkFormula (rs, f, req, steps, table);
  In  : f - a conjunctive formula
        table - table of the more general verified formula
  Out: results of the control
2 if table.isAnyElSubsetOf(f) then
3   | return True
4 present, absent  $\leftarrow$  extractPresence(f);
5 allPossibilities  $\leftarrow$  generatePowerset(rs.allInstances());
6 allPossibilities.filter(\s  $\rightarrow$  s.containsAll(present))
7   .filter(\s  $\rightarrow$  !s.containsAtLeast(absent));
8 foreach conf  $\in$  allPossibilities do
9   | if !simulateEvolutionOverReactionSystem(conf, req, steps) then
10    |   return False
11 return True

```

---

Only the most 'general' verified formulas are stored in `table`, i.e. formulas for which no subformula has already been verified. Initially, it is then checked, in accordance with the Theorem 3.8, that there is a formula in the table that is a subset of `f` and if so, true is returned immediately. If we

are not lucky enough, then we must generalise all possible initial configurations allPossibilities that satisfy  $f$  and check that they all lead to  $req$  after steps.

A system without inhibitors can enjoy an important simplification of `checkFormula`, i.e. check the simulation result only with the minimum configuration of  $f$ , since it was proven with Corollary 4.6, that this is sufficient to verify the entire formula  $f$ . In this case the minimal configuration is obtained by considering only instances in  $f$  that are not dummy.

---

```

1 function checkFormulaNoIni ( $rs, f, req, steps, table$ );
  In :  $f$  - a conjunctive formula
         $table$  - table of the more general verified formula
  Out: results of the control
2 if  $table.isAnyElSubsetOf(f)$  then
3   | return True;
4 present  $\leftarrow$  onlyRealEntities( $f$ );
5 if !simulateEvolutionOverReactionSystem( $present, req, steps$ ) then
6   | return False;
7 return True;

```

---

**Multithreading feature.** As will be stressed in a moment, these two functions `generateAllSets` and `checkFormula` can be time-consuming in direct proportion to how complicated the reaction system is. A multithreaded solution is needed to improve these timings. First, the presence of a power set

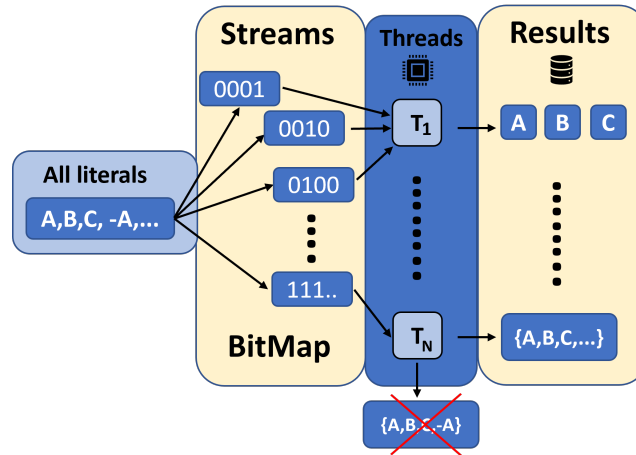


Figure 5.3: Workflow of multithreading system for powerset computation

calculation that depends on the number of entities  $|S|$  gives us a time complexity of  $O(2^{|S|})$  only for function `generatePowerset`, which is even worse

when it comes to a reaction system without inhibitors for the presence of new entities, with  $|S_{AwI}| > |S|$ .

The multithreaded version for powerset computation, summarised in Figure 5.3, uses Java's Stream to encode the presence/absence of the various literals in binary sequences. These sequences are processed in parallel with a fork/join mechanism, so that only non-contradictory sequences are kept, and finally collected in a list that will contain the already filtered power set. Another problem is that each formula in the list from the previous step

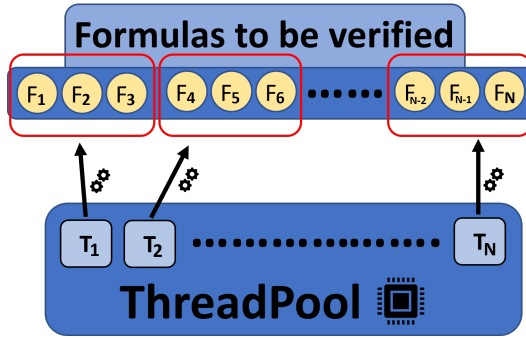


Figure 5.4: Workflow of multithreading system for verification stage

should be passed to `checkFormula`. In `checkFormula` we have to check for all initial configurations satisfying the input formula if, after a simulation up to the desired number of steps, it is obtained the wanted product. This operation can also be expensive both in terms of the number of configurations and the number of steps to be simulated. Moreover, it may be repeated lots of times. This motivates our choice to provide a multithreaded solution also for verification phase.

The idea is summarised in Figure 5.4. We divide the list of possible formulas between a fixed number of thread in a threadpool. Each thread works on a list partition having the same number of formulas each, to make tasks computation balanced. At the end, threads inserts the result for all the formulas in the task in a shared list, that will be displayed in the user interface.

**Attractors feature.** Exploiting what is described in Section 3.4.1, we have developed a method to find and exploit the system's attractors during the verification phase. The reason why collecting information on attractors is always about improving the verification phase, which from the results obtained is the most expensive phase. In the verification phase, a number of simulations equal to the number of configurations satisfying a formula must be carried out for each formula that can be generated from the must/maybe

sets. Each simulation is carried out for  $\text{steps} + 1$  states, so if  $\text{steps}$  is large, the duration weight of this phase is worse.

In a hash table, all the states in a cycle are stored as keys. Each state is assigned a list value representing the state sequence from the key state that is needed to complete the cycle. When an attractor is encountered, the hash table will return a value for the particular state belonging to the loop that was met. This way, we know where we are in the loop and can easily compute in one move which state we will end up in. This functionality has

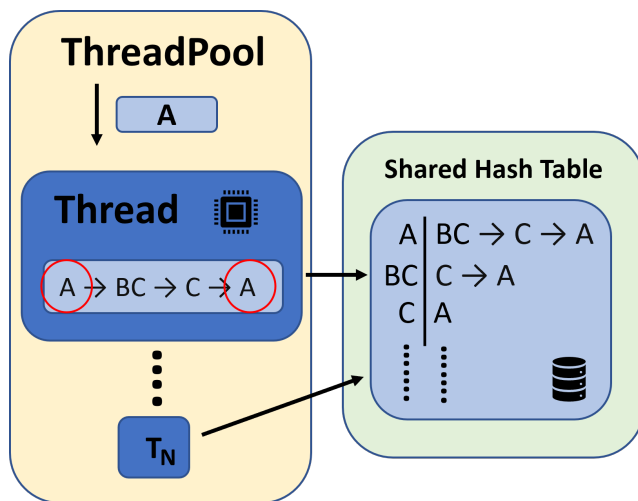


Figure 5.5: Multithread organization for attractor detection

also been integrated into the multi-thread version and is structured as in Figure 5.5. Here, the hash table is a shared data structure. Each thread in the threadpool, during the verification phase, accesses the table to check whether the simulation has finished within a known cycle. If this is not the case, then the thread saves in a local copy all the states passed through during the current simulation and, as soon as it finds a repeated one, it modifies the table to add the new cycle.

### 5.3 Experimental

At this point, we wanted to test two different aspects of the MuMa Predictor. Firstly, we evaluated the performance through benchmarks with respect to the new ideas we introduced in this thesis, such as reaction systems without inhibitors, attractors and the multithreaded implementation. Secondly, we converted existing boolean networks into reaction systems and attempted to capture dynamic behaviour already known from studies of these networks.

### 5.3.1 Benchmarks

**Reaction systems vs positive reaction systems.** This test is intended to demonstrate in the verification phase the effectiveness of using the positively converted reaction system rather than the original reaction system. The starting system consists of 20 instances  $S_1, \dots, S_{20}$  and 33 different reactions. The starting point was the calculation of the ancestor of  $S_7$  for the first 4 steps. In these cases, despite coming up with formulas with a very simple structure, the verification phase is very difficult to deal with for the classical reaction system, since only for a few instances, say  $k < 5$ , is the possible presence/absence fixed, and then the verification of the trace has to be carried out from  $2^{16}$  different configurations.

Steps	R.S.	Positive R.S.	Formulas
1	35.64s	4ms	1
2	130.03s	4ms	5
3	undef	17ms	255
4	undef	31ms	3071

As we can see, thanks to the version without inhibitors, verification formulae that seem intractable with a classic reaction system become very feasible.

**Attractors vs no-attractors.** Another aspect to be investigated is the performance increase introduced by the attractor method. The starting system

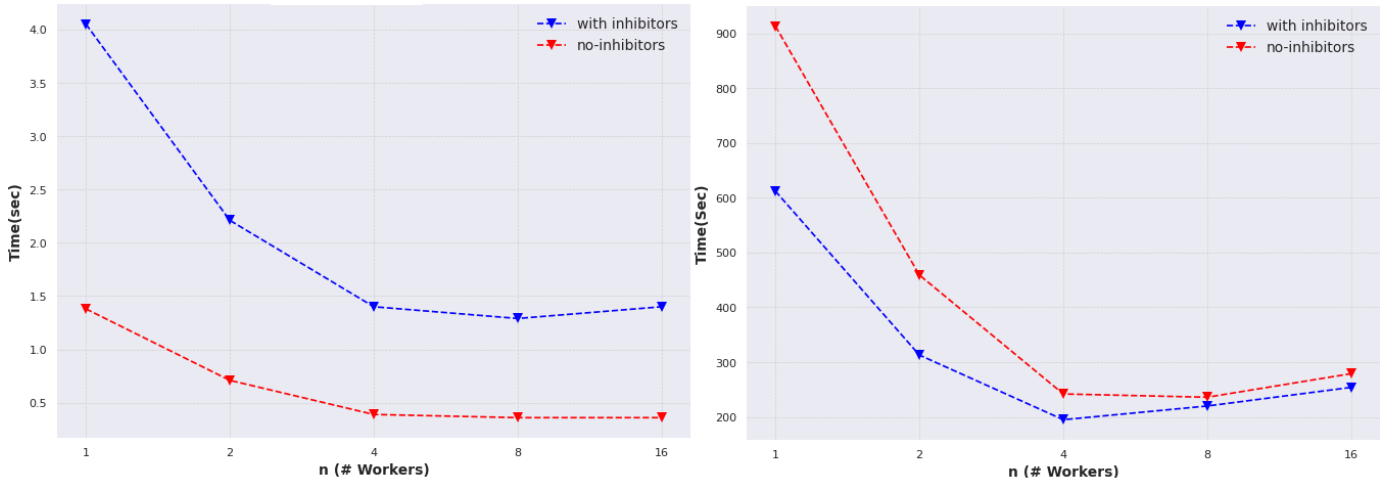


Figure 5.6: Comparing of execution time attractors vs no-attractors method

have 23 reactions and 10 instances. We computed a predictor for a set of products *chd1 p27 Rb*, specifying 10000 steps. Being a very large number,

the expected behavior was to obtain a very long verification phase. From the image we can observe two things. First of all, we note the undeniable advantage in using the attractor method to simplify the verification phase, in fact with 1 thread we go from a time of about 900 seconds to just 4 seconds. The improvements are distinct, especially for the version without inhibitors, which we have seen is generally the fastest in the verification phase.

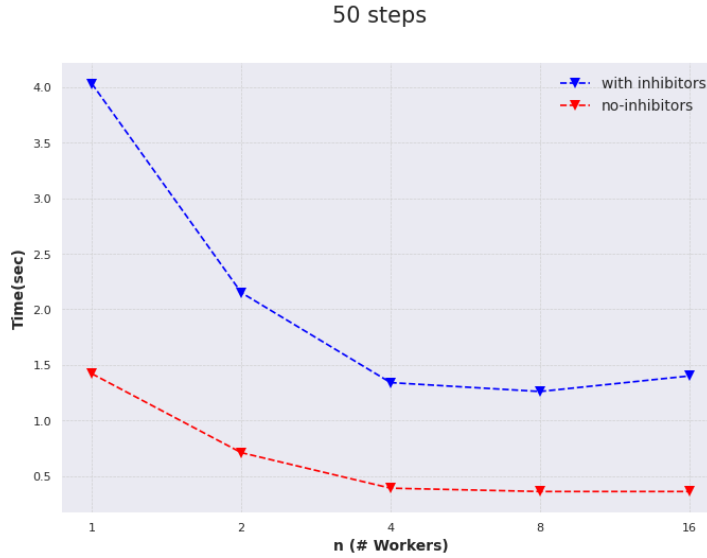


Figure 5.7: Execution time with 50 steps

Furthermore, from the Figure 5.6, we can see that the version without inhibitors is even slower than the one with. We saw with Corollary 4.6 that the positive reaction system has difficulty verifying a formula as true if it turns out to be poorly specified. Consequently although the verification time for a single formula is better, it wastes more time verifying false formulas. This problem, combined with the absence of the attractors, results in a worsening of performance to the point where the classical system is preferable.

Another interesting aspect to note is the one in the Figure 5.7. As we can see, due to the use of attractors the execution time to search for a 10000 steps ancestor is similar to that obtained by computing a 50 steps ancestor. This is not only a consequence of using attractors, but results from the finite and deterministic nature of the traces obtainable from a certain reaction system.

**Multithread performance.** Finally, we present the results obtained in the multithread-specific performance analysis.



We evaluated two classic metrics of multithreaded implementations, namely scalability and speedup. Scalability is the degree to which workload throughput benefits from the availability of additional processors. It is usually expressed as the quotient of the throughput of the workload on a multiprocessor divided by the throughput on a comparable uniprocessor. Instead speedup is the ratio of performance for the entire task using the enhancement and performance for the entire task without using the enhancement.

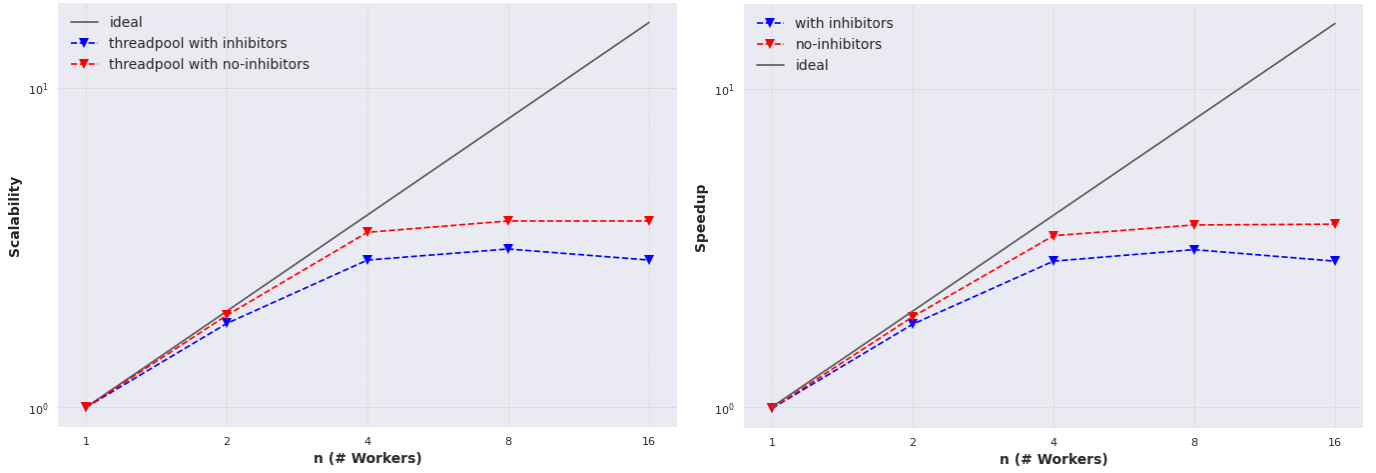


Figure 5.8: Scalability vs Speedup for a 10000 steps execution

The reaction system used as a benchmark has 23 reactions and 10 instances. The reactions were chosen in such a way that the abstraction was imprecise; in fact, the result is a maybe set containing all possible reaction system literals along with negative literals. It means that the resulting concretization phase is a list containing all possible non-contradictory formulas that can be formatted with any conceivable literal and its negate. In this way the threads, which work independently on formula partitions during verification, can run at full capacity.

As we can see from the Figure 5.8, scalability and speedup are quite similar, since the execution time of sequential code is close to the execution time with a single thread. The code turns out to scale well as long as the number of threads is kept less than or equal to 4. This can be understood from the fact that the performance for the entire task executed by  $k$  threads ( $k \leq 4$ ) improves by about  $k$  times compared to the execution time without multithreading. This behavior is the best we can expect from the point of view of speedup and scalability. On the other hand, for  $k > 4$ , this tendency seems to vanish, showing, for example, that with 8 threads, execution is as fast as if we were still using 4 threads. One possible explanation for this behavior is that with this specific reaction system not all  $k > 4$  threads are working

with the same workload, for instance one thread in verification might get more derived formulas than the others. Finding a derived formula means at the level of chunk of code to execute only one access to the hash table. Overall, the results of the analysis lead to calibrate a level of parallelism around 4, since with a larger number of workers the expected benefits do not return.

### 5.3.2 Simulation results

To test the correctness and potential of the tool, we considered two Boolean networks convertible into reaction systems, and once we obtained the corresponding reaction systems, we studied their dynamic properties through the MuMa Predictor and compared our results with those obtained by the creators of the networks. The experiments conducted by the authors in both networks were carried out with a well-known biological Boolean network analysis tool called GINSim[27].

The goal is to demonstrate the consistency of the results obtained through reaction systems with respect to the starting boolean networks, , showing the potential of MuMa Predictor as a supporting tool for studying the dynamic properties of complex boolean networks such as those under consideration.

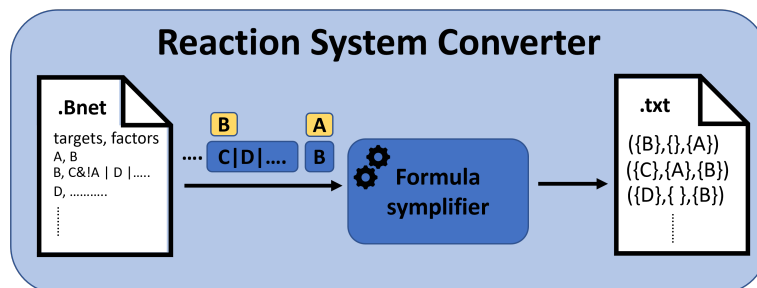


Figure 5.9: Workflow of reaction system converter

**Converting a boolean network.** Most of the papers that we have read study systems that are represented through Boolean networks through a software called GINSim. We decided to write a small tool in Python, named "Reaction System Converter", for converting a Boolean Network file into a reaction system. It uses the `boolean` library to be able to apply the various simplifications to boolean expressions.

The workflow of the tool is summarized in the Figure 5.9. The boolean network format used is quite simple, we have an example in Figure 5.10.

As we can see we have targets and their factors. Each factor is made by the following classical boolean operators:

- ! is the negation
- & is the AND operator
- | is the OR operator

```

targets, factors
Aop,      !Rl
Cnk,      Cnk
Pvr,      !Pvf1&!Pvf2&Pvf3 | !Pvf1&Pvf2 | Pvf1
Raf,      Ras&Cnk&Src42&Ksr
Ras,      Sos&!Sty
Rl,       Dsor1&Msk

```

Figure 5.10: An example of boolean network format (specifically it is a .bnet format)

From expressions written in this way, we can transform most of boolean network in our format for reaction system, explained in Section 6.1, simply translating line by line.

Before proceeding with the translation, however, it is necessary to simplify the expressions in OR associated to every product, since they can be very long and complex. The result of the simplification will be a DNF formula and we can proceed to translation in our format.

For each expression in OR associated with a target, a new reaction having that target as product must be generated within the reaction system. That reaction will have as reactants all the elements activating the target and as inhibitors all the elements inhibiting the target. For the targets having associated a boolean expression composed only by negative entities, we have introduced in the reaction system a dummy entity called *create*, since every reaction must have at least one reactant to be considered well-formed. We need also to be able to create it. Consequently, we have introduced a new reaction that allows *create* to self-generate.

In order to avoid unexpected behaviour on reaction system traces, it is assumed that *create* is always present in the initial state and consequently in all other states. Moreover, if present in must or maybe, it is omitted because it does not model any real entity.

### Mammalian cell cycle

This first experiment examines a complex regulatory biological network, which originated as a logical version of a model built with ODEs to control

the mammalian cell cycle[9].

The model examines a succession of molecular events that lead to the reproduction of a cell's genome and its division through Cell Mitosis. This process is tightly controlled and must be coordinated with the overall growth of the organism. The coordination is regulated by signals from certain instances that activate or inhibit the stages leading to cell growth. These instances are partly expressed in the network itself.

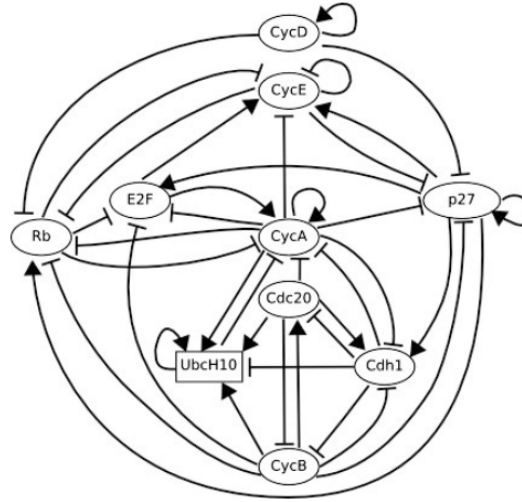


Figure 5.11: Boolean network describing mammalian cell cycle from [9]

Figure 5.11 represents the activity of each node, that is a regulatory element. The edges represent regulation types, that can be activating (i.e. normal arrows) or inhibiting (i.e. blunt arrows).

The conversion to reaction system allowed by the code described before, produced a total of 23 different reactions, which governs the activity of 10 instances.

Of particular interest to the authors is the asymptotic behaviour of the system. This involves the study of attractors and fixed points. In the paper, the authors identified two attractors: a fixed-point attractor and a 7-state attractor.



The fixed-point attractor has as instances p27 cdh1 Rb. Quoting what the authors say: "this state is reached from all the other states lacking CycD activity". We tried to study the predictor of this set of products, for several possible steps. Already from a relatively small number of steps, the result of the approximate predictor settles as follows:

Must( $\neg$  CycD)

Maybe(Cdc20, Cdh1, CycA, CycB, CycE, E2F, Rb, UbcH10,  
p27,  $\neg$  Cdc20,  $\neg$  Cdh1,  $\neg$  CycA,  $\neg$  CycB,

$$\neg \text{CycE}, \neg \text{E2F}, \neg \text{Rb}, \neg \text{UbcH10}, \neg \text{p27})$$

According to the approximation obtained, in all states from which the stable state will be reached, CycD will be missing. When we then start the verification phase, generating the actual solutions, the result is that all formulas having  $\neg \text{CycD}$  are verified.

Formula	Is verified?
 $\neg \text{CycD}$	

Filter by: ☒ Not Satisfied ☒ Satisfied

Figure 5.12: Formulae verified by MuMa Predictor.

This means that the information obtained with the tool is consistent with the authors' results.

When we go to analyse the attractors found during the verification phase, we find the one whose predictor was computed and nothing else.

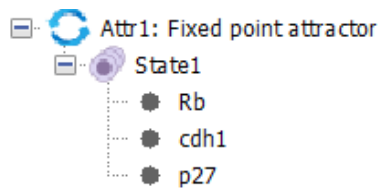


Figure 5.13: Attractors found by MuMa Predictor.

The other possible attractor is a cycle having period 7. On the study of this attractor, the authors say: "...in the presence of CycD, all trajectories lead to a unique dynamical cycle, made of a sequence of seven successive states". By investigating on the predictor of one state of this cycle, we find that for a number of steps greater or equal to 3, the approximation found always becomes the following:

Must(CycD)  
 Maybe(Cdc20, CycA, CycB, CycE, E2F, Rb, UbcH10, cdh1,  
 p27,  $\neg \text{Cdc20}$ ,  $\neg \text{CycA}$ ,  $\neg \text{CycB}$ ,  $\neg \text{CycE}$ ,  $\neg \text{E2F}$ ,  
 $\neg \text{Rb}$ ,  $\neg \text{UbcH10}$ ,  $\neg \text{cdh1}$ ,  $\neg \text{p27}$ )

step	CycD	Rb	E2F	CycE	CycA	p27	Cdc20	cdh1	UbcH10	CycB
1	1	0	1	1	1	0	0	1	0	0
2	1	0	0	1	1	0	0	0	0	0
3	1	0	0	0	1	0	0	0	1	1
4	1	0	0	0	1	0	1	0	1	1
5	1	0	0	0	0	0	1	1	1	0
6	1	0	1	0	0	0	0	1	1	0
7	1	0	1	1	0	0	0	1	0	0

Table 5.1: Attractor lenght 7 found by the tool

Again, the result of the approximation captured the information highlighted by the authors themselves. Even when looking at the attractors found by MuMa predictor, we find exactly the same cycle length 7.

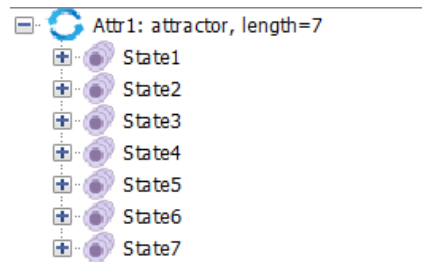


Figure 5.14: Attractors found by MuMa Predictor.

### Fission yeast cell cycle

Our second and final experiment examines a Boolean network that synthesises the regulatory processes controlling the cell cycle of *Schizosaccharomyces pombe* [13]. *S. pombe* is a widely used yeast species in research and its cell cycle has been studied extensively.

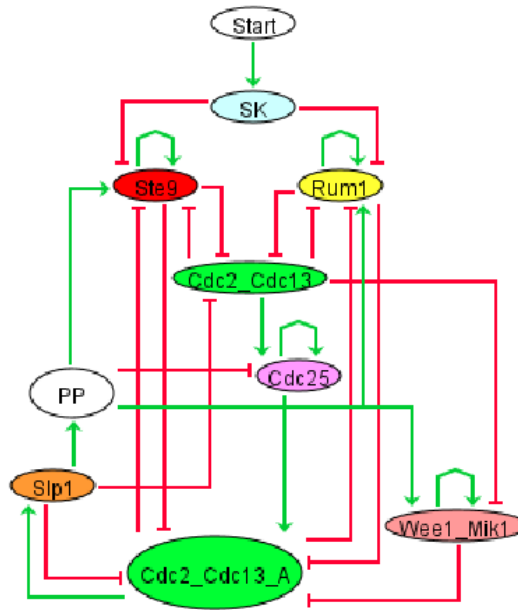


Figure 5.15: Boolean network of yeast cell division from [13].

The authors outlined a Boolean network simulating the process of yeast cell division. The nodes of the network are the proteins that control the stages of cell division, the edges express an inhibiting or activating interaction, similar to the network for mammary cells. Again, in the paper studying this network [13], the dynamic behaviour is analyzed, also referring to the case where a synchronous update is assumed.

The stable states found by the authors are almost all fixed point attractors and there are 13 of them in total, as we can see from Figure 5.16.

As we can see from the table in Figure 5.16, there is a Start entity, which according to the authors initially triggers a preliminary phase that activates another entity called SK. We have chosen to simulate the Start activity through the virtual entity create. Apart from that, the reaction system was obtained by the usual conversion procedure and comprises a total of 22 reactions for 10 entities.

It is emphasised by authors that for every possible trace started with one of the  $2^{10} = 1024$  possible initial states, the system reaches these attractors in at most 8 steps, as depicted in Figure 5.17. This was our starting point for

Attractor	Type	Basin size	Start	SK	Cdc2/Cdc13	Ste9	Rum1	Slp1	Cdc2/Cdc13*	Wee1/Mik1	Cdc25	PP
1	FP	762	0	0	0	1	1	0	0	1	0	0
2	LC	208	0	0	0	0	0	0	0	0	1	1
	LC	0	0	0	0	0	0	1	0	0	1	0
	LC	0	0	0	1	1	1	0	1	1	0	0
3	FP	18	0	0	0	0	1	0	0	1	0	0
4	FP	18	0	0	0	1	0	0	0	1	0	0
5	FP	2	0	0	0	1	0	0	0	0	0	0
6	FP	2	0	0	0	1	0	0	0	0	1	0
7	FP	2	0	0	0	1	0	0	0	1	1	0
8	FP	2	0	0	0	0	1	0	0	0	0	0
9	FP	2	0	0	0	0	1	0	0	0	1	0
10	FP	2	0	0	0	0	1	0	0	1	1	0
11	FP	2	0	0	0	1	1	0	0	0	0	0
12	FP	2	0	0	0	1	1	0	0	0	1	0
13	FP	2	0	0	0	1	1	0	0	1	1	0

doi:10.1371/journal.pone.0001672.t003

Figure 5.16: All attractors of the dynamics of the network model. Image taken from [13].

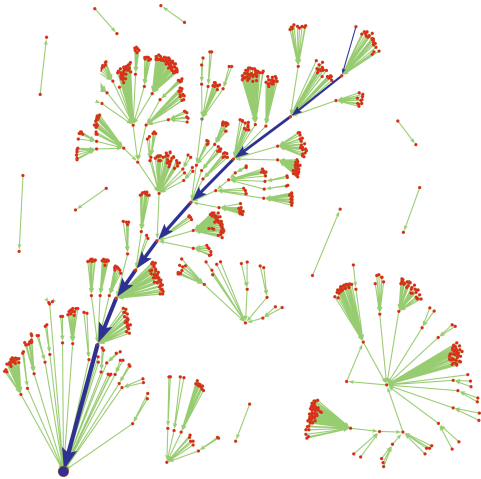


Figure 5.17: State space of the 1024 possible network states and their dynamical trajectories. Image taken from [13].

the dynamic analysis of the reaction system. Specifically, we took the state associated with the fixed-point attractor, namely Ste9 Rum1 Wee1\_Mik1, that was most highly reached of all, having a basin size of 762, and computed its predictor for 8 steps. The approximate result is not particularly interesting, as it puts all possible entities in the maybe set both positively and negatively. At the same time, however, it allows us to study the attractors for all possible initial states.

There are exactly 13 attractors found by the tool. By checking, the tool captures all attractors which are marked 2 to 13 in the Figure 5.18. Attractor number 1, which is the one from which we computed the predictor, was not found. In its place is an attractor of length 9, which also includes



Ste9 Rum1 Wee1\_Mik1.

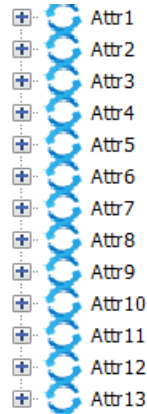


Figure 5.18: Attractor found by the tool.

This result may depend on how we have modelled the behaviour of the Start entity. This entity must only be present in the initial context for initial computation, and then no longer appear in the rest of the trace. Instead, create is also used in successive states as an omnipresent object to trigger all those reactions conditioned uniquely by the absence of the inhibiting entities. Consequently, it is not entirely accurate to model the behaviour of Start with create. In order to solve this problem, one could insert an additional field in the tool to introduce clauses on certain steps, such as that Start should only appear in the first step. This aspect is to be included in future work.



# Chapter 6

## Conclusions

Over-approximation techniques make it possible to handle complex problems, the exact solution of which is NP-hard, and in some ways reduce computational complexity.

In this thesis we used a general methodology called Abstract Interpretation, to formalize an over-approximation in order to solve the difficult problem of computing the predictor of a reaction system. We designed and formalised the backward process in order to extract the possible causes that lead to the production of a set of  $s$  products in  $n$  steps through an over-approximation that allowed us to obtain a linear-time solution for the computation of an abstract predictor. Being an abstraction, the complete process for predictor computation must consist of two additional steps. The first one is concretisation, needed to obtain the set of possible boolean formulae specifying which elements must be present or absent at the initial context. The last step is verification, that consists to check which of these formulae actually generate valid contexts to obtain the result searched for in  $n$  steps.

Concretisation and verification can be time consuming for various reasons. Concretisation involves the computation of power set of the maybe set, an operation known to be exponential with the number of elements in the set. This number depends on the different reactants and inhibitors that constitute each reaction considered for a single product. The verification phase's complexity depends on the number of formulas generated by the previous phase, the number of initial contexts satisfying a formula, and the number of steps performed at each simulation.

In order to simplify the verification phase, we have introduced positive reaction system. Any reaction system can be transformed in positive reaction system, in which inhibitory entities are treated as new entities. Despite of having a larger reaction system, we have the advantage to simplify verification phase, since for all formulas it is sufficient to check the  $n$ -steps long

trace starting from the minimal configuration satisfying that formula. In general, if a formula under a positive reaction system is verified, then it will be verified also in reaction system. However, the reverse may be not true, since some formulae can be underspecified for positive reaction system.

Designing and building the MuMa Predictor gave us the opportunity to work on an applicative version of these concepts. In order to further improve concretisation and verification phases, we implemented a multi-thread version of these stages, together with the attractor concept. We have compared performances of all these features by benchmarks. From results, the union between theoretical improvement and parallel processing allows us to reach good time performances. Moreover, the tool was used to study dynamical behavior of already defined biological systems, represented by boolean networks, in order to verify information consistency found by tool with respect to results already known. The outcome of these experiments is positive, as the tool responds in a manner coincident with that known from the studies. A key point for success is the construction of a reaction system that simulates all the behaviours of a boolean network, which may not be possible in some cases.

## 6.1 Future works

Even if in general a positive reaction system from a performance point of view behaves better than a reaction system, there could be some corner cases which makes more convenient to compute predictor with classical reaction system. During our studies, we have identified two of these cases.

One case is when the resulting positive reaction system grows in the number of reactions to the point where it loses all improvement in the verification phase. A better analysis of how the number of reactions "increase" could help to decide which system to prefer. For instance, one could make a more precise study of  $k_{simpl}$ , the factor that quantifies how many of these additional rules during the transformation are discarded.

The other case is when the majority of formulae in positive reaction systems is not verified, in contrast to the classical one. Intuitively, one solution might be to study the starting structure of the reaction system beforehand. Indeed, the more inhibitors we find for each reaction, the more likely it is that less specific formulae will not be verified.

As far as possible extensions of MuMa Predictor are concerned, one structural limitation we came up against is the list capacity in Java. The maximum number of elements of a list is  $2^{32}$ , a limit that is easily overcome when dealing with the powerset computation of a set. In our case, this is also reflected in the number of possible entities in the reaction system (ap-

proximately 30). To overcome this limitation, we propose implementing a parallel solution based on the divide-and-conquer paradigm using multi-lists. In order to improve the tool's ability to analyse real systems based on boolean networks, it is useful to implement the possibility of adding constraints to trace states in order to be able to simulate behaviour typical of boolean networks that cannot be captured by a reaction system. In this way it is possible to deactivate (or activate) useful reactions up to (from) a certain point in the computation.



## References

- [1] Patrick Cousot and Radhia Cousot. “Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints”. In: *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1977, pp. 238–252.
- [2] Patrick Cousot. “Semantic foundations of program analysis”. In: *Program flow analysis: theory and applications*. Prentice Hall, 1981, pp. 303–342.
- [3] David A Plaisted. “Theorem proving with abstraction”. In: *Artificial Intelligence* 16.1 (1981), pp. 47–108.
- [4] Edmund M Clarke, Orna Grumberg, and David E Long. “Model checking and abstraction”. In: *ACM transactions on Programming Languages and Systems (TOPLAS)* 16.5 (1994), pp. 1512–1542.
- [5] Rance Cleaveland and James Riely. “Testing-based abstractions for value-passing systems”. In: *CONCUR’94: Concurrency Theory*. Springer, 1994, pp. 417–432.
- [6] Bruno Monsuez. “System F and abstract interpretation”. In: *International Static Analysis Symposium*. Springer. 1995, pp. 279–295.
- [7] Peter Ørbæk. “Can you trust your data”. In: *Colloquium on Trees in Algebra and Programming*. Springer. 1995, pp. 575–589.
- [8] Susanne Graf. “Characterization of a sequentially consistent memory and verification of a cache memory by abstraction”. In: *Distributed Computing* 12.2 (1999), pp. 75–90.
- [9] Adrien Fauré et al. “Dynamical analysis of a generic Boolean model for the control of the mammalian cell cycle”. In: *Bioinformatics* 22.14 (2006), e124–e131.
- [10] Nadia Busi. “Causality in membrane systems”. In: *International Workshop on Membrane Computing*. Springer. 2007, pp. 160–171.
- [11] Andrzej Ehrenfeucht and Grzegorz Rozenberg. “Reaction systems”. In: *Fundamenta informaticae* 75.1-4 (2007), pp. 263–280.

- [12] Nadia Busi. “Towards a causal semantics for Brane Calculi”. In: (July 2008).
- [13] Maria I Davidich and Stefan Bornholdt. “Boolean network model predicts cell cycle sequence of fission yeast”. In: *PloS one* 3.2 (2008), e1672.
- [14] Élisabeth Remy, Paul Ruet, and Denis Thieffry. “Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework”. In: *Advances in applied mathematics* 41.3 (2008), pp. 335–350.
- [15] Robert Brijder, Andrzej Ehrenfeucht, and Grzegorz Rozenberg. “A Note on Causalities in Reaction Systems”. In: *ECEASST* 30 (Jan. 2010). doi: 10.14279/tuj.eceasst.30.429.
- [16] Roberta Gori and Francesca Levi. “Abstract interpretation based verification of temporal properties for BioAmbients”. In: *Information and Computation* 208.8 (2010), pp. 869–921. issn: 0890-5401. doi: <https://doi.org/10.1016/j.ic.2010.03.004>. url: <https://www.sciencedirect.com/science/article/pii/S0890540110000611>.
- [17] Robert Brijder et al. “A tour of reaction systems”. In: *International Journal of Foundations of Computer Science* 22.07 (2011), pp. 1499–1517.
- [18] Luca Corolli et al. “An excursion in reaction systems: From computer science to biology”. In: *Theoretical computer science* 454 (2012), pp. 95–108.
- [19] Chiara Bodei, Roberta Gori, and Francesca Levi. “An Analysis for Causal Properties of Membrane Interactions”. In: *Electronic Notes in Theoretical Computer Science* 299 (2013). Proceedings of the fourth International Workshop on Interactions between Computer Science and Biology (CS2Bio’13), pp. 15–31. issn: 1571-0661. doi: <https://doi.org/10.1016/j.entcs.2013.11.003>. url: <https://www.sciencedirect.com/science/article/pii/S1571066113000741>.
- [20] Sepinoud Azimi, Bogdan Iancu, and Ion Petre. “Reaction system models for the heat shock response”. In: *Fundamenta Informaticae* 131.3-4 (2014), pp. 299–312.
- [21] Chiara Bodei, Roberta Gori, and Francesca Levi. “Causal static analysis for Brane Calculi”. In: *Theoretical Computer Science* 587 (2015). Interactions between Computer Science and Biology, pp. 73–103. issn: 0304-3975. doi: <https://doi.org/10.1016/j.tcs.2015.03.014>. url: <https://www.sciencedirect.com/science/article/pii/S0304397515002182>.



- [22] Alberto Dennunzio et al. “Ancestors, descendants, and gardens of eden in reaction systems”. In: *Theoretical Computer Science* 608 (2015), pp. 16–26.
- [23] Artur Męski, Wojciech Penczek, and Grzegorz Rozenberg. “Model checking temporal properties of reaction systems”. In: *Information Sciences* 313 (2015), pp. 22–42.
- [24] Roberto Barbuti et al. “Investigating dynamic causalities in reaction systems”. In: *Theoretical Computer Science* 623 (2016), pp. 114–145.
- [25] Sepinoud Azimi. “Steady states of constrained reaction systems”. In: *Theoretical Computer Science* 701 (2017), pp. 20–26.
- [26] Roberto Barbuti, Roberta Gori, and Paolo Milazzo. “Multiset patterns and their application to dynamic causalities in membrane systems”. In: *International Conference on Membrane Computing*. Springer. 2017, pp. 54–73.
- [27] *GINsim (Gene Interaction Network simulation)*. <http://ginsim.org/>.
- [28] *MuMa Predictor*. <https://github.com/valquake/MuMa-Predictor>.



# How to use

We now present a brief tutorial on how to use our tool.

## Input format and r.s. building

At the top of the MuMa Predictor window, clicking on "file" gives us the option of importing an existing reaction system or creating it from scratch within the tool. The tool takes as input text files representing reaction systems having one of this structure:

```
({CycB},{},{Cdc20})  
({CycA},{Cdc20,Rb,UbcH10},{CycA})  
({CycA},{Cdc20,Rb,cdh1},{CycA})  
({E2F},{Cdc20,Rb,UbcH10},{CycA})
```

A	B	→	C		D
A	D	→	C		B
B		→	D		
C		→	D		
D	→	B			B
D	→	A			A

1. The first format have in all the row of the txt file a reaction constructed in this way:

({Reactants},{Inhibitors},{Products})

Every name is separated with a comma.

2. The second format have this structure:

Reactants → Products | Inhibitors

Once the reaction system is imported, we can view the entire reaction system within the window and, if we want, we can add other reactions with the appropriate form located above the view of the reaction system. We can also save our reaction system as a txt file with the same format specified above.

The screenshot shows the MuMa Predictor window with the 'Reaction System' panel active. At the top, there are three input fields labeled 'Reactants', 'Products', and 'Inhibitors', separated by a right arrow and a vertical bar. Below these is an 'Add Reaction' button. The main area of the panel is a list of reactions in the form 'Reactants → Products | Inhibitors'. The reactions listed are:

- CycB → Cdc20
- CycA → CycA | Cdc20 Rb Ubch10
- CycA → CycA | Cdc20 Rb cdh1
- E2F → CycA | Cdc20 Rb Ubch10
- E2F → CycA | Cdc20 Rb cdh1
- create → CycB | Cdc20 cdh1
- CycD → CycD
- E2F → CycE | Rb
- create → E2F | CycA CycB Rb
- p27 → E2F | CycB Rb
- create → Rb | CycA CycB CycD CycE
- p27 → Rb | CycB CycD
- create → Ubch10 | cdh1
- Cdc20 Ubch10 → Ubch10
- CycA Ubch10 → Ubch10

At the bottom of the panel are two buttons: 'Save Reaction System' and 'Delete Selected'.

## Predictor

Predictor panel is located in the bottom part of the MuMa Predictor window.

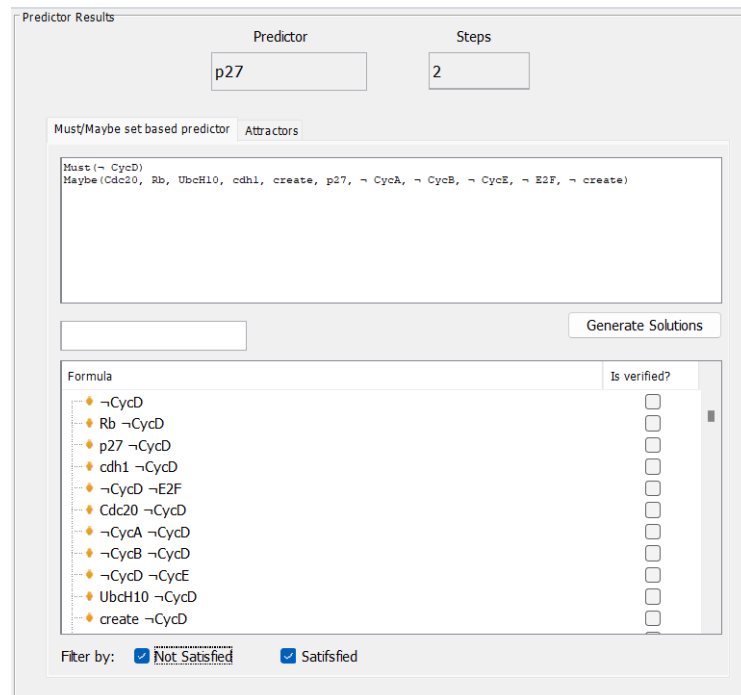
The screenshot shows the MuMa Predictor window with the 'Predictor' panel active. It contains a text input field with the text 'ex1 ex2 ex3'. To the right of the input field is a 'Steps:' label followed by a spinner box set to '1'. To the right of the spinner box is a 'Compute Predictor' button.

We can add in the predictor as many elements as we want as long as they can be produced by the reaction system. If this does not happen we would be alerted with a warning window specifying which of the elements cannot be produced by the system.

Every element written on the predictor form must be separated from a space, as shown in the figure above.

By clicking on the "Compute Predictor" button, we can see on the right part of the window the result of the computation, the Must and the Maybe set.

Finally, we can check all the solutions derived from these two sets by clicking on the button "Generate Solutions", the result will look something like the Figure below.



In the window below the Must/Maybe sets text area, we have a list of all formulas created starting from the combination of Must and Maybe sets and, for every formula, in the second column, if that formula is verified or not by the system.

In addition, as we can see in the image above, we have a filter to view only all the satisfied formula, or only not satisfied, or both of them and a formula search bar located just above the list.

Through these filters it becomes easier to be able to verify the presence of certain formula and their satisfiability.

Within the list we may encounter verified formulas represented by a folder, as in the following case:

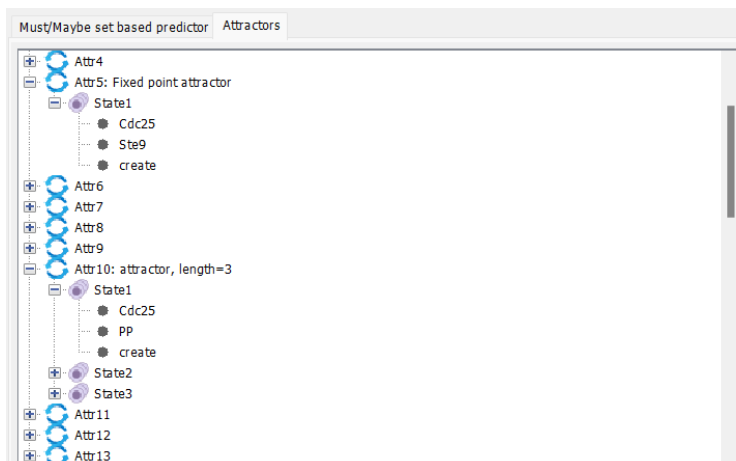
Formula	Is verified?
<div> <div></div> Rb cdh1 create ¬CycD </div>	<input checked="" type="checkbox"/>
<div> <div></div> Cdc20 Rb create ¬CycD </div>	<input checked="" type="checkbox"/>
<div> <div></div> Cdc20 create ¬CycD ¬E2F </div>	<input checked="" type="checkbox"/>
<div> <div></div> Cdc20 p27 ¬CycA ¬CycB ¬CycD </div>	<input checked="" type="checkbox"/>

We represent formulas in this way when all elements inside the folder are verified formulas containing the formula specified as name of the folder.

## Attractors

Next to the "Must/Maybe set based predictor" tab, there's another tab representing the section about attractors.

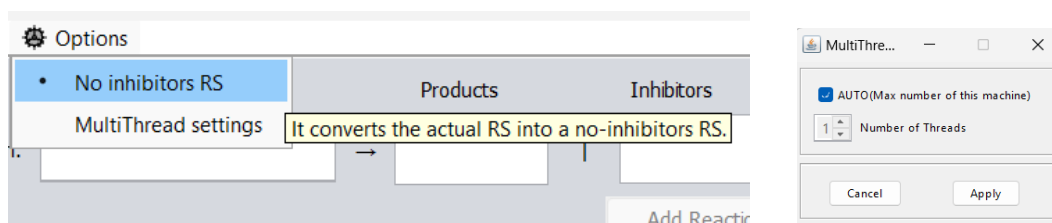
In this section, we find a list of all the attractors encountered in the computation of the predictor.



As we can see on the image above, essentially an attractor can be a fixed point, as "Attr5", or a cycle, as "Attr10".

## Options

At the top of the MuMa Predictor window, next to "file" we have a menu called "options" with two items, "no inhibitors rs" and "multithread settings".



- "No inhibitors rs"  
This is an option that you can select or not by clicking on it. If you have this option checked the actual reaction system is converted into a no-inhibitors reaction system. You can switch from one system to another simply by checking it or not.
- "Multithread settings"  
If we click on the "Multithread settings" on the "options" menu it will appear a window like the one in the image.  
By default, the tool configures the number of threads to be used for formula computation based on the maximum number of threads supported by the machine.

If we want this behavior have to check “AUTO” or uncheck it and select a desired number of threads with the spinner.