PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Dynamic Biological Modelling

a language-based approach

Alessandro Romanel

Advisor:

Prof. Corrado Priami

Università degli Studi di Trento

March 2010

Abstract

Systems biology investigates the interactions and relationships among the components of biological systems to understand how they globally work. The metaphor "cells as computations", introduced by Regev and Shapiro, opened the realm of biological modelling to concurrent languages. Their peculiar characteristics led to the development of many different bio-inspired languages that allow to abstract and study specific aspects of biological systems. In this thesis we present a language based on the process calculi paradigm and specifically designed to account for the complexity of signalling networks. We explore a new design space for bio-inspired languages, with the aim to capture in an intuitive and simple way the fundamental mechanisms governing protein-protein interactions. We develop a formal framework for modelling, simulating and analysing biological systems. An implementation of the framework is provided to enable in-silico experimentation.

Keywords
[systems biology, process calculi, stochastic simulation, protein-protein interaction, computational power]

Acknowledgements

I would like to thank all the people that, in one way or another, helped me to reach this important goal.

I thank my supervisor Corrado Priami not only for the support he gave me during these three years but also, and foremost, for motivating me, giving me the freedom to follow the research topics I liked more and permitting me to be part of the CoSBi team.

I thank the referees Pierpaolo Degano and Ian Stark for the corrections and precious suggestions which considerably helped to improve this thesis.

I want also to thank my PhD mates Lorenzo Dematté, Alida Palmisano, Roberto Larcher, Michele Forlin and Judit Zámborszky. We discussed about problems, shared ideas and had very good time together. Working with them enriched me both professionally and personally.

Thanks to Davide Prandi and Tommaso Mazza for the great time we had together in these years and for the courage they showed reviewing this thesis, providing me with useful suggestions and corrections. Thanks also to all the other colleagues and friends for the lighter and funny moments that helped me to work hard.

I thank my family for the endless patience and continuous help in my everyday life; without them I would have hardly succeeded in this work.

I also would like to thank Prof. Benjamin C. Pierce for the opportunity he gave me to visit Philadelphia and the CIS department at the University of Pennsylvania. I really enjoyed working with him on new ideas and approaches in process calculi.

Philadelphia is also the place where I met Lorena, to which my final and special thank is devoted. Thanks for your love, for your patience and for having been always present.

Contents

List of Tables

List of Figures

An answer is always the stretch of road that's behind you.
Only a question can point the way forward.

Jostein Gaarder

Chapter 1

Introduction

1.1 The context

Biology is the science of life and embraces many different disciplines and areas. One of these is *molecular biology*, which aims at studying biology on a molecular level. Molecular biology deals with the formation, structure, and function of macromolecules essential to life (e.g., DNA, RNA and proteins) and hence overlaps with other areas of biology and chemistry, particularly genetics and biochemistry.

In the last thirty years, the constant advances in technologies and experimental techniques in molecular biology led to a massive increase in the data available to scientists. The application of *informatics* to store, manage and analyse this enormous amount of data results in what we call today *bioinformatics*. This first convergence between computing and biology conducted scientists to a deeper comprehension of the basic mechanisms governing living organisms, leading to unthinkable achievements like the *Human Genome Project* that sanctioned the beginning of the *post-genomic* era.

However, the advent of bioinformatics happened under the *reductionist* perspective, and hence is impregnated of its methods, concepts and obviously its limits. The reductionist method studies biological systems by analysing in detail the structure of their individual components (e.g., molecules and proteins) and from this determining the connections between the different components. The method is based on the assumption that the single components contain enough information to explain the complexity of a whole system. Although effective, this assumption represent one of the main limits of the reductionist approach [106] because even having a complete *network* of connections between the components, still we could not understand the overall behaviour of a system. Biological systems are indeed extremely complex and can have emergent properties and behaviours that cannot be explained or predicted by studying only the structure of their individual components. As Kitano points out in [57] a complete system-level understanding requires

a shift in the notion of *what to look for* in biology. While the understanding of genes and proteins continues to be important, the focus is on understanding the system's structure and dynamics. Although subverting the traditional reductionist approach, this system-level perspective is not against it but throws the basis for a new paradigm that in the last years has been identified with *systems biology*. An explicit and precise definition of systems biology is difficult to give [57, 56, 110, 82, 52], or even still impossible, but we can capture its main principles and goals in the words of Kirschner [56]:

> *"I would simply say that systems biology is the study of the behaviour of complex biological organization and processes in terms of the molecular constituents. It is built on molecular biology in its special concern for information transfer, on physiology for its special concern with adaptive states of the cell and organism, on developmental biology for the importance of defining a succession of physiological states in that process, and on evolutionary biology and ecology for the appreciation that all aspects of the organism are products of selection, a selection we rarely understand on a molecular level. Systems biology attempts all of this through quantitative measurement, modelling, reconstruction, and theory."*

It is clear how systems biology is strongly inter-disciplinary and calls for the integration and convergence of many different sciences and disciplines. Its domain indeed spreads from several branches of biology to more distant disciplines, such as physics, mathematics, statistics and informatics. The main role of these latter distant disciplines is to devise adequate formal tools and concepts to describe and model efficiently biological systems, analyse their properties, and reproduce and predict their behaviour through computer-based simulations (e.g., [105, 66, 68, 70]), with the ultimate and challenging goal of delineating the basis for a foundation theory of systems biology.

Systems biology gave hence the opportunity to propel a further step into the convergence between informatics and biology already started years ago with bioinformatics, resulting in a moment of incredible scientific ferment where various computational approaches have been proposed and equipped with supporting software tools (e.g., [96, 79, 40, 46, 49]). A computational model differs from a traditional mathematical one, because it is executable and not just simply solvable [37]. Execution means that we can predict/describe the flow of control between species and reactions (e.g., not only the time, but also the causality relation among the events that constitute the history of the dynamics of the model).

In particular, the abstraction of molecules as parallel, interacting computational entities, introduced in [98], opened the realm of concurrency theory to an unexpected field

of application. This approach stresses the importance of concurrency and interaction between molecules as the main tools that drive the execution of biological processes and is at the basis of emergent schools of thought like [92, 62]. Furthermore, it gave the opportunity to reuse several theoretical results and analysis techniques developed for concurrent and distributed systems into the realm of biological system, with the primary goal of bringing new insights on their functioning.

One of the first examples of application of these concepts can be found in [96], where a framework for modelling biological processes is developed around the π-calculus [73]. The π-calculus is probably the most famous representative of the *process calculi* family, invented to specify and study the behaviour of concurrent software systems through an hierarchy of formal specifications written in the same calculus more and more concrete till the implementation on a specific architecture [71, 50]. The peculiarity of describing a system at different levels of abstraction using the same calculus and the automatic generation of intermediate states by the semantics rather than the need of specifying all of them since the beginning, make process calculi particularly suitable to model biological systems. Process calculi are provided also with stochastic variants, primarily developed as tools for analysing the performances of concurrent systems [91, 48]. Models constructed with these stochastic calculi are usually Continuous Time Markov Chains (CTMC), the same kind of stochastic processes used to directly simulate systems of biochemical reactions under certain physical considerations [43]. This further analogy allowed to use stochastic simulation by means of Gillespie's algorithm [42] also in the context of process calculi, denoting them definitely as a powerful abstraction for the representation of biological systems.

During the last years a number of process calculi have been developed for applications in systems biology (e.g., [10, 97, 26, 90, 94, 18]). On top of these process calculi several languages have been defined and frameworks for analysis and stochastic simulation have been implemented (e.g., [85, 16, 76, 1]). Some of these new *bio-inspired process calculi* differ from classical process calculi because they are devised from the beginning for biology and aim at overcoming some expressivity limitations by developing new conceptual tools.

Motivated by the intention to take some significant step in this direction we started three years ago by exploring a new design space for process calculi with the aim to capture in an intuitive and simple way the fundamental mechanisms of biological modelling.

1.2 The problem

Modelling is an essential aspect in systems biology, as well as in many other scientific activities. Quoting Von Neumann *"The sciences do not try to explain, they hardly even*

try to interpret, they mainly make models".

A model is a partial representation of reality and its main aim is to show which are the necessary and sufficient characteristics of a system that allow to understand it. In [82], Noble says that the power of a model lies in identifying what is essential, whereas a complete representation would live us just as wise, or as ignorant, as before. Being the process calculi approach strictly related with the modelling activity, one the first question we have to ask ourselves when developing a new modelling language is: *What do we expect from our language?*

The initial aim of this thesis work was to design a language, based on the process calculi paradigm, for modelling and study *signalling networks*. The importance of being able to represent in an effective way the relevant mechanistic details of signal transduction systems is stressed also in [57] where Kitano states that *"The most feasible application of systems biology research is to create a detailed model of cell regulation, focused on particular signal-transduction cascades and molecules to provide system-level insights into mechanism-based drug discovery".*

The main problem we have to deal with when we model signalling networks is the complexity of protein-protein interactions, i.e., the enormous number of possible post-translational covalent modifications of proteins and the formation of protein complexes. This problem is also called *combinatorial explosion* problem. Combinatorial explosion usually arises because: proteins sometimes exist in different species with common functionalities; proteins can present multiple conformational states (e.g., different combinations in sites phosphorylation) and still present the same functionalities; proteins can be part of different complexes and still present the same functionalities. In general, the number of reactions of systems presenting these characteristics grows exponentially with the number of protein species, protein domains and binding capabilities.

1.3 The contribution

The starting point of our work is Beta-binders [94, 30, 95]. In this formalism processes are encapsulated into boxes with typed interfaces. Types represent the interaction capabilities of the boxes. Beta-binders aims at enabling non-determinism of communication by introducing the concept of *compatibility* [90], a notion that extends the *key-lock* notion of complementarity between actions and co-actions typical of process calculi, i.e., the precise matching between an input and an output over a given channel is always required. The formalism is also provided with *join* and *split* operations, i.e. parametric rules that drive the merging and splitting of boxes depending on their structure. In Beta-binders the description of such operations is left as undetermined as possible, with the goal to

accommodate possible distinct instances of the same macro-behaviour.

Recognizing that Beta-binders improves the abstract representation of biological interactions and considering this improvement fundamentally important for a realistic and effective representation of signalling pathways, we selected it as a basis for our work. Although maintaining the basic principles characterizing the soul of Beta-binders, the result of this work deviates substantially from it, hence justifying a different name for our new language, the BlenX language. BlenX inherits from Beta-binders the basic abstraction of biological entity (a process encapsulated into a box with interaction capabilities) and the notion of *communication by compatibility*. However, many new features are introduced. In BlenX we introduce the notion of *event*, a reformulation of the join and split operations of Beta-binders. Moreover, we introduce *complexes*. A complex is a combination of two or more proteins and molecules attached together along compatible surfaces. In BlenX complexes are represented as graphs with boxes as nodes. Although similar to [26, 13], our interpretation of complexes is substantially different in the way complexes can be formed and modified. By extending the notion of compatibility, indeed, we enable also at the level of complexes generation the kind of non-determinism already characterizing the communication approach instantiated by Beta-binders.

The coarse-grained modelling level allowed by BlenX is also characterized by the presence of a general mechanism for encoding high-level operations as sequences of low-level ones. The language is indeed enriched with priorities [19], a scheduling scheme that permits to compose sequences of low-level operations atomically. Priorities are essential to implement scenarios in which for example we want to describe atomically sequences of modifications happening on a whole complex.

It is clear how an interesting question is whether and how those modifications and new features affect the ability of BlenX to act as a computational device. Some first insights to this question are given in this thesis. We show indeed that for a core subset of the language termination is decidable. Moreover, we prove that by adding either priorities or events to this core language, we obtain Turing equivalent languages.

The direct application of BlenX to biological modelling is accomplished by the presentation of a stochastic variant of the language. Following [30], also in our extension the speed of actions is provided with systems specifications assuming as underlying as underlying theory the law of mass action. However, reducing a biological system model to elementary steps is most of the times very complicated and often impracticable and hence there are cases in which law of mass action is not applicable. Indeed, it is common to find in models reactions that are abstractions of scenarios whose details are unknown. For these reasons, we incorporate in the language also the possibility to associate to certain classes of actions *generic kinetic laws*.

All the theoretical framework presented in this thesis is accompanied with the implementation of a framework for validation and simulation purposes: The Beta Workbench (BWB). It is a collection of tools built on top of BlenX to model and simulate biological systems.

Having in our hand this framework for the *in-silico* experimentation we first investigate the modelling of signalling pathways, proposing general design patterns for the definition of them. Then, we propose a framework for simulating the evolution of protein-protein interaction networks where evolution proceeds through selection acting on the variance generated by random mutation events, and individuals replicate in proportion to their performance, referred to as fitness. Finally, we investigate the modelling of self-assembly, providing general design patterns for modelling non-trivial structures like filaments, trees and symmetric rings.

It is important to underline that the BlenX language is in continuous evolution and several extensions are currently subject of other parallel works. Here we present only the subset concerning my thesis work, the *kernel* part of BlenX, representing its heart.

1.4 Structure of the thesis

- **Chapter 1** is this introduction;
- **Chapter 2** gives basic preliminaries about biological systems, introduces process calculi and presents an overview of the main process calculi applied or developed for systems biology;
- **Chapter 3** presents informally the BlenX language, providing an overview of its main features;
- **Chapter 4** gives a formal presentation of the BlenX language and its operational semantics, and explores some theoretical properties regarding structural congruence and its decidability and complexity;
- **Chapter 5** shows that for a core subset of the language termination is decidable. Moreover, it shows that by adding either global priorities or events to this core language, we obtain Turing equivalent languages;
- **Chapter 6** describes sBlenX, the stochastic extension of BlenX. Moreover, it presents a stochastic abstract machine for sBlenX that, using results presented in Chapter 4, compresses the system state space;
- **Chapter 7** presents the Beta Workbench, a framework to run sBlenX models, and describes the concrete syntax of the language;

- **Chapter 8** investigates the modelling of signalling networks, providing models and design patterns (of increasing complexity) of the well know MAPK cascade signalling network;
- **Chapter 9** describes a framework that allows the study of signalling networks evolution. A case study based on the MAPK cascade is presented.
- **Chapter 10** investigates the modelling of self-assembly in BlenX, providing models and design patterns for the formation of non-trivial structures like filaments, trees and symmetric rings;
- **Chapter 11** concludes and summarises the thesis.

1.5 Related publications

Book Chapters

L. Dematté, R. Larcher, A. Palmisano, C. Priami, A. Romanel. **Programming Biology in BlenX**. *In Systems Biology for Signaling Networks*, Springer, 2010.

L. Dematté, C. Priami, A. Romanel. **The BlenX Language: A Tutorial**. *In SFM 2008*, LNCS 5016:313-365, Springer, 2008.

International journals

R. Larcher, C. Priami, A. Romanel. **Modelling self-assembly in BlenX**. *Transactions on Computational Systems Biology XII*, LNBI 5945:163-198, Springer, 2010.

A. Romanel and C. Priami. **On the Computational Power of BlenX**. *Theoretical Computer Science* 411(2):542-565, Elsevier, 2010.

L. Dematté, C. Priami, A. Romanel, O. Soyer. **Evolving BlenX programs to simulate the evolution of biological networks**. *Theoretical Computer Science* 408(1):83-96, 2008.

L. Dematté, C. Priami, A. Romanel. **The Beta Workbench: a computational tool to study the dynamics of biological systems**. *Briefings in Bioinformatics* 9(5):437-449, 2008.

A. Romanel and C. Priami. **On the Decidability and Complexity of the Structural**

Congruence for Beta-binders. *Theoretical Computer Science* 404(1-2):156-169, 2008.

L. Dematté, C. Priami, A. Romanel. **Modelling and simulation of biological processes in BlenX**. *SIGMETRICS Performance Evaluation Review* 35:32-39, 2008.

International conferences and workshops

L. Dematté, C. Priami, A. Romanel, O. Soyer. **A Formal and Integrated Framework to Simulate Evolution of Biological Pathways**. *In Proceedings of Computational Methods in Systems Biology (CMSB)*, Springer LNCS, 4695:106-120, 2007.

L. Dematté, C. Priami, A. Romanel. **BetaWB: modelling and simulating biological processes**. *In Proceedings of Summer Computer Simulation Conference (SCSC)*, 777-784, 2007.

C. Priami, A. Romanel. **The Decidability of the Structural Congruence for Beta-binders**. *In Proceedings of Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC), 2006*, ENTCS 171(2):155-170, 2007.

Chapter 2

Preliminaries

2.1 Biological systems

A cell is a microscopic structure containing nuclear and cytoplasmic material enclosed by a semipermeable membrane and, in plants, a cell wall. It is meant to be the basic structural unit of all organisms.

From a topological perspective, a cell can be seen as an architecture of physical membrane bounded locations (i.e., compartments whose boundary are determined by membranes). Each compartment is a container of substances and the role it plays in regulating cell life depends on the substances that it contains. The content of compartments changes over time as a result of cell regulation. Consequently, substances continuously migrate from one location of the cell to another in response to internal and/or external stimuli.

An eukaryotic cell is made up of a plasma membrane that surrounds the internal subcompartments, the organelles. The interior, or lumen, of each organelle is enclosed by one or more membranes and contains a unique set of proteins that characterizes the functions of the organelle together with the membrane-bound proteins. The largest organelle is the nucleus. The part outside the nucleus is called cytoplasm and it contains all the other organelles. The aqueous part of the cytoplasm is called cytosol and it contains a large number and variety of substances. These substances continuously produces a chemical activity that allows a cell to grow, multiply, and do its daily tasks.

The main categories of cell substances are listed in the following.

Nucleic acids: the most important nucleic acid is surely the *Deoxyribonucleic Acid*, the DNA. DNA codifies the blueprint for the complete biochemical machinery of the organism that carries it. It has a stable molecular structure and is meant as long-term storage of the complete heritage of an organism. The other major nucleic acid present in cells is the *Ribonucleic Acid* (RNA), which performs a multitude of cellular tasks in

gene expression and reaction catalysis. The two main cellular processes involving DNA are replication and transcription. Replication is the process of making a complete copy of the DNA sequence and occurs whenever the cell divides. Transcriptions concerns the decodification of the information that is stored in the DNA sequence: DNA is transcribed into RNA while RNA is translated into proteins.

Proteins: they are essential parts of organisms and participate in almost every process within cells. They perform several different tasks, like reaction catalysis and regulation, transmission of signals, gene expression and regulation, membrane transport. Proteins are polymers of amino acids, called polypeptides. The sequence of amino acids is usually referred to as the protein's primary structure and determines the protein's three dimensional structure and shape, which is the key of its functioning. This structure depends on how the primary sequence folds around itself. Although the process of folding is still poorly understood, several studies revealed some local regularities in the folding pattern, called protein's secondary structure. The secondary structure gives rise to elements called *structural motifs* (or *domains*) that allow the proteins to interact with other molecules. Besides local foldings, the overall three dimensional structure of a protein is called tertiary structure. When proteins complex with other proteins or molecules, the obtained structure is referred to as the quaternary structure. The way in which proteins function is extremely interesting. When they interact or combine with other proteins or molecules, they maintain their overall structure (their identity) but change their shape. The different three dimensional shapes that proteins can assume are also called *conformational states*. For example, there are enzymes with active and inactive states; when active, these enzymes can bind to substrates and catalyse the corresponding reaction. The conformational states and the binding capabilities of a protein define its behaviour and function.

Metabolites: a metabolite is an intermediate product of the metabolism. Metabolites can be fuels and signalling molecules, cellular building blocks, nucleotides, carbohydrates, lipids, hormones, vitamins, and various other molecules concerned with the vast range of cellular tasks. Usually, metabolites can perform very specific tasks and their structure and chemical properties are relatively simple if compared, for example, to proteins. We can think as if they have single identities and states. Indeed, if a metabolite reacts it becomes a different metabolite with a different identity and function.

2.1.1 Biochemical reactions

Cell substances interact continuously in thousands of different ways. Biochemists see these systems as networks of coupled chemical reactions.

An example of reaction is:

$$2A + B \rightarrow 3C$$

Molecules A and B are *reactants*, while C is the *product*. When in a system this reaction happens, two molecules A and a molecule B will vanish, and three molecules C will appear. Values 2 and 3 are known as *stoichiometries*, and they are usually natural numbers. A molecule can be both consumed and produced in a single reaction. In particular, if a chemical species occurs on both the left and the right hand side, it is referred to as *modifier*. A reaction that can happen in both directions is known as reversible. Reversible reactions are quit common in biology. They are written explicitly adding a reverse arrow for the backward reaction:

$$2A + B \rightleftharpoons 3C$$

This notation is simply a shorthand for the two separate reaction processes taking place.

Chemical kinetics is concerned with the time-evolution of a reaction system specified by a set of coupled chemical reactions. In particular, it is concerned with the system behaviour away from equilibrium. Although the reaction equations capture the key interactions between substances, on their own they are not enough to determine the full system dynamics. The *rates* at which each of the reactions occurs, together with the initial concentration of the reacting molecules, are the missing information. The rate of a reaction is a measure of how the concentration of the involved substances changes with time. To better understand the crucial concept relating to the dependency between the rate and the concentration, let us consider one single stage reaction in which one molecule A reacts with a molecule B, giving one molecule of C.

$$A + B \rightarrow C \tag{2.1}$$

According to the collision theory, C is produced with a rate that is proportional to the hits frequency of A and B. Let us imagine to have a certain number of molecules B and only one molecule A in a container. The frequency of their hits is proportional to the number of molecules B. Let us suppose now to introduce another molecule A. It results that the number of molecules C doubles since the number of molecules A doubles. In other words, the frequency at which molecules hit and consequently the rate at which molecules collide, is proportional to the concentration of both A and B. This relation is captured by the *law of mass action*, which states that *the rate of a chemical reaction is directly proportional to the product of the effective concentrations of each participating*

molecule. In our case we hence have:

$$rate = \frac{d[C]}{dt} = k[A][B]$$

where k is the *basal rate* and $[A]$, $[B]$ and $[C]$ denote the concentration of molecules A, B and C, respectively. In general, the rate is proportional to the concentration of the reactants involved raised to the power of their stoichiometry. For example, given the homogeneous[1] reaction:

$$nA + mB \rightarrow P \tag{2.2}$$

the corresponding relation between the rate and the concentration is:

$$rate = \frac{dP}{dt} = k[A]^n[B]^m \tag{2.3}$$

and the global order of the reaction corresponds to the sum of n and m.

2.1.2 Enzymatic catalysis

Enzymes play the important role of catalysing those biochemical reactions that make life possible. Basically, enzymes bind to one or more ligands, called *substrates*, and convert them into one or more chemically modified *products*. A simple enzymatic reaction is considered in Fig 2.1. This mechanism is also called single substrate catalysed reaction. The enzyme E and the substrate S encounter to form the enzyme-substrate intermediate ES. This reaction can be reversed, but the formation of ES is favoured when many substrates molecules are available. When S is bound, E sends a signal enabling the modification of the substrate S into product P. Finally, the product P is released, and the enzyme E regenerated.

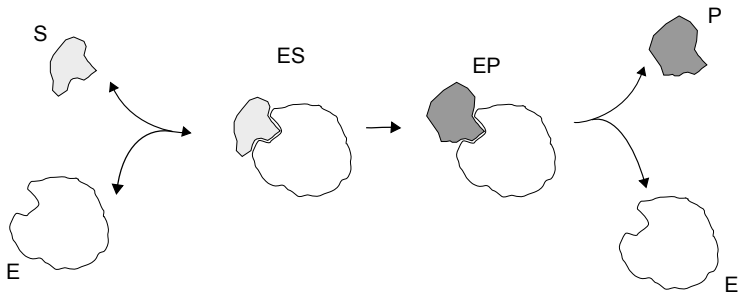


Figure 2.1: Single substrate catalysed reaction.

[1] A reaction is said to be homogeneus if it does not summarize any hidden or intermediate reaction.

More concisely we can write this reaction as:

$$E + S \underset{r_2}{\overset{r_1}{\rightleftharpoons}} ES \xrightarrow{r_3} EP \xrightarrow{r_4} E + P$$

Note that only the first reaction is considered reversible. Indeed, many enzymatically catalysed reactions are irreversible, meaning that under normal conditions the backward reaction can be neglected.

Enzymatic catalysis is usually described by the Michelis-Menten kinetics. In this setting the single substrate catalysed reaction is represented using the formulation:

$$E + S \underset{r_2}{\overset{r_1}{\rightleftharpoons}} ES \xrightarrow{r_3} E + P$$

that abstracts from the presence of the intermediate EP. The Michaelis-Menten equation describes the relationship between the rate of substrate conversion by an enzyme to the concentration of the substrate:

$$r = \frac{V_{max} \times [S]}{K_m + [S]}$$

where,

$$K_m = \frac{r_2 + r_3}{r_1} \quad \text{and} \quad V_{max} = k_3 \times [E]_t$$

In this equation, r is the rate of conversion, V_{max} is the maximum rate of conversion, $[S]$ is the substrate concentration, and K_m is the Michaelis constant. The Michaelis constant is equivalent to the substrate concentration at which the rate of conversion is half of V_{max}. K_m approximates the affinity of enzyme for the substrate. A small K_m indicates high affinity, and a substrate with a smaller K_m will approach V_{max} more quickly. Very high $[S]$ values are required to approach V_{max}, which is reached only when $[S]$ is high enough to saturate the enzyme. While the derivation is not shown in this discussion, V_{max} is equivalent to the product of the catalyst rate constant (k_3) and the (total) concentration of the enzyme.

Inhibitors are molecules that decrease the speed of enzymatic reactions. When an inhibitor binds to an enzyme it can prevent a substrate from entering the active domain of the enzyme or directly block the catalytic activity of the enzyme. The action of an inhibitor can be either reversible or irreversible. A particular type of reversible inhibition, named *competitive inhibition*, is sketched in Fig. 2.2. In this mechanism the substrate S and the inhibitor I compete for the active domain of the enzyme E. Often, competitive inhibitors strongly resemble the structure of the real substrate of the enzyme. Therefore, the inhibitor occupies the active domain of the enzyme, and it prevents normal substrate from binding and being catalysed. The biochemical representation of competitive inhibi-

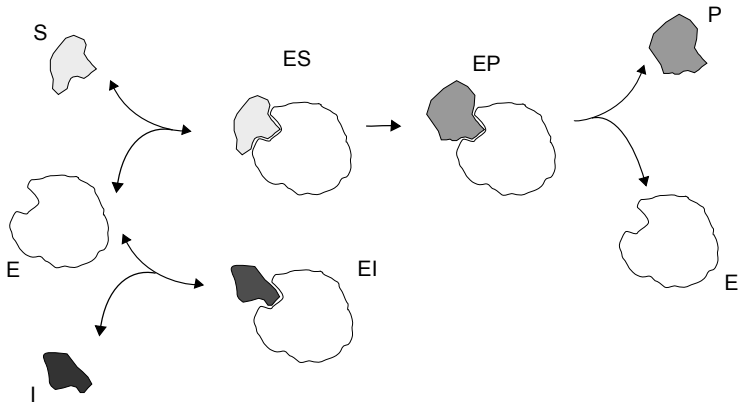


Figure 2.2: Competitive inhibition.

tion is:

$$E + S \underset{r_2}{\overset{r_1}{\rightleftharpoons}} ES \xrightarrow{r_3} E + P$$
$$E + I \underset{r_6}{\overset{r_5}{\rightleftharpoons}} EI$$

The derivation of the Michaelis-Menten equation is the same as for the uninhibited mechanism except for an additional term in the expression that accounts for the total enzyme concentration and for the new intermediate EI. The derived equation is:

$$r = \frac{V_{max} \times [S]}{K_m + [S] + K_m \times \dfrac{r_6}{r_5} \times [I]}$$

More complicated forms of enzymatic reactions are given by multi-substrate catalysed reactions, where more substrates are involved in the process. A large group of these reactions are the *bisubstrate* reactions, which have two substrates. For bisubstrate reactions three basic reaction mechanisms have been discerned: the *ordered sequential mechanism*, the *random sequential mechanism* and the *ping pong mechanism*. As an example, in the first mechanism (see Fig. 2.3) the enzyme first binds both substrates and then proceeds to the actual catalytic reaction step; the substrates can only bind in a given order.

2.1.3 Signalling networks

An important category of biological systems is the one involved in signal transduction. Signal transduction networks, or shortly *signalling networks*, constitute the communication lines of a cell. A signalling network is any biological process that converts one kind of signal or stimulus into another. In general, a signalling network results in a composition or cascade of biochemical reactions that are carried out by proteins and linked through

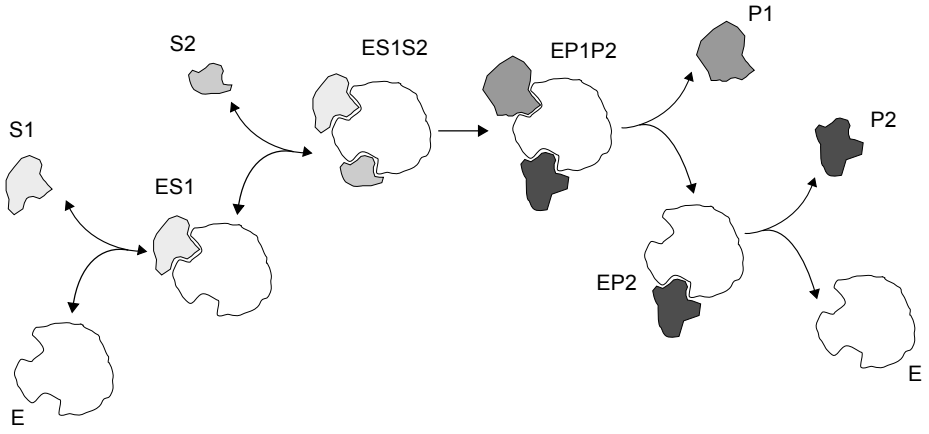


Figure 2.3: Ordered sequential bisubstrate enzymatic reaction.

second messengers. Biological signal transduction allows a cell or organism to sense its environment and react accordingly. Typically, a signalling network has one (or more) inputs, represented by any environmental stimulus, and one (or more) outputs, represented by an active protein. The types of signals that are transmitted are numerous: synaptic signals transmitted by neurons, signals indicating the presence of harmful substances, signals indicating to single cells that another cell is ready to mate, and signals transmitted through hormones that convey a whole range of cellular instructions. An example of signalling network is the MAPK cascade.

The MAPK cascade: The mitogen-activated protein kinase cascade (MAPK cascade) is a series of three protein kinases[2] which is responsible for the cell response to some growth factors. In [51], a model of the MAPK cascade was presented and analysed using Ordinary Differential Equations (ODEs); the cascade was shown to perform the function of an ultra-sensitive switch and the response curves were shown to be steeply sigmoidal. Fig. 2.4 presents schematically MAPK cascade as described in [51]. *KKK* denotes *MAPKKK*, *KK* denotes *MAPKK* and *K* denotes *MAPK*. The signal *E1* transforms *KKK* to *KKKp*, which in turn transforms *KK* to *KKp* to *KKpp*, which in turn transforms *K* to *Kp* to *Kpp*. In particular, when an input *E1* is added, the output of *Kpp* increases rapidly. The transformations in the reverse direction are the result of the signal *E2*, the *KKpase* and the *Kpase*. In particular, by removing the signal *E1*, the output level of *Kpp* reverts back to zero. The formal model in [51] is built using 30 single reactions (see Tab. 2.1). The set of reactions is used to derive a system of 25 mathematical equations (18 ODEs plus 7 conservation equations).

[2]a kinase is a type of enzyme that transfers phosphate groups from high-energy donor molecules, such as *ATP*, to specific substrates.

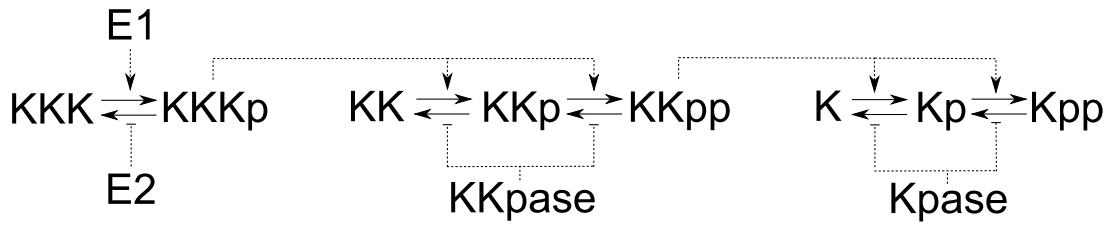


Figure 2.4: MAPK cascade.

$$E_1 + KKK \underset{d_1}{\overset{a_1}{\rightleftharpoons}} E_1 : KKK \xrightarrow{k_1} E_1 + KKKp$$
$$E_2 + KKKp \underset{d_2}{\overset{a_2}{\rightleftharpoons}} E_2 : KKKp \xrightarrow{k_2} E_2 + KKK$$
$$KKKp + KK \underset{d_3}{\overset{a_3}{\rightleftharpoons}} KKKp : KK \xrightarrow{k_3} KKKp + KKp$$
$$P_2 + KKp \underset{d_4}{\overset{a_4}{\rightleftharpoons}} P_2 : KKp \xrightarrow{k_4} P_2 + KK$$
$$KKKp + KKp \underset{d_5}{\overset{a_5}{\rightleftharpoons}} KKKp : KKp \xrightarrow{k_5} KKKp + KKpp$$
$$P_2 + KKpp \underset{d_6}{\overset{a_6}{\rightleftharpoons}} P_2 : KKpp \xrightarrow{k_6} P_2 + KKp$$
$$KKpp + K \underset{d_7}{\overset{a_7}{\rightleftharpoons}} KKpp : K \xrightarrow{k_7} KKpp + Kp$$
$$P_1 + Kp \underset{d_8}{\overset{a_8}{\rightleftharpoons}} P_1 : Kp \xrightarrow{k_8} P_1 + K$$
$$KKpp + Kp \underset{d_9}{\overset{a_9}{\rightleftharpoons}} KKpp : Kp \xrightarrow{k_9} KKp + Kpp$$
$$P_1 + Kpp \underset{d_{10}}{\overset{a_{10}}{\rightleftharpoons}} P_1 : Kpp \xrightarrow{k_{10}} P_1 + Kp$$

Table 2.1: MAPK cascade reaction set.

2.2 Stochastic simulation

Given a set of coupled chemical reactions describing a biological system, one of the most immediate and natural analyses that can be performed on it is a *simulation*. The term simulation is generally used to indicate the calculation of the system's dynamics over time, given an initial specific system configuration; for biological systems the initial configuration corresponds usually to the initial concentration of molecules.

Biological systems can be simulated in different ways using different algorithms depending on the assumptions made about the underlying kinetics. Once the kinetics have been specified, these systems can be used directly to construct full dynamic simulations of the system behaviour on a computer.

Usually, given a set of coupled chemical reactions, a set of ODEs is derived. The usual assumption made for these models is that the system is well-stirred and at the ther-

modynamic limit (i.e., system's volume and molecules quantities approach infinite, while keeping their ratio constant). Given these assumptions, ODEs models are *deterministic*: given an initial configuration, their dynamics is univocally determined. However, the thermodynamic limit cannot be always assumed, because in many systems some molecules quantity can be very low. In these cases, microscopic random effects arise, making the system naturally *stochastic*. Chemical stochastic systems are usually represented by a chemical master equation (CME) that describes the time evolution of the probability distribution of the discrete molecule quantities (expressed by natural numbers). This evolution is a Continuous Time Markov Chain (CTMC), of which any possible realisations can be generated through Monte Carlo sampling methods. The most famous of these methods for coupled chemical reactions is the SSA algorithm of Gillespie [42, 43].

2.2.1 Gillespie algorithm

Gillespie designed an efficient way to simulate a trajectory of a set of coupled chemical reactions. The algorithm he proposed is exactly consistent with the underlying principles behind the CME [42, 43]; it simulates a jump Markov process and is based on the assumption that two events take place at the same time with zero probability.

In [43] a generic system of coupled chemical reactions is described as a vector $S = (S_1, \cdots, S_n)$, representing a well-stirred mixture of $n \geq 1$ interacting molecular species, confined in constant volume Ω and containing $M \geq 1$ reaction channels (chemical reactions described by stoichiometric equations) $R = (R_1, \cdots, R_m)$. The dynamics of a system is specified by a vector of random variables $\mathbf{X}(t) = (X_1(t), \cdots, X_n(t))$, where $X_i(t)$ is the population of the species S_i present in the system at time t. Given $\mathbf{X}(t) = x$, for each reaction channel R_j, there exists a function $a_j(x)$, called *propensity function*, defined as the probability that a reaction R_j occurs in the next infinitesimal time interval $[t, t + d\tau)$. The function $a_j(x)$ is defined as $a_j(x) = c_j \times h_j(x)$ and its result is denoted with *propensity value*. Constant c_j, for the corresponding channel R_j, represents the *specific probability rate constant* and consists of a base rate determined empirically and dependent on the specific type of reaction plus some environmental conditions. Function $h_j(x)$, instead, returns the number of all the possible distinct reactions that can occur between molecules on channel R_j, when the system is in state x. Vectors S, R, $\mathbf{X}(t)$ together with the function $a_j(x)$, for each $j \in \{1, \cdots, m\}$, completely specify the system at time t.

Gillespie proposes two mathematically equivalent methods: *the direct method* (DM) and the *first reaction method* (FRM). Gibson and Bruck [41] later achieved a significant reduction in complexity with respect to the Gillespie algorithms, however the underlying principles remain the same. Many other variants of the algorithm can be found in literature [104].

First Reaction Method (FRM)

At each step a random putative reaction time is calculated for each reaction and the one with the shortest time is chosen and executed.

1. *Set the initial number of molecules for each species in* S
2. *Set* $t \leftarrow 0$
3. *Calculate the propensity value* a_i *for each* $i \in \{1, \cdots, m\}$
4. *For each* $i \in \{1, \cdots, m\}$ *generate a putative time* τ_i *in accordance with an exponential distribution of parameter* a_i
5. *Let* τ_μ *and* μ *be the fastest time* τ_j *and the corresponding reaction channel* R_j
6. *Update the number of molecules to reflect the execution of* μ
7. *Set* $t \leftarrow t + \tau_\mu$
8. *Return to step* 3

Direct Method (DM)

Two separate calculations are considered: the selection of the reaction and the evaluation of the time. The dynamics of the system is described by the following density function: (1) $P(j \mid x) = a_j(x)/\sum_{i=1}^{m} a_i(x)$, that is the probability that the next reaction is on channel j; (2) $P(\tau \mid x) = (\sum_{i=1}^{m} a_i(x))e^{-\tau(\sum_{i=1}^{m} a_i(x))}$. The pseudo-code of the algorithm is defined as follows:

1. *Set the initial number of molecules for each species in* S
2. *Set* $t \leftarrow 0$
3. *Calculate the propensity value* a_i *for each* $i \in \{1, \cdots, m\}$
4. *Choose reaction channel* μ *in accordance with distribution* (1)
5. *Choose time* τ *in accordance with distribution* (2)
6. *Update the number of molecules to reflect the execution of* μ
7. *Set* $t \leftarrow t + \tau$
8. *Return to step* 3

Next Reaction Method (NRM)

The Next Reaction Method [41] of Gibson and Bruck builds on the FRM to achieve a significant improvement in efficiency for typical biochemically reacting systems. An improvement in complexity from $O(m)$ to $O(log\ m)$ of an individual reaction step is gained, where m is the number of reaction channels in the system. The key modifications are the following:

- *Reaction dependency graph*: By using a reaction dependency graph, during the simulation the algorithm needs only to update those propensities which it knows could have changed. Reaction dependency in biological pathways is typically much less than m and hence the reaction dependency graph is *sparse*;

- *Indexed priority queue*: Reaction times are arranged in a indexed priority queue [21]. The root holds the reaction with the faster time and the reactions further from the root always have longer times than those closer. Reactions at the same height are not ordered. The fastest reaction is selected in $O(1)$. When the reaction with the root time is executed, its propensity and those of the set of dependent reactions changes, along with the corresponding times in the priority queue. Since it is not necessary to maintain a horizontal ordering, the authors provide an algorithm which applies pairwise exchanges of reaction times at adjacent heights in the tree to maintain the tree's invariant property. This has complexity proportional to the number of reaction times that change multiplied by the height of the tree. Since it has already been assumed that the average reaction dependency of the system is sparse, the overall complexity of selecting a reaction becomes $O(log\ m)$.

Another minor advantage is obtained by considering absolute times. By considering reaction times relative to t_0, thus absolute time, the NRM is able to re-use the putative times of reactions whose propensities have been changed by the firing of the selected reaction. In this way the NRM needs only to generate one random number per simulation step, hence reducing the computational cost related with random numbers generation.

2.3 Process calculi in systems biology

Several approaches have been developed and used to model and study complex interaction mechanisms in biological systems, mainly based on mathematical modelling, which generally takes the form of a system of ordinary differential equations (ODEs) and for which ODE solvers with various interfaces are available.

Besides classical mathematical approaches, other research lines focus in interpreting and describing systems' behaviour by relying on *computational modelling*. A computational model differs from a mathematical one, because it is executable and not just simply solvable [37]. Execution means that we can predict/describe the flow of control between molecules and reactions (e.g., not only the time, but also the causality relation among the events that constitute the history of the dynamics of the model). In other words, the computational modelling interpretation is similar to programming, the step by step behaviour of a system, rather than describing only the outcome of the system or scripting some code

to solve mathematical formulations of problems. Various computational approaches have been proposed and equipped with supporting software tools (e.g., boolean networks [55], Petri nets [46], Bayesian networks [40], graphical gaussian models [80], process calculi [96], rule-based modelling [49]).

We consider in this thesis the process calculi paradigm, that we believe can provide an interesting abstraction to an executable philosophy of computational modelling. Process calculi indeed have the ability of handling concurrency, execution causality, nondeterminism, stochastic behaviour and cooperation/competition for resources that are usual features of computational approaches.

2.3.1 Overview of process calculi

Starting from the *Calculus of Communicating Systems* [71] (CCS), process calculi have been defined to provide us with formal specifications of concurrent systems, i.e., computational entities executing their tasks in parallel and able to synchronise. The model of a system S is typically given as a term that defines the possible behaviours of the various components of S. Calculi are equipped with syntax-driven rules, the so-called operational semantics [88]. These rules allow to infer the possible future of the system under analysis and can be automatically implemented. For instance, they can specify that a certain system P evolves into system Q, written $P \to Q$. The basic entities of process calculi are names, an abstract representation of the interaction capabilities of processes. Names are used to build elementary computations, called actions and co-actions (complementary actions). In the most basic view, like e.g., in CCS, an action is seen as an input or an output over a channel. Input and output are complementary actions. The actual interpretation of complementarity varies from one calculus to the other. The relevant fact to be pointed out here is that *complementary* actions are those that parallel processes can perform together to synchronise their (otherwise) independent behaviours. A process is computational unit that evolves by performing actions (a, b, $\cdots$) and co-actions (e.g. $\overline{a}$, $\overline{b}$, $\cdots$). The possible temporal order of the concurrent activities is specified by a limited set of operators. Sequential ordering is rendered via the prefix operator written as an infix dot. For instance the term $a.\overline{b}.P$ denotes a process that may execute the activity a, then $\overline{b}$, and then all the activities modelled by P. Two processes P and Q that run in parallel are represented by the infix parallel composition operator $\mid$ as in $P \mid Q$. Processes P and Q can either evolve independently or synchronise over complementary actions. For instance, the operational semantics of $\overline{a}.P \mid a.Q$ allows to infer the transition:

$$\overline{a}.P \mid a.Q \to P \mid Q$$

Another operator is the *choice* operator, written $+$. The process $P + Q$ can proceed either as the process P or the process Q, meaning that the two behaviours are mutually exclusive. For instance, the operational semantics of $\overline{a}.P \mid a.Q + a.R$ allows to infer the transitions:

$$\overline{a}.P \mid (a.Q + a.R) \to P \mid Q \quad \text{and} \quad \overline{a}.P \mid (a.Q + a.R) \to P \mid R$$

Another essential operator is the *restriction*. In basic calculi as CCS, this operator, written (νa), is meant to limit the visibility of actions. For instance, is not possible to infer $\overline{a}.P \mid (\nu a)a.Q \to P \mid (\nu a)Q$ because a is a private resource of the right-hand process of the parallel composition and the left-hand process cannot interact on it. This fact guarantees, e.g., that the two processes R and S in $(\nu a)(R \mid S)$ may interact over a without any interference by the external world. In more sophisticated calculi, as the π-calculus [73], the restriction operator ensures a relevant gain in expressiveness. As in CCS, the view about complementarity is limited to input and output over channels. Over CCS, however, the π-calculus allows to send channel names in interactions. This permits the representation of mobile (i.e., dynamically changing) systems: receiving new names means acquiring new interaction capabilities.

Infinite behaviours are usually obtained in process calculi by using operators like *replication*, denoted by $!P$, which allows one to create an unbounded number of parallel copies of a process P, all placed at the same level. Other mechanisms to generate infinite behaviours are *recursion* and *iteration* [77].

Above we recalled only the fundamental operators which are common to various process calculi [71, 73]. Each calculus then adopts some specific operators and has a specific view about which activities must be considered complementary. A common feature of process calculi is that their operational semantics allows to interpret process behaviours as a graph, called transition system. The nodes of the graph represent processes, and there is an arc between the two nodes P and Q if P can evolve to Q. For instance the immediate future of $P = \overline{a}.P_1 \mid a.P_2 \mid \overline{a}.P_3$ is drawn as:

$$P \to P_1 \mid P_2 \mid \overline{a}.P_3 \quad \text{and} \quad P \to \overline{a}.P_1 \mid P_2 \mid P_3$$

The depicted transitions highlight that both $\overline{a}.P_1$ and $\overline{a}.P_3$ can communicate with $a.P_2$. The evolution of the system depends upon the temporal order of the interaction. Since no assumption can be made on this, both transitions are reported in the graph. Typically, the language of any process calculus contains all the ingredients for the description of concurrent systems: a system is described in terms of *what it can* do rather than of *what it is*.

The behaviour of a complex system is expressed in terms of the meaning of its components. A model can be designed following a bottom-up approach: one defines the basic operations that a system can perform, then the whole behaviour is obtained by composition of these basic building blocks. This property is called *compositionality*. Moreover, the mathematical rules defining the operational semantics of process calculi allow to automatically generate the transition system of a given process by parsing the syntactic structure of the process itself. So, process calculi are specification languages that can be directly executed.

Process calculi are provided also with stochastic variants, primarily developed as tools for analysing the performances of concurrent systems [91, 48]. In these variants, process calculi are usually decorated with quantitative information representing the speed and probability of actions; these information are used to derive a CTMC.

2.3.2 Process calculi abstractions

The abstraction introduced by Regev and Shapiro in [98] (see Tab. 2.2), opened the realm of process calculi to the field of systems biology. A molecule is seen as a computation unit, a process, with interaction capabilities abstracted as channels names. Molecules interact/react through complementary capabilities as processes communicate on channels with the same name (action and co-action). The change of a state after a communication abstracts the dynamics of a molecule after a reaction.

Biology	**Process calculi**
Molecule	Process
Interaction capability	Channel
Interaction	Communication
Dynamics	State change

Table 2.2: Process calculi abstraction for systems biology.

Stochastic extensions of process calculi, moreover, allows to describe the same kind of stochastic processes used to directly simulate systems of coupled chemical reactions. This analogy allowed to use stochastic simulation by means of SSA Gillespie's algorithm also in the context of process calculi, denoting them definitely as a powerful abstraction for the representation of biological systems.

After the work of Regev and Shapiro [98] a number of process calculi have been adapted or newly developed for applications in systems biology. We briefly introduce the main

process calculi proposed in the last years. It is interesting to note that the various calculi are developed to study a particular aspect, i.e., they abstract a specific (or a set of) characteristic of a biological system. Indeed, an approach founded on language theory allows to fast develop specific calculi that are hypothesis driven.

Biochemical π-calculus [96]: it is the first process calculus used to represent biological systems. In biochemical π-calculus, complementary domains of interaction are represented by channel names and co-names; molecular complexes and cellular compartments are rendered by the appropriate use of restrictions on channels; molecules interaction capabilities are represented by communication. Moreover, since the calculus is stochastic, the behaviour of biological system can also be described and analysed quantitatively. Two simulators for the biochemical stochastic π-calculus have been implemented: BioSPI [96] and SPiM [85]; this simulators implement the DM of Gillespie. Moreover, interesting applications of biochemical π-calculus on real biological scenarios can be found in [67, 15, 61].

Performance Evaluation Process Algebra (PEPA) [48]: it is a formal language for describing CTMC. PEPA allows to quantitatively model and analyse large pathway systems. PEPA is supported by a large community and a lot of software tools for analysis and stochastic simulations are available. Moreover, in [17] the authors show how the combined use of PEPA and the probabilistic model checker PRISM [45] can be used to describe, simulate and analyse biochemical signalling pathways. Recently, an extension called Bio-PEPA [18] has been introduced. In this extension PEPA is modified to deal with some features of biological models, such as stoichiometry and the use of generic kinetic laws. The language is provided with a complete set of tools for performing various kinds of analyses [16].

BioAmbients [97]: it is a variant of Mobile Ambients [12] for systems biology and its focus is on biological compartments. Localization of molecules in specific compartments is extremely important in regulatory mechanisms and often a molecule can perform its task only if is in the right compartment. Ambient can be nested and organized in hierarchical way and (like in π-calculus) biological entities interact by means of communication, which can occur only between processes belonging to the same compartment, to parallel compartments or to compartments one of which is contained in the other one. Moreover, new primitives for movement between compartments are present. The language is equipped with a stochastic extension and a simulator, based on Gillespie's algorithm and implemented as part of the BioSPI project. Moreover, in [81, 20] results concerning the applicability of Control Flow Analysis and Abstract Interpretation for the static analysis

of BioAmbients models are presented.

Brane Calculi [10]: it is a calculus focused on biological membranes, which are not considered only as containers, but are active entities. A system is viewed as a set of nested membranes and a membrane as a set of actions. Brane Calculi primitives are inspired by membrane properties; membranes can merge, split, shift or act as channels. In [27], an extension called Projective Brane Calculus is presented. The goal of the extension is to refine Brane Calculi with directed actions, which tell whether an action is looking inwards or outwards the membrane. This modification brings the calculus closer to biological membranes. Recently, to improve the consistency with biological characteristics of membrane reactions, a new extension has been proposed in [28]. This extension uses a generalized formalism for action activation with a receptor-ligand type channel construction that incorporates multiple association and a concept of affinity.

CCS-R [25]: it is a CCS-like process calculus which allows the management of reversibility. Reversibility is embedded in the syntax of the calculus, which is equipped with memories that trace communications and backtrack them when needed. It is not clear whether the main features of CCS-R are relevant for modelling biological systems, because the fact that the system obtained after backtracking a reaction is the same as the initial one is still debated.

The κ-calculus [27, 23]: it is a language of formal proteins. Proteins are modelled by an identification name and by two multisets of domains, the first containing visible domains and the second containing hidden domains. Domains are simply identified by names. The authors propose *complexation* and *activation* as basic primitives of the calculus. The activation operation has the effect of moving some domains from the visible to the hidden state, and vice-versa. Complexation, instead, is used to form protein complexes, represented as graphs built over proteins and their domains; bonds between nodes are represented by means of shared names. Both operations are local and can be embedded in bigger complexes. Both operations are described by reaction rules, while proteins and complexes are combined together forming a solution. The language is equipped with a clear and useful visual notation, where proteins are represented by boxes with domains on their boundaries. The calculus is provided with an exact scalable stochastic simulator [24], and a series of tools, part of [1], that allow different kind of analyses on κ-calculus models. Moreover, [64] introduces the bioκ-calculus, a calculus for describing proteins and cells which tries to unify primitives and concepts of Brane Calculi and κ-calculus.

The π@ calculus [109]: it is an extension of the π-calculus, obtained by the addition of polyadic synchronisation and priorities. The expressiveness of the calculus is shown in [107] by providing encodings of bio-inspired formalisms like BioAmbients and Brane calculi. The language is provided with a stochastic variant (the Sπ@) that is shown to be able to model consistently several phenomena such as formation of molecular complexes, hierarchical subdivision of the system into compartments, inter-compartment reactions, dynamic reorganisation of compartment structure consistent with volume variation.

The continuous π-calculus [63]: it is process calculus for modelling behaviour and variation in molecular systems. Processes are parallel combinations of species, where species are very similar to π-calculus processes. Communication is through named channels, but there is no distinction between names and co-names. Any name can in principle communicate with any other; an *affinity matrix* specifies whether any two names can communicate and at what rate. The calculus is provided with an operational semantics in terms of real vector spaces, that offers a fully modular and compositional method of generating a set of ordinary differential equations (ODEs). The calculus is specifically designed to study evolutionary properties of biological systems.

The attributed π-calculus [54]: it is an extension of the π-calculus with attributed processes and attribute dependent synchronization. The calculus is parametrized with a call-by-value λ-calculus, which defines possible values of attributes. The calculus is provided with a non-deterministic and a stochastic semantics, where stochastic rates may depend on attribute values.

Bigraphs [74]: they are conceived as a unifying framework for designing models of concurrent and mobile systems. These reactive systems are construed as a set of rewriting rules together with an initial bigraph on which the rules operate. The main entities of bigraphs are nodes and named edges; nodes are arranged in a tree structure, they have prescribed *arities* telling how many edges must be incident to them, and edges can connect arbitrarily many nodes. In [60], a stochastic version of bigraphs is presented, along with some examples of their applicability in some biological domains.

The 3π calculus [11]: this calculus proposes an alternative form of location. Rather than keeping locations abstract, 3π uses a 3D coordinate system (actually, a full 3D affine transformation) as its set of locations. This provides great expressiveness for modelling physical systems. A similar abstraction can be found in [53], where a spatial extension of the π-calculus is considered. Processes are embedded into a vector space and move

individually. Only processes that are sufficiently close can communicate.

Beta-binders [94]: In this formalism processes are encapsulated into boxes with typed interfaces. Types represent the interaction capabilities of the boxes. Beta-binders aims at enabling non-determinism of communication by introducing the concept of *compatibility* [90], a notion that extends the *key-lock* notion of complementarity between actions and co-actions typical of process calculi, i.e., the precise matching between an input and an output over a given channel is always required. In Beta-binders, boxes have to be ready to perform complementary actions (input/output) over one of their interfaces, and the types of the involved interfaces have to be compatible. In this way, whichever notion of type compatibility is assumed, the communication ability of boxes is mainly determined by the types of their interfaces rather than by the actual naming of the relevant input and output actions. An interesting research line regarding some possible notions of type compatibility can be found in [89]. The formalism is also provided with *join* and *split* operations, i.e. parametric rules that drive the merging and splitting of boxes depending on their structure. In Beta-binders the description of such operations is left as undetermined as possible, with the goal to accommodate possible distinct instances of the same macro-behaviour. A stochastic extension of Beta-binders for quantitative experiments is presented in [30].

Part I

The **BlenX** language

Chapter 3

Informal presentation

BlenX is a language thought to deal with the complexity of protein-protein interaction. It is inspired by Beta-binders and hence has a process calculi soul. Although maintaining the basic principles characterizing Beta-binders, BlenX deviates substantially from it, hence justifying a different name. BlenX inherits from Beta-binders the abstraction used to represent biological substances (processes encapsulated into boxes with interfaces) and the notion of communication based on compatibility. It is important to underline that one important difference between BlenX and Beta-binders is that BlenX omits the restriction operator in the description of processes. Its expressive power is recovered by using a set of constructs that fit better in the biological context. These new features are:

- **complexes**: boxes can *bind* (and *unbind*) through interfaces to form (and break down) graphs of boxes, called complexes. Bindings between interfaces are dedicated *links* that allow enclosed processes to communicate in an exclusive way. Binding and unbinding operations are obtained by extending the notion of compatibility of Beta-binders;

- **priorities**: they provide a mechanism to encode high-level operations as sequences of low-level ones. In BlenX, priorities are associated with actions and are represented by positive natural numbers where one assumes that greater is the number, greater the priority it represents. The priority protocol we implement is *global*, meaning that actions with higher priority prevents the execution of all the actions with lower priority.

- **events**: they represent a reformulation of the join and split operations of Beta-binders. Intuitively events represent rules able to substitute sets of boxes with other sets of boxes. Their use is essential in modelling the dynamics of networks in which the presence of non elementary reactions makes difficult and not intuitive the use of

the communication primitives;

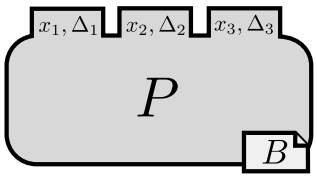
- **conditions**: the execution of primitives can be prevented by conditions that check the state of the box interfaces. They allow to condition the internal behaviour of boxes depending on the actual configurations that box interfaces can assume.

The direct application of BlenX to biological modelling is accomplished by providing a stochastic variant of BlenX. In such an extension, the language is decorated with quantitative information which are used to derive the speed and the probabilities associated with actions, as usual in the stochastic setting.

3.1 BlenX on the road

BlenX represents a biological substance as a computational object, a *box*, composed by a set of *interfaces* and an *internal process*. Interfaces are associated with structures, that we call *sorts*, and are the places where a box can interact with other boxes; the internal process, instead, codifies for the mechanism that transforms an interaction into a box structure modification. In this setting, a protein can be represented as a box, its domains by interfaces, and protein conformational states can be described by internal processes.

We use the following graphical notation of boxes:



B is the box and P is its internal process that describes its behaviour. Intuitively, P is used to program proper replies to external signals caught by the interfaces (which are the small squares on the border of the box). Sorts Δ_1, Δ_2 and Δ_3 discriminate among allowed and disallowed interactions, mimicking the interaction mechanism based on compatibility described in [90]. The names x_1, x_2 and x_3 are used by the process P to modify or to interact through the associated sorts Δ_1, Δ_2 and Δ_3, respectively. Process P is written in a process calculi style; it has few primitives inspired by π-calculus, extended with primitives inspired by molecular biology [2].

A BlenX system consists in a set of boxes, as depicted in Fig. 3.1, that run in parallel, can interact and can be attached together through their interfaces (the black lines in Fig. 3.1) forming complexes. The dynamics of a BlenX system emerges from the way in which boxes interact and change and is described in terms of an operational semantics. However, we postpone the formal description of BlenX and its dynamics to the next

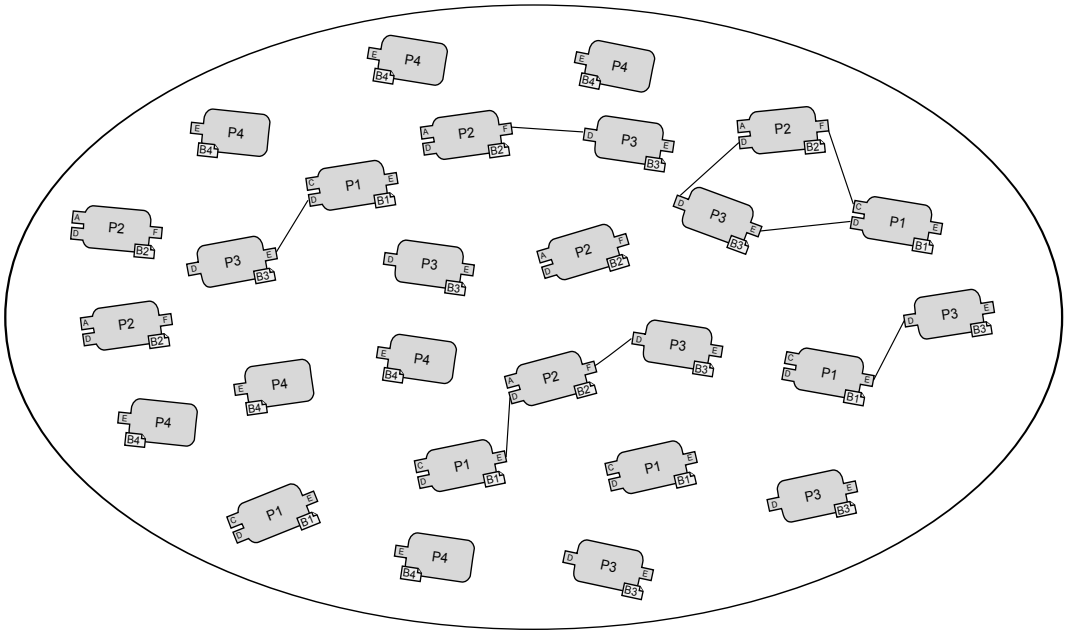
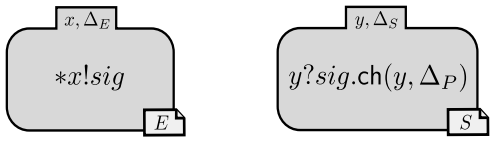


Figure 3.1: Example of BlenX system.

chapter and we put here the language on the road by considering the single substrate catalysed reaction presented in Sec. 2.1.2. We will present three different implementations, giving hence an overview of all the main features of BlenX.

Using complexes

Enzyme and substrate can be modelled in BlenX by the parallel composition of two boxes E and S, written $E \parallel S$, representing the enzyme and the substrate, respectively:



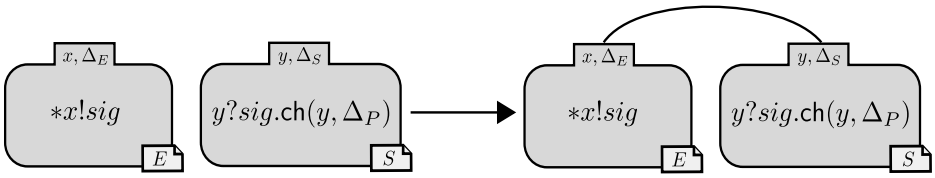
The primitive $x!sig$, in the internal process of E, sends a signal sig through the interface (x, Δ_E). Symbol $*$ indicates the replication operator (typical operator of process calculi that allows for the generation of infinite behaviours) and assures that the process sends a signal each time it is needed, i.e., each time the substrate and the enzyme interact. The primitive $y?sig$, in the internal process of S, waits for a signal on the interface (y, Δ_S) that enables the change of its sort in Δ_P, by means of $\mathsf{ch}(y, \Delta_P)$ action.

Following the reaction description in Fig.3.1, in order to interact enzyme and substrate have to bind, forming the enzyme-substrate intermediate. We assume the existence of a

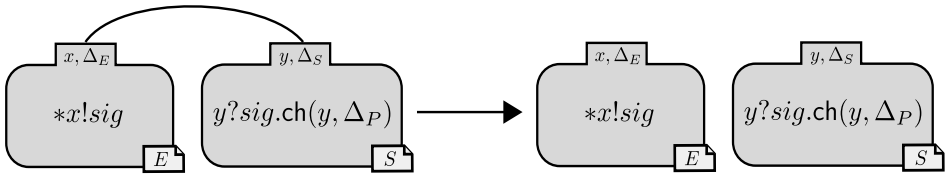
symmetric function α that, given a pair of sorts, returns a triple of values. For instance,

$$\alpha(\Delta_E, \Delta_S) = (1, 1, 1) \tag{3.1}$$

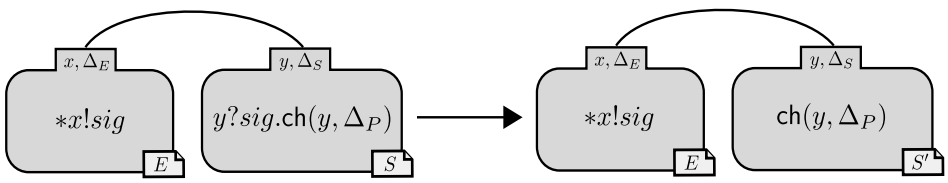
means that the sorts Δ_E and Δ_S, associated with the interfaces of the boxes E and S, may bind, unbind and communicate, respectively; the three type of actions are enabled over a certain pair of sorts only if the corresponding values are greater than zero. The result of $\alpha(\Delta_E, \Delta_S)$ in (3.1) allows hence inferring the following reaction enabled in the system $E \parallel S$:



The two boxes bind over their interfaces with sorts Δ_E and Δ_S, respectively, creating a link that only they can use. Following Fig. 2.1, the reaction that leads to the intermediate form enzyme-substrate can be reversed, namely the boxes E and S can unbind. This is represented by the following pictorial reaction:

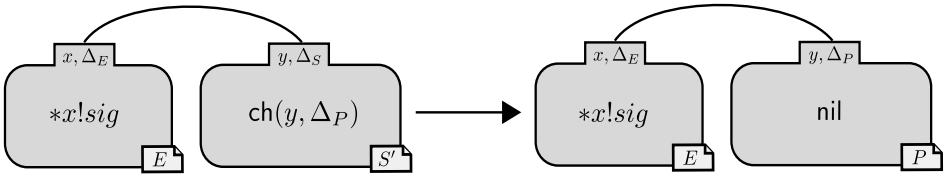


Their unbind is allowed because the corresponding value in the result of $\alpha(\Delta_E, \Delta_S)$ in (3.1) is greater than zero. However, instead of breaking up the binding, the enzyme-substrate complex can undergo certain modifications that lead to the release of the product. This is modelled in BlenX as a communication over the link created between the two interfaces through the binding, followed by a change in the sort of the interface (y, Δ_S). In particular, the process $* \; x!sig$ can send a signal sig through the interface (x, Δ_E); the process $y?sig.\,\mathsf{ch}(y, \Delta_P)$ can receive a signal from the interface (y, Δ_S) and subsequently ("subsequently" is represented by an infix dot) enable the change primitive. Since the box E can output and the box S can input, and they are bound over interfaces with a communication compatibility (the third value in the triple results of α in (3.1)) greater than zero, then a communication is possible; it leads to:



Note that the internal process of the box E is not changed because the replication operator $*$ allows to regenerate the output $x!sig$ each time it is consumed.

The internal process $\mathsf{ch}(y, \Delta_P)$ of the box S' changes the sort of the interface (y, Δ_S) from Δ_S to Δ_P, completing the transformation from substrate S to product P. This transformation is described by the following reaction:

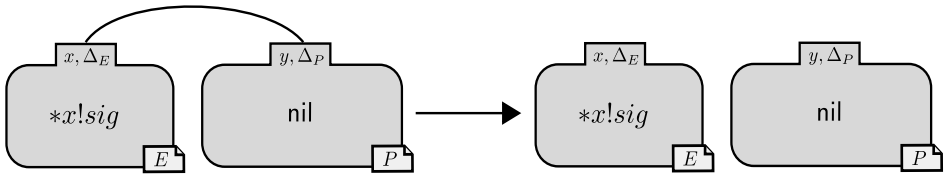


The process nil in the reaction products is the empty process. Note that more complex behaviour can be specified for the resulting product P.

Looking back to Fig. 2.1, the last step of enzymatic catalysis is the release of the product and the regeneration of the enzyme. The unbind of boxes E and P is guaranteed by assuming:

$$\alpha(\Delta_E, \Delta_P) = (0, 2, 0) \tag{3.2}$$

that prescribes that each time the types Δ_E and Δ_P are bound they have to unbind. The zeros in the first and third positions mean that types Δ_E and Δ_P cannot bind and cannot interact if they are bound. The α result (3.2) allows inferring the following reaction:



Note that in (3.2) we wrote the value 2 instead of 1. This introduces priorities and means that this action has a priority which is higher than those with priority 1. The mechanism of priorities will be explained formally and in more detail in the next chapters.

As usual, the behaviour of a BlenX system can be interpreted as a transition system. A sketch of the transition system of the enzyme catalysis example is given in Fig.3.1, where we assume to have an initial system that is the parallel composition of two enzymes and two substrates, written $E \parallel E \parallel S \parallel S$. Note that the representation of the system behaviour relies on the classical interleaved interpretation. From the initial configuration, there are four different combinations according to which enzymes and substrates can bind together and in each of the possible graph paths there are different ways to proceed. The multiple arcs exiting from a node represent all the possible reactions that can happen in that configuration, which can be independent or in competition. This reflects the real

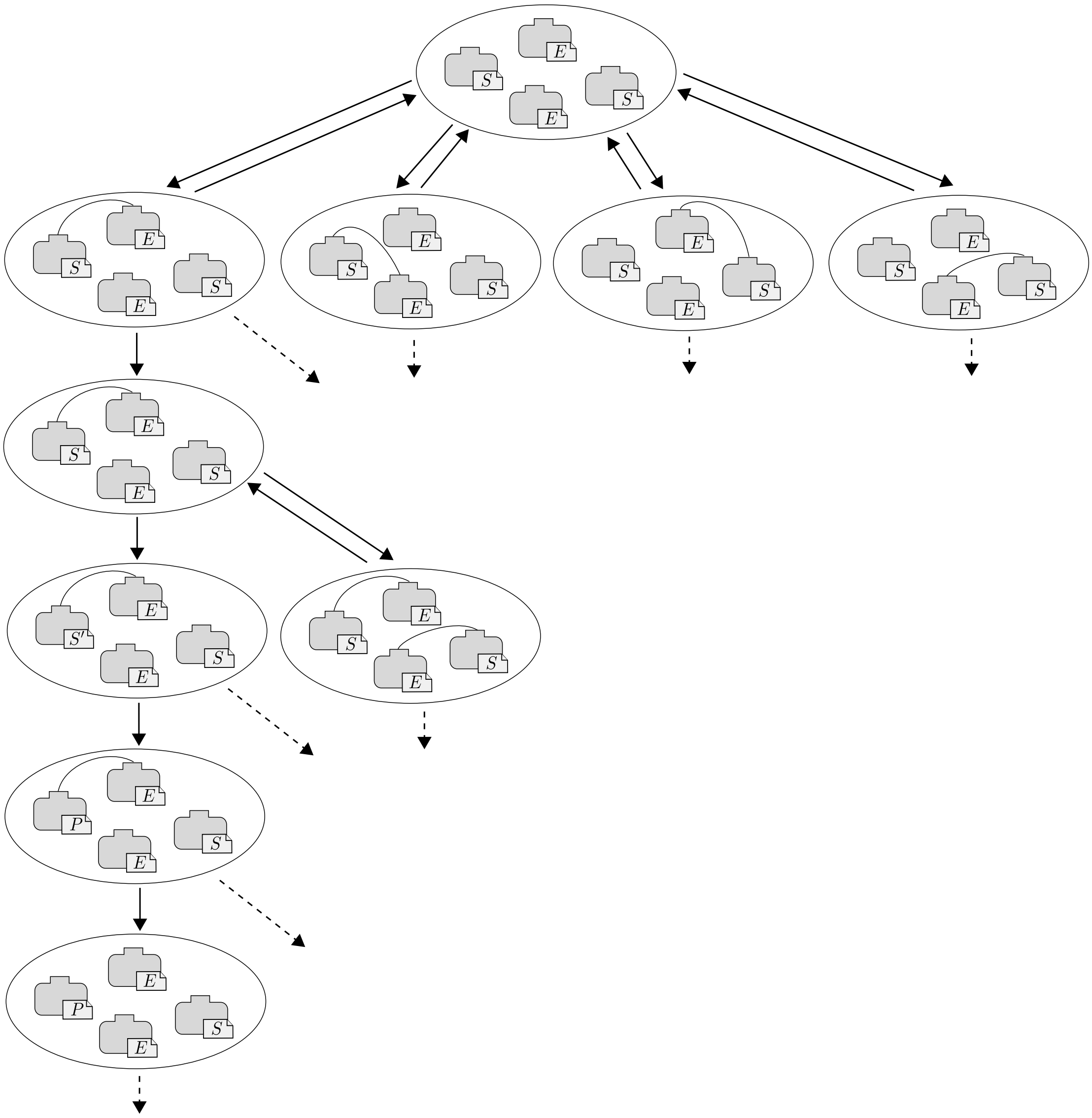
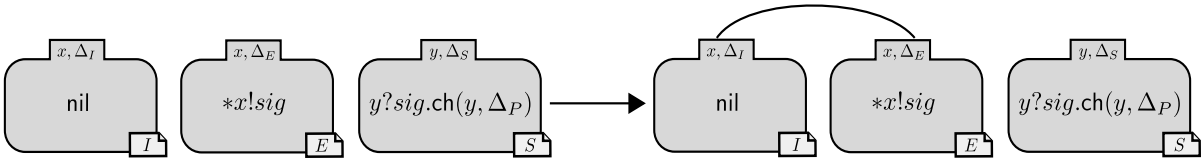


Figure 3.2: Example of BlenX system dynamics. Normal arrows represent the dynamics associated with single reactions, while dashed arrows represent the dynamics of more than one reaction.

biological world, where, e.g., internal modifications of the structure of a complex are in competition with some environmental solicitations that lead a complex to break. We will see in the next chapter how the operational semantics of BlenX allows for the automatic construction of such a transition system.

This representation of the enzymatic catalysis mechanism is pretty accurate. Furthermore, it is trivial to modify the system to introduce competitive inhibition. Indeed, it can be obtained by adding to the previous BlenX system a box I representing the inhibitor,

putting it in parallel with the existing enzymes and substrates. If we assume this box to have an interface (x, Δ_I) and we update the α function definition by adding the relation $\alpha(\Delta_E, \Delta_I) = (1, 1, 0)$, then we can infer the following dynamics:



The inhibitor I can bind to the enzyme E, hence preventing the creation of the intermediate enzyme-substrate.

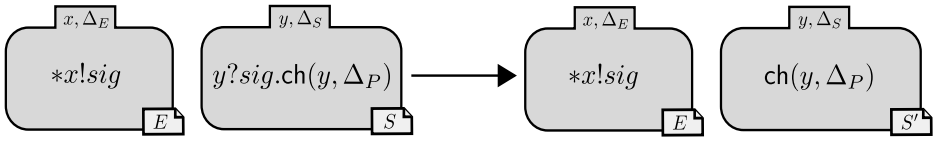
Using communication

In the previous example we have assumed that a communication between different boxes can happen only thorugh formed links. This is true only if the α specification of the corresponding interface sorts contains values for bind and unbind that are not both equal to zero. As we will see more in detail in the next chapter, if an α specification is equal to $(0, 0, n)$ with $n > 0$, then binding and unbinding are not contemplated for the pair of sorts and the boxes exposing them can communicate without the need of first creating a link. To show this we consider a simplification of the enzyme activation example where an interaction between and enzyme and a substrate leads to the release of the product without the generation of intermediate complexes.

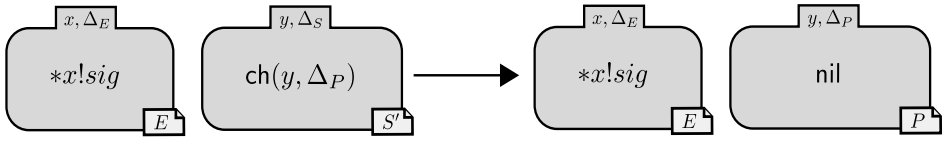
By specifying an affinity like the following:

$$\alpha(\Delta_E, \Delta_S) = (0, 0, 1) \tag{3.3}$$

we have that the initial system composed by an instance of an enzyme and a substrate can evolve as follows:



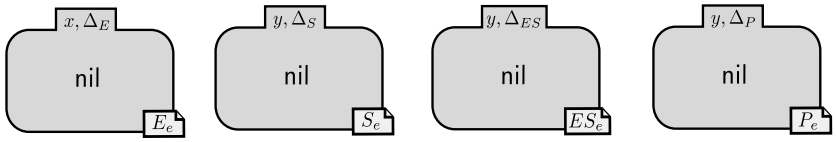
The two boxes indeed can consume an intra-communication without creating the link. Box S' can then execute the change action and transform itself in the product:



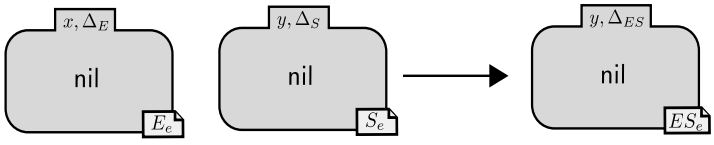
Using events

The primitives we described above work mainly with elementary reactions. In our setting, with elementary reactions we identify those reactions that do not abstract any intermediate configuration with respect to a certain known biological mechanism. In the previous example, indeed, the language primitives allowed us to model the reactions and the intermediate configurations of enzyme activation in detail, by maintaining also the identity of the single biological components. However, it is generally difficult to describe biological systems only in terms of elementary reactions, because there are scenarios in which the underlying biological mechanisms are not known with enough detail. In these cases, the application of the primitives described above results difficult. Thus, to deal also with these scenarios we introduce events.

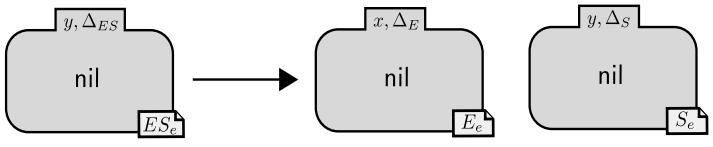
Events implement a class of rewriting rules that substitute sets of boxes with other sets of boxes. In order to better explain how events work, we consider again the previous enzyme catalysis example and model it only by using events. In this setting, we can model the enzyme, the substrate, the intermediate complexes and the product as different boxes:



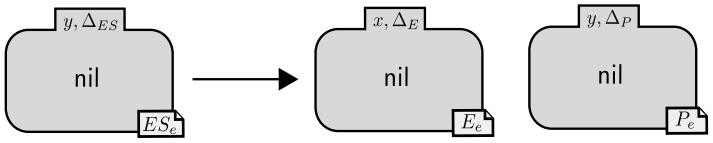
The initial configuration of the system can be modelled by $E_e \parallel S_e$. The binding and the unbinding of the enzyme and the substrate can be modelled by two different events. The first one substitutes the configuration $E_e \parallel S_e$ with ES_e, meaning that a box representing the enzyme and a box representing the substrate are substituted with a box representing their complexation:



The other event does the inverse operation, substituting a box representing the complex enzyme-substrate with a box representing the enzyme and a box representing the substrate:



Also events can be associated with priorities and here we assume all the considered events to have the same priority 1. Having generated a box ES_e we can describe the transformation of the intermediate complex and the release of the product directly using an event that substitutes the box ES_e with the parallel composition of boxes $P_e \parallel E_s$, abstracting also from the reaction that in the previous model was associated with an higher priority:



The use of events can be combined with complexes and communications. Specific configurations of internal processes and interfaces can be used control the application of events, so that they can be applied for example only after a certain number of reaction have happened. The different modelling strategies, allow hence to switch between different modelling abstraction levels, depending on the information and knowledge we have with respect to the biological process under analysis.

Chapter 4

Syntax and semantics

4.1 Syntax and notation

Let $\mathcal{N}$ be a countably infinite set of *names* (ranged over by x, y, n, x_1, x', $\cdots$) and let $\mathcal{T}$ be a finite set of *sorts* (ranged over by Δ, Γ, Δ', Δ_0, $\cdots$) such that $\mathcal{T} \cap \mathcal{N} = \emptyset$.

A BlenX system, written (B, E, ξ), is a triple made up of a bio-process B, a composition of events E and an environment ξ. We denote with $\mathcal{S}$ the set of all possible systems (ranged over by S, S', S_1, $\cdots$).

Tab.4.1 presents the complete syntax of the language. Hereafter, to simply the presentation, we will overload function names and symbols when unambiguous.

Bio-processes are generated by the non-terminal symbol B and with $\mathcal{B}$ we denote the set of all possible bio-processes (ranged over by B, B', B_1, $\cdots$). A bio-process can be either empty (Nil), a box ($I[\,P\,]_n$) or the parallel composition of bio-processes ($B_0 \parallel B_1$). In the definition of box $I[\,P\,]_n$, I represents its interaction capabilities, P is its internal engine, and n is used as an identifier to address the box at hand. Given a bio-process B, the function $Boxes(B)$ is used to extract the set of boxes composing B.

I is a non-empty string of interfaces of the form $K(x, \Delta)^p$, where K denotes the state of the interface, which can be either *free* ($\oplus$) or *bound* ($\otimes$), the name x is the *subject* of the interface, Δ is a sort representing its structure, and $p \in \mathbb{N}$ is a value (representing a priority) that will be explained in later sections. We sometimes use *active* as synonymous of free, and *complexed* as synonymous of bound. The subject x of the interface $K(x, \Delta)^p$ of a box $K(x, \Delta)^p\, I[\,P\,]_n$ is a binder for the free occurrences of x in P. We write $I = I_1 I_2$ or $I = I_1 I_2 I_3$ to mean that I is given by the juxtaposition of other interfaces. The metavariables I^*, I_1^*, $\cdots$ stay for either a sequence of interfaces or the empty string, denoted with ϵ, and the above notation for juxtaposition is extended to these metavariables in the natural way. We denote with $\mathcal{I}$ the set of all the possible interface sequences. Moreover, the functions $\mathsf{sub}(I)$ and $\mathsf{sorts}(I)$ are used, respectively, to

B	::=	**Bio-processes**
	Nil	empty
	$I[P]_n$	box
	$B \parallel B$	composition
I	::=	**Interfaces**
	$K(x,\Gamma)^p$	interface
	$K(x,\Gamma)^p\, I$	sequence
K	::=	**Interface states**
	$\oplus$	free
	$\otimes$	bound
P	::=	**Processes**
	M	capability
	$P \mid P$	composition
M	::=	**Capabilities**
	nil	empty
	$*\langle C\rangle\pi.\, P$	replicated action
	$\langle C\rangle\pi.\, P$	action
	$M + M$	choice
π	::=	**Prefixes**
	$x?y$	input
	$x!y$	output
	$\mathsf{ch}(x,\Gamma,p)$	change
C	::=	**Conditions**
	$true$	true
	(x,Γ)	check sort
	(x,K)	check state
	$op_u\ C$	unary operation
	$C\ op_b\ C$	binary operation
E	::=	**Events**
	Nil	empty
	$(B,\xi) \rhd_p (B,\xi)$	event
	$E \parallel E$	composition

Table 4.1: BlenX syntax.

extract from I the set of its subjects and the set of its sorts, and are defined in Tab. 4.2.

$\mathsf{sub}(K(x,\,\Delta)^p)$	$=$	$\{x\}$	$\mathsf{sorts}(K(x,\,\Delta)^p)$	$=$	$\{\Delta\}$
$\mathsf{sub}(K(x,\,\Delta)^p\,I)$	$=$	$\{x\} \cup \mathsf{sub}(I)$	$\mathsf{sorts}(K(x,\,\Delta)^p\,I)$	$=$	$\{\Delta\} \cup \mathsf{sorts}(I)$

Table 4.2: Functions collecting subjects and sorts of interfaces.

Definition 4.1.1. *A box $I[\,P\,]_n$ is* well-formed *if all the subjects and sorts of the interfaces composing I are distinct.*

The non-terminal symbol P generates processes and we denote with $\mathcal{P}$ the set of all the possible processes (ranged over by P, P', P_1, $\cdots$). A process can be either a capability M, or the parallel composition of two processes ($P_0 \,|\, P_1$). A capability can be either the empty process (nil), or an action-guarded process ($\langle C\rangle\pi.\,P$), or the replication of an action-guarded process ($*\langle C\rangle\pi.\,P$), or the non-deterministic choice of capabilities ($M_0 + M_1$). An action π can be either an input $x?y$, an output $x!y$ or a change $\mathsf{ch}(x, \Gamma, p)$. While the first two actions are well-known in process calculi, the last one is a peculiar of BlenX and allows to change the sort associated with the interface with subject x (if any). In particular, in $\mathsf{ch}(x, \Gamma, p)$ the name x refers to an interface subject, Γ is the sort we use for the replacement and p is a natural number representing the priority associated with the action. Guards of the shape $\langle C\rangle\pi$ extend the usual notion of action prefix in process calculi. The intuition behind process $\langle C\rangle\pi$ is that the condition C is a control over the execution of the prefix π. If C evaluates to *true*, then π can fire and P gets unblocked. Unary and binary operators used in conditions are $op_u \in \{\neg\}$ and $op_b \in \{\wedge, \vee\}$, while atoms refer to interfaces via their unique subjects and check whether they have a given sort (e.g. Δ) or whether they are in a given state (e.g. $\otimes$). Note that since the number of interfaces and the description of an interface are finite and contain all the information needed by the check, then atoms and conditions are decidable. Note that when C is equal to *true* we recover the action prefixes of classical process calculi. With $\mathcal{C}$ we denote all the possible conditions (ranged over by C, C', C_1, $\cdots$).

Given a bio-process B, the definition of free names for B, denoted by $\mathsf{fn}(B)$, is given in Tab. 4.3, along with the definition of bound names, denoted by $\mathsf{bn}(B)$. The set of names, denoted by $\mathsf{n}(B)$, the definition of α-convertibility [99] and the definition of substitution(s) [99] are extended consequently.

Boxes can be linked together through their interfaces to form complexes, which are best thought of as graphs with boxes as nodes. The environment component ξ is used to record these links; thus, a complete description of a set of complexes is given by a pair

$\mathsf{fn}(\mathsf{Nil}) = \mathsf{fn}(\mathsf{nil}) = \emptyset$	$\mathsf{bn}(\mathsf{Nil}) = \mathsf{bn}(\mathsf{nil}) = \emptyset$
$\mathsf{fn}(I[\,P\,]_n) = \mathsf{fn}(P) \setminus \mathsf{sub}(I)$	$\mathsf{bn}(I[\,P\,]_n) = \mathsf{bn}(P) \cup \mathsf{sub}(I)$
$\mathsf{fn}(B_0 \parallel B_1) = \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1)$	$\mathsf{bn}(B_0 \parallel B_1) = \mathsf{bn}(B_0) \cup \mathsf{bn}(B_1)$
$\mathsf{fn}(P_0 \,|\, P_1) = \mathsf{fn}(P_0) \cup \mathsf{fn}(P_1)$	$\mathsf{bn}(P_0 \,|\, P_1) = \mathsf{bn}(P_0) \cup \mathsf{bn}(P_1)$
$\mathsf{fn}(M_0 + M_1) = \mathsf{fn}(M_0) \cup \mathsf{fn}(M_1)$	$\mathsf{bn}(M_0 + M_1) = \mathsf{bn}(M_0) \cup \mathsf{bn}(M_1)$
$\mathsf{fn}(*\langle C\rangle\pi.\,P) = \mathsf{fn}(\pi.\,P) \cup \mathsf{fn}(C)$	$\mathsf{bn}(*\langle C\rangle\pi.\,P) = \mathsf{bn}(\pi.\,P)$
$\mathsf{fn}(\langle C\rangle\pi.\,P) = \mathsf{fn}(\pi.\,P) \cup \mathsf{fn}(C)$	$\mathsf{bn}(\langle C\rangle\pi.\,P) = \mathsf{bn}(\pi.\,P)$
$\mathsf{fn}(x?y.P) = \{x\} \cup (\mathsf{fn}(P) \setminus \{y\})$	$\mathsf{bn}(x?y.P) = \{y\} \cup \mathsf{bn}(P)$
$\mathsf{fn}(x!y.P) = \{x, y\} \cup \mathsf{fn}(P)$	$\mathsf{bn}(x!y.P) = \mathsf{bn}(P)$
$\mathsf{fn}(\mathsf{ch}(x, \Delta, p).P) = \{x\} \cup \mathsf{fn}(P)$	$\mathsf{bn}(\mathsf{ch}(x, \Delta, p).P) = \mathsf{bn}(P)$
$\mathsf{fn}(C_1 \wedge C_2) = \mathsf{fn}(C_1) \cup \mathsf{fn}(C_2)$	$\mathsf{bn}(C_1 \wedge C_2) = \emptyset$
$\mathsf{fn}(C_1 \vee C_2) = \mathsf{fn}(C_1) \cup \mathsf{fn}(C_2)$	$\mathsf{bn}(C_1 \vee C_2) = \emptyset$
$\mathsf{fn}((x, \Delta)) = \mathsf{fn}((x, S)) = \{x\}$	$\mathsf{bn}((x, \Delta)) = \mathsf{bn}((x, S)) = \emptyset$
$\mathsf{fn}(true) = \emptyset$	$\mathsf{bn}(true) = \emptyset$

Table 4.3: Definition of free and bound names for bio-processes.

(B, ξ). In detail, ξ is a set of links of the shape $\{\Delta_1 n_1, \Delta_2 n_2\}$, meaning that the two boxes addressed by n_1 and n_2 are linked together through the interfaces with sorts Δ_1 and Δ_2, respectively.

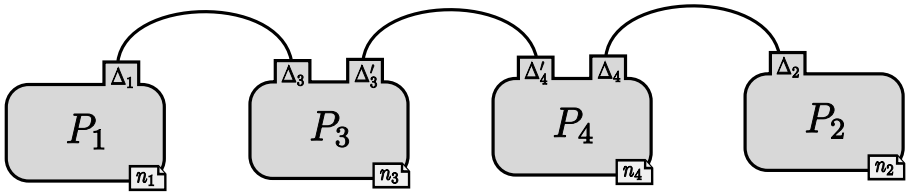


Figure 4.1: Example of a complex.

As an example, Fig. 4.1 shows a complex formed by four boxes running in parallel which contain processes P_1, P_2, P_3, P_4, and are addressed, respectively, by n_1, n_2, n_3, and n_4. In the figure, each box has one or more complexed interfaces with associated values Δ_1, Δ_2, $\cdots$. An environment representing the complex in Fig.4.1 is given by:

$$\xi_1 = (\Delta_1 n_1, \Delta_3 n_3), (\Delta'_3 n_3, \Delta'_4 n_4), (\Delta_4 n_4, \Delta_2 n_2)$$

We denote with $\mathsf{id}(\xi)$ and $\mathsf{id}(B)$ the set of names used as identifiers in the environment

ξ and bio-process B, respectively (see Tab. 4.4 for the formal definition). For instance, referring to the example in the figure and its environment ξ_1, we have that $\mathsf{id}(\xi_1) = \{n_1, n_2, n_3, n_4\}$.

$$
\begin{array}{lcl}
\mathsf{id}(\mathsf{Nil}) & = & \emptyset \\
\mathsf{id}(I[\,P\,]_n) & = & \{n\} \\
\mathsf{id}(B_1 \parallel B_2) & = & \mathsf{id}(B_1) \cup \mathsf{id}(B_2) \\
\\
\mathsf{id}(\emptyset) & = & \emptyset \\
\mathsf{id}(\{\{\Delta_1 n_1, \Delta_2 n_2\}\} \cup \xi) & = & \{n_1, n_2\} \cup \mathsf{id}(\xi)
\end{array}
$$

Table 4.4: Functions collecting boxes identifiers.

The non terminal symbol E generates a parallel composition of events. The set of all possible events is denoted with $\mathcal{E}$. An event can be either empty (Nil) or a substitution denoted by $(B_1, \xi_1) \rhd_p (B_2, \xi_2)$ (where, like interfaces, $p \in \mathbb{N}$ is a value representing a priority that will be explained in detail later). In the simplest case, events are used to substitute bio-processes with other bio-processes. Besides this, the possibility to specify explicitly the environment corresponding to a bio-processes, gives the ability to define events that represent substitutions of complexes and subcomplexes. Given a bio-process B and an environment ξ, the function CB defined in Tab.4.5 is used to collect the identity of interfaces of boxes in B which are involved in bindings outside ξ, i.e., outside the borders of B.

$$
\begin{array}{l}
\mathsf{CB}(\mathsf{Nil}, \xi) = \emptyset \\
\mathsf{CB}(B \parallel B', \xi) = \mathsf{CB}(B, \xi) \cup \mathsf{CB}(B', \xi) \\
\mathsf{CB}(I[\,P\,]_n, \xi) = \mathsf{CI}(I, \xi, n) \\
\mathsf{CI}(K(x,\, \Delta)^p\, I, \xi, n) = \mathsf{CI}(K(x,\, \Delta)^p, \xi, n) \cup \mathsf{CI}(I, \xi, n) \\
\mathsf{CI}(\oplus(x,\, \Delta)^p, \xi, n) = \emptyset \\
\mathsf{CI}(\otimes(x,\, \Delta)^p, \xi, n) = \left\{ \begin{array}{ll} \Delta n & \text{if } \nexists l \in \xi \text{ s.t. } \Delta n \in l \\ \emptyset & \text{otherwise} \end{array} \right.
\end{array}
$$

Table 4.5: Function CB for the description of the borders of complexes.

Hereafter, the typical post-fixed notations $\{{}^{a}\!/\!_{b}\}$ and σ, used to represent substitutions of the free occurrences of entities with other entities, are naturally extended to other domains, e.g., to sets. For example, $\xi\{{}^{\Delta m}\!/\!_{\Gamma n}\}$ denotes the substitution of the occurrences

of Γn with Δm in ξ. Moreover, with $\langle x/y \rangle$ we denote a notion of substitution applied only on boxes identifiers. When applied to a bio-process B (using the notation $B\langle x/y \rangle$) we obtain a bio-process in which all the boxes identifiers equal to y are substituted with x. As we will see later, we require boxes to have all distinct identifiers, hence leading to single identifiers substitutions.

Moreover, when applied to an environment ξ (using the notation $\xi\langle x/y \rangle$) it substitutes all the occurrences of the name y, present in the pairs Γy, with occurrences of x, obtaining corresponding pairs Γx. As clarifying examples, consider the following substitutions:

$$(I[\,P\,]_n \parallel I'[\,P'\,]_{n'})\langle y/n \rangle = I[\,P\,]_y \parallel I'[\,P'\,]_{n'}$$
$$\xi_1\langle n_6/n_3 \rangle = (\Delta_1 n_1, \Delta_3 n_6), (\Delta'_3 n_6, \Delta'_4 n_4), (\Delta_4 n_4, \Delta_2 n_2)$$

The language is also provided with a notion of structural congruence that identifies syntactical different systems that intuitively represent the same system. In particular, we provide notions of structural congruence for processes, bio-processes, complexes and events, and use them to define the notion of structural congruence for systems.

Definition 4.1.2. *Structural congruence on processes, denoted $\equiv_p$, is the smallest congruence on processes that satisfies the axioms in Tab. 4.6(a).*

Structural congruence on bio-processes, denoted $\equiv_b$, is the smallest congruence that satisfies the axioms in Tab. 4.6(b).

Structural congruence on complexes, denoted $\equiv_c$, is the smallest congruence that satisfies the axioms in Tab. 4.6(c).

Structural congruence on events, denoted $\equiv_e$, is the smallest congruence that satisfies the axioms in Tab. 4.6(d).

Structural congruence on systems, denoted $\equiv$, is the smallest congruence that satisfies the axiom in Tab. 4.6(e).

Axioms in Tab. 4.6(a) describe, respectively, a rule that states that α-convertible processes are structurally congruent, the monoidal axioms for parallel and choice compositions (i.e. operators $|$ and $+$ are commutative, associative and have identity element nil), the commutativity and associativity of $\wedge$ and $\vee$ boolean operators, and a rule stating that replication $*\langle C \rangle \pi.\, P$ can proceed only after firing $\langle C \rangle \pi$.

Axioms in Tab. 4.6(b) contain, respectively, the monoidal axioms for parallel composition of bio-processes and a rule that declares that the actual ordering within a sequence of interfaces is irrelevant, and states that the structural congruence of processes is reflected at the level of boxes. Axioms in Tab. 4.6(c) describe, respectively, a rule that states that the structural congruence of bio-processes is reflected at the level of complexes and a rule declaring that two complexes are equivalent up-to renaming of boxes identifiers. Ax-

a) axioms for processes

1. $P =_\alpha P' \Rightarrow P \equiv_p P'$
2. $P \mid \mathsf{nil} \equiv_p P$
3. $P_1 \mid P_2 \equiv_p P_2 \mid P_1$
4. $P_1 \mid (P_2 \mid P_3) \equiv_p (P_1 \mid P_2) \mid P_3$
5. $M + \mathsf{nil} \equiv_p M$
6. $M_1 + M_2 \equiv_p M_2 + M_1$
7. $M_1 + (M_2 + M_3) \equiv_p (M_1 + M_2) + M_3$
8. $C_1 \wedge C_2 \equiv_p C_2 \wedge C_1$
9. $C_1 \wedge (C_2 \wedge C_3) \equiv_p (C_1 \wedge C_2) \wedge C_3$
10. $C_1 \vee C_2 \equiv_p C_2 \vee C_1$
11. $C_1 \vee (C_2 \vee C_3) \equiv_p (C_1 \vee C_2) \vee C_3$
12. $*\langle C\rangle\pi.\, P \equiv_p \langle C\rangle\pi.\, (P \mid *\langle C\rangle\pi.\, P)$

b) axioms for bio-processes

1. $B \parallel \mathsf{Nil} \equiv_b B$
2. $B_1 \parallel B_2 \equiv_b B_2 \parallel B_1$
3. $B_1 \parallel (B_2 \parallel B_3) \equiv_b (B_1 \parallel B_2) \parallel B_3$
4. $P_1 \equiv_p P_2 \Rightarrow I_1^* I_2 I_3 I_4^* [\, P_1 \,]_n \equiv_b I_1^* I_3 I_2 I_4^* [\, P_2 \,]_n$
5. $P_1 \equiv_p P_2 \wedge y \notin \mathsf{fn}(P_2) \cup \mathsf{sub}(I^*) \Rightarrow$
   $K(x,\, \Delta)^p\, I^* [\, P_1 \,]_n \equiv_b K(y,\, \Delta)^p\, I^* [\, P_2\{y/x\} \,]_n$

c) axioms for complexes

1. $B_1 \equiv_b B_2 \Rightarrow (B_1, \xi) \equiv_c (B_2, \xi)$
2. $x \in \mathsf{id}(B) \wedge y \notin \mathsf{id}(B) \cup \mathsf{id}(\xi) \Rightarrow (B, \xi) \equiv_c (B\langle y/x\rangle, \xi\langle y/x\rangle)$

d) axioms for events

1. $E \parallel \mathsf{Nil} \equiv_e E$
2. $E_1 \parallel E_2 \equiv_e E_2 \parallel E_1$
3. $E_1 \parallel (E_2 \parallel E_3) \equiv_e (E_1 \parallel E_2) \parallel E_3$
4. $(B_1, \xi_1) \equiv_c (B_1', \xi_1') \Rightarrow (B_1, \xi_1) \rhd_p (B_2, \xi_2) \equiv_e (B_1', \xi_1') \rhd_p (B_2, \xi_2)$
5. $(B_2, \xi_2) \equiv_c (B_2', \xi_2') \Rightarrow (B_1, \xi_1) \rhd_p (B_2, \xi_2) \equiv_e (B_1, \xi_1) \rhd_p (B_2', \xi_2')$

e) axioms for systems

1. $(B_1, \xi_1) \equiv_c (B_2, \xi_2) \wedge E_1 \equiv_e E_2 \Rightarrow (B_1, E_1, \xi_1) \equiv (B_2, E_2, \xi_2)$

Table 4.6: Structural congruence axioms.

ioms in Tab. 4.6(d) describe, respectively, the monoidal axioms for parallel composition of events, and rules stating that the structural congruence of bio-processes and complexes is reflected at the level of events. Axiom in Tab. 4.6(e) state simply that the structural congruence of complexes and events is reflected at the level of systems. Note that in rule Tab. 4.6(c.2) a box identifier substitution has to be reflected also in the corresponding environment.

Abbreviations: We denote with $\prod_{i=1}^{n} B_i$ a parallel composition of bio-processes associated on the right (e.g., $B_1 \parallel (B_2 \parallel (B_3 \parallel B_4)) = \prod_{i=1}^{4} B_i$), with $\prod_{i=1}^{n} P_i$ a parallel composition of processes associated on the right (e.g., $P_1 \mid (P_2 \mid (P_3 \mid P_4)) = \prod_{i=1}^{4} P_i$), and with $\sum_{i=1}^{n} \langle C_i \rangle \pi_i.P_i$ a choice process associated on the right (e.g.,$\langle C_1 \rangle \pi_1.P_1 + (\langle C_2 \rangle \pi_2.P_2 + (\langle C_2 \rangle \pi_2.P_2 + \langle C_4 \rangle \pi_4.P_4)))$. Moreover, when $C = true$ we simply write $\pi.P$ and $*\pi.P$.

4.2 Static semantics

Once having the syntax of the language, the first mandatory question is whether all the systems that can be generated by the syntax, hence belonging to $\mathcal{S}$, can be considered valid. The answer is straightforwardly no, because by permitting the specification of complexes, we have to guarantee that the underline graph structure of these complexes is represented consistently in the system definition. In particular, since the specification of complexes is the result of the combination of a bio-process and an environment, we have to guarantee that given a bio-process B and an environment ξ, there is no ambiguity in interpreting from the environment how the boxes are linked together. Moreover, we want to guarantee that there is no pending information both in B and ξ.

Given a system (B, E, ξ), we require it to be *well-formed* with respect to a set of appropriate syntactical restrictions. We first require *(a) all the boxes composing B to be well-formed*. An example of not valid B is:

$$B \;=\; \otimes(x_1,\, \Delta_1)^p \otimes (x_1,\, \Delta_2)^p [\, P_1 \,]_{n_1} \parallel \otimes(x_2,\, \Delta_2)^p \otimes (x_3,\, \Delta_2)^p [\, P_2 \,]_{n_2}$$

where the interfaces of both the boxes have not distinct subjects and not distinct sorts, respectively. Then we require *(b) all the boxes identifiers in B to be distinct*. An example of invalid B is:

$$B \;=\; \otimes(x_1,\, \Delta_1)^p [\, P_1 \,]_{n} \parallel \otimes(x_2,\, \Delta_2)^p [\, P_2 \,]_{n}$$

where both the boxes have the same identifier. The environment ξ is valid only if *(c) each link component Δx in the environment ξ appears only once in ξ and no self-links are*

present. For example, an invalid environment is:

$$\xi = \{\{\Delta n, \Gamma n_1\}, \{\Delta n, \Gamma n_2\}, \{\Delta n_3, \Gamma n_4\}\}$$

Given valid B and ξ, we have to be sure that a series of requirement are satisfied by the combination B and ξ. We require that *(d) if Δn appears in ξ then a box in B with identifier n and complexed interface with sort Δ exists.* An example of invalid system contains the following B and ξ:

$$\begin{aligned} B &= \oplus(x_1, \Delta_1)^p[\, P_1 \,]_{n_1} \parallel \otimes(x_2, \Delta_2)^p[\, P_2 \,]_{n_2} \parallel \oplus(x_4, \Delta_4)^p[\, P_3 \,]_{n_3} \\ \xi &= \{\{\Delta_1 n_1, \Delta_2 n_2\}, \{\Delta_2 n_2, \Delta_3 n_3\}\} \end{aligned}$$

Note that $\Delta_3 n_3$ is in ξ, but there is no box with n_3 identifier in B. Moreover, $\Delta_1 n_1$ is in ξ but the corresponding interface of box $\oplus(x_1, \Delta_1)^p[\, P_1 \,]_{n_1}$ in B is not signed as complexed.

Similarly we require that *(e) if a box in B with identifier x and complexed interface with sort Δ exists then the link component Δx appears in the environment ξ.* An example of invalid system is:

$$\begin{aligned} B &= \otimes(x_1, \Delta_1)^p[\, P_1 \,]_{n_1} \parallel \otimes(x_2, \Delta_2)^p[\, P_2 \,]_{n_2} \parallel \otimes(x_4, \Delta_4)^p[\, P_3 \,]_{n_3} \\ \xi &= \{\{\Delta_4 n_4, \Delta_2 n_2\}, \{\Delta_2 n_2, \Delta_3 n_3\}\} \end{aligned}$$

Note that the combination of the previous requisites prevents also from the description of systems with boxes interfaces having multiple links.

All the inconsistencies checked by the previous syntactical restrictions can rise also at the level of events definition. When considering events, anyway, the main difference with respect to the problems seen above, is that in the specification $(B_1, \xi_1) \rhd_p (B_2, \xi_2)$ we do not require a complete correspondence between the bound interfaces present in the bio-processes and the link components in the environments. This is because, as previously informally described, events can be used to represent substitutions of subcomplexes. In this cases, however, we have to guarantee that subcomplexes substitutions do not generate hanging bounds. Thus, surely *(f) for all the events $(B, \xi) \rhd_p (B', \xi')$ composing E, conditions (a), (b), (c) and (d) have to hold for B and ξ, and for B' and ξ'.* Moreover *(f) for all the events $(B, \xi) \rhd_p (B', \xi')$ composing E, the sorts of complexed interfaces in B without a link component in ξ and the sorts of complexed interfaces in B' without a link component in ξ' have to coincide.*

All the conditions above are formalized by providing a simple proof system. The axioms and inference rules characterizing our proof system are defined inductively on the

structure of BlenX systems and are reported in Tab.4.7. Given a system S, if S satisfies the judgement $\vdash S : ok$, then it is well-formed. The main characteristic of the proof system, is that it associates specific signatures to the bio-processes, interfaces and environments composing a system, in order to collect all the information needed to prove the well-formedness property. Depending on the syntactic category, signatures are different and collect different information.

The well-formedness of interfaces is verified by judgements and rules (w1-3). These rules verify that all the subjects and sorts of the interfaces of a box are distinct; they generate a signature that is a set containing all the sorts of bound interfaces.

A well-formed bio-process B is associated with a signature (N, L), where $N \subset \mathcal{N}$ contains the identifiers of all the boxes defined in B, and L contains a pair (Δ, n) if and only if an interface with sort Δ is declared as bound in the box with identifier n. Note that the notion of identifier substitution $\langle y/x \rangle$ can be easily extended to L. All the judgements and rules guarantee that no boxes with identical identifiers are defined. Rule (w4) says that the empty bio-process Nil is well-formed and has signature $(\emptyset, \emptyset)$. Rule (w5) controls the well-formedness of boxes. A well-formed box $I[\,P\,]_n$ is associated with a signature $(\{n\}, \{(\Delta, n) \mid \Delta \in \mathcal{T}\})$, where the first element is a set containing the identifier of the box and the second element contains the sorts $T \subset \mathcal{T}$ of the bound interfaces in I. The premise of rule (w5) controls the well-formedness of the interface I and generates the signature T. Rule (w6) states that given two bio-processes B_1 and B_2, well-formed with signatures (N_1, L_1) and (N_2, L_2) where $N_1 \cap N_2 = \emptyset$, the bio-process obtained by composing in parallel the two bio-processes is well-formed with signature $(N_1 \cup N_2, L_1 \cup L_2)$. Note that by controlling that the intersection of the two set of boxes identifiers is empty, we guarantee that no boxes in B_1 and B_2 have the same identifier.

An environment is well-formed (w7-9) if it does not contain multiple link components Δn. This guarantees that the environment is not ambiguous in terms of complex links (i.e., we don't have the specification of multiple links on the same interface) and that no self-links are introduced. Parallel compositions of events are well-formed (w10-12) if: all the single events are composed by well-formed bio-processes B_i and environments ξ_i (with $i = 1, 2$); signatures (N_i, L_i) and L'_i, respectively of B_i and ξ_i, guarantee that all the link components in L'_i are also in L_i ; pending binding sorts of B_1 and B_2 coincide. The last two conditions are verified by checking that $L'_i \subseteq L_i$ and that for each $\Delta \in \mathcal{T}$ we have that the cardinalities of sets $\{(\Delta, n) \in L_1 \setminus L'_1 \mid n \in \mathcal{N}\}$ and $\{(\Delta, n) \in L_2 \setminus L'_2 \mid n \in \mathcal{N}\}$ coincide, respectively. Notice that we do not require here equality because in events we can also specify bio-processes representing subcomplexes. Equality is instead a requisite in rule (w13), because we want to ensure that all the bindings specified in the environment are be contained in the corresponding bio-processes, and vice-versa.

(w1) $\vdash_i \oplus(x,\,\Delta)^p : \emptyset$ (w2) $\vdash_i \otimes(x,\,\Delta)^p : \{\Delta\}$

(w3) $$\frac{\vdash_i K(x,\,\Delta)^p : T \wedge \vdash_i I : T'}{\vdash_i K(x,\,\Delta)^p\, I : T \cup T'}\ ,\ x \notin \mathsf{sub}(I) \text{ and } \Delta \notin \mathsf{sorts}(I)$$

(w4) $\vdash_b \mathsf{Nil} : (\emptyset, \emptyset)$

(w5) $$\frac{\vdash_i I : T}{\vdash_b I[\,P\,]_n : (\{n\}, \{(\Delta, n) \mid \Delta \in T\})}$$

(w6) $$\frac{\vdash_b B_1 : (N_1, L_1) \ \wedge\ \vdash_b B_2 : (N_2, L_2)}{\vdash_b B_1 \parallel B_2 : (N_1 \cup N_2, L_1 \cup L_2)}\ ,\ N_1 \cap N_2 = \emptyset$$

(w7) $\vdash_{en} \emptyset : \emptyset$

(w8) $\vdash_{en} \{\Delta_1 n_1, \Delta_2 n_2\} : \{(\Delta_1, n_1), (\Delta_2, n_2)\}\ ,\ n_1 \neq n_2$

(w9) $$\frac{\vdash_{en} \xi_1 : L_1 \ \wedge\ \vdash_{en} \xi_2 : L_2}{\vdash_{en} \xi_1 \cup \xi_2 : L_1 \cup L_2}\ ,\ L_1 \cap L_2 = \emptyset$$

(w10) $\vdash_{ev} \emptyset : ok$ (w11) $$\frac{\vdash_{ev} E_1 : ok \wedge \vdash_{ev} E_2 : ok}{\vdash_{ev} E_1 \parallel E_2 : ok}$$

(w12) $$\frac{\vdash_b B_1 : (N_1, L_1) \wedge \vdash_b B_2 : (N_2, L_2) \wedge \vdash_{en} \xi_1 : L_1' \wedge \vdash_{en} \xi_2 : L_2'}{\vdash_{ev} (B_1, \xi_1) \rhd_p (B_2, \xi_2) : ok}$$

provided $L_1' \subseteq L_1,\ L_2' \subseteq L_2, N_1 \neq \emptyset$ and
$\forall\ \Delta \in \mathcal{T}.\ |\{(\Delta, n) \in L_1 \setminus L_1' \mid n \in \mathcal{N}\}| = |\{(\Delta, n) \in L_2 \setminus L_2' \mid n \in \mathcal{N}\}|$

(w13) $$\frac{\vdash_b B : (N, L) \wedge \vdash_{ev} E : ok \wedge \vdash \xi : L}{\vdash (B, E, \xi) : ok}$$

Table 4.7: Well-formedness proof system.

Definition 4.2.1. *The set of well-formed BlenX systems is* $\overline{\mathcal{S}} = \{S \in \mathcal{S} \mid \vdash S : ok\}$.

We show now that well-formedness condition is preserved by structural congruence. We do it by first showing it on bio-processes, complexes and events. Intermediate lemmas are also introduced.

Lemma 4.2.2. *Let $B, B' \in \mathcal{B}$ s.t. $B \equiv_b B'$. Then $\vdash_b B : (N, L)$ implies $\vdash_b B' : (N, L)$.*

Proof Sketch. By induction on the length of the derivation $B \equiv_b B'$. For the base cases Tab. 4.6(b.4) and Tab. 4.6(b.5) it is enough to observe that the structural modifications introduced by the axioms do not change the signature T generated by the rule premise and the identifier of the box. □

Lemma 4.2.3. *Le $B \in \mathcal{B}$ s.t. $\vdash_b B : (N, L)$ and $y \notin \mathsf{id}(B)$. Then $\vdash_b B\langle y/x\rangle : (N\{y/x\}, L\langle y/x\rangle)$.*

Proof Sketch. By induction on the derivation $\vdash_b B : (N, L)$ and identifier substitution definition. □

Lemma 4.2.4. *Le ξ be an environment s.t. $\vdash_{en} \xi : L$ and $y \notin \mathsf{id}(\xi)$. Then $\vdash_{en} \xi\langle y/x\rangle : L\langle y/x\rangle$.*

Proof Sketch. By induction on the derivation $\vdash_{en} \xi : L$ and identifier substitution definition. □

Lemma 4.2.5. *Let $(B, \xi), (B', \xi')$ be complexes such that $(B, \xi) \equiv_c (B', \xi')$. Then $\vdash_b B : (N, L)$ and $\vdash_{en} \xi : L$ implies $\vdash_b B' : (N', L')$ and $\vdash_{en} \xi' : L'$.*

Proof Sketch. By induction on the length of the derivation $(B, \xi) \equiv_c (B', \xi')$. The case Tab. 4.6(c.1) follows by Lemma 4.2.2, while case Tab. 4.6(c.2) follows by Lemmas 4.2.3 and 4.2.4. □

Lemma 4.2.6. *Let $E, E' \in \mathcal{E}$ such that $E \equiv_e E'$. Then $\vdash_b E : ok$ implies $\vdash_b E' : ok$.*

Proof Sketch. By induction on the length of the derivation $E \equiv_e E'$ and by using the Lemma 4.2.5. □

All the previous Lemmas can be combined to show that given a well-formed system S, we have that all the systems structurally congruent to S are also well-formed.

Theorem 4.2.7. *Let $S \in \overline{\mathcal{S}}$ and $S' \in [S]_{\equiv}$. Then $S' \in \overline{\mathcal{S}}$.*

Proof Sketch. From Lemmas 4.2.2, 4.2.5 and 4.2.6. □

4.3 Species and complexes

In biology, a *species* is defined by a set of characteristic that are shared by all the components of a species and only by them. Notions of species can be found starting from

molecular level till to organisms level: in chemistry, a species defines an ensemble of chemically identical molecular entities that can explore the same set of molecular energy levels on a characteristic or delineated time scale; in proteomics, the meaning of species is used to classify proteins with respect to their functions, structures, isoforms, post-translational modifications; individuals belonging to a group of organisms having common characteristics and capable of mating with one another define a species.

The development of a notion of species, therefore, is essential for classification purposes at all the observational levels. In this thesis we are dealing with the dynamics of biological systems, and hence our need and intuition is that a notion of species in this context has to be related with the concept of dynamics. Besides the intrinsic interest in developing a notion of species for dynamical systems, it is also evident how it can be useful in practice. Indeed, it allows for example to move from an individual level view to a population level view, hence giving the possibility to reduce the system state space and study the dynamics of populations rather than of single individuals.

In BlenX boxes represent the basic abstraction that allow us to represent biological entities and hence we are mainly interested in defining a notion of species over boxes. Our idea is to provide a notion of species that relates boxes both in terms of their structure and dynamics. It is clear that there are several choices, starting from a strict syntactic equivalence, till to the development of a behavioural theory that relates boxes purely in terms of their dynamics. Searching something in the middle, we finally decided to base our notion of species on the notion of structural congruence, which we think relates boxes both considering in an appropriate way their structure and their dynamics. We think that a notion of species based on structural congruence is a good candidate because theoretically permits to identify syntactical different boxes that intuitively represent the same box and more practically, as we will show in this section, can be computed efficiently in our case. Moreover, a notion of species based on structural congruence of boxes can be naturally reflected at the level of complexes, where structural congruence can be itself used to compare and classify complexes in terms of their components and their topologies.

In this section we develop on the structural congruence of boxes and complexes, showing that the problem of deciding if two boxes or two complexes are structurally congruent is a decidable and efficiently solvable, i.e., there is a procedure that works polynomially in the size of the input. We start by developing on the structural congruence of processes, passing then in a natural way to reason about the structural congruence of boxes and finally develop on the structural congruence of complexes. All the reductions, procedures and functions given in the section are presented in a constructive way, so that it results easier to see how they can be implemented. Moreover, we state that all the results of this section can be combined and used to prove that the general structural congruence of

BlenX systems is decidable and efficiently solvable.

We think all the results of this section are important because shows how BlenX is provided with a notion of structural congruence that can be really used for implementation purposes. Indeed, this is not true in general for many other process calculi, where notions of structural congruence are heavily used theoretically, but are very difficult to check or even it is not know whether they are decidable [32, 33, 34, 35].

4.3.1 Structural congruence of processes

Here we show that for our processes $\mathcal{P}$, the problem of deciding if two of them are structurally congruent can be polynomially reduced to a tree isomorphism problem, hence being not only decidable but also efficiently solvable [59].

Although our processes belong to the class of processes studied in [35], our notion of structural congruence is slightly different with respect to the standard one and hence their results cannot be applied directly. In [35], the main difficulty in showing the decidability of structural congruence is the treatment of replication, which allows a process to grow indefinitely without maintaining a precise structure in its number of subprocesses. The structural axiom for replication used in this thesis, instead, allows a process to grow indefinitely in its number of subprocesses maintaining a precise in-depth structure. As shown in the following, this is one of the key ingredients on which the work of this section is based.

We start by defining a function that recognizes and eliminates all the expanded replications of a given process P. As an example, consider the following process:

$$P = x?a.\,(\ z?d.\,\mathsf{nil} \mid\ *\,x?a.\,(*y?b.\,\mathsf{nil} \mid z?c.\,\mathsf{nil}) \mid y?b.\ *\,y?e.\,\mathsf{nil}\ )$$

We want a function that, applied on P, recognizes that the subprocess $y?b.\ *\,y?e.\,\mathsf{nil}$ is a one level expansion of the replication $*y?e.\,\mathsf{nil}$, and compresses it. Then, the function has to recognize that the whole process is a one level expansion of the replication $*x?a.\,(*y?b.\,\mathsf{nil} \mid z?c.\,\mathsf{nil})$, and has to return a process that does not contain expanded replications.

This function, that we call *Impl*, is defined in Tab. 4.8 by induction on the structure of P, where with $\equiv_p^{\mathbf{rep}}$ we identify a structural congruence relation that omits the rule for the replication.

Definition 4.3.1. *$\equiv_p^{rep}$ is the smallest congruence relation on processes that satisfies all the axioms in Tab. 4.6 but the last one, the axiom for replication.*

In Tab. 4.8, moreover, the function $Par(P)$ extracts from P all the unguarded parallel capabilities and, given a finite set of capabilities S, with $\prod_S P_i$ we denote a process

$S_1 \,|\, (S_2 \,|\, (\, \cdots \,|\, S_n) \cdots))$ generated by the parallel composition of all the capabilities in S, where $\prod_S P_i = \mathsf{nil}$ if $S = \emptyset$. Note that since processes have finite length, then the function *Impl* surely ends. Moreover, $Par(P)$ and the generation of $\prod_S P_i$ (given a finite set S) have a complexity which is linear in the number of the unguarded parallel capabilities in P (which is a finite number) and in the number of capabilities in S, respectively.

```
IMPL (P):
  switch  P
    case  P1 | P2  :
      return  IMPL(P1) | IMPL(P2)

    case  M1 + M2  :
      return  IMPL(M1) + IMPL(M2)

    case  nil  :
      return  nil

    case  *⟨C⟩π. P'  :
      return  *IMPL(⟨C⟩π. P')

    case  ⟨C⟩π. P'  :
      if  ∃P'' ∈ PAR(IMPL(P')) s.t.
          P'' = ⟨C'⟩π'.R ∧ ⟨C⟩π.Q ≡_p^rep P''
          with Q = ∏_{PAR(IMPL(P'))\P''} P_i
        then
          return  ⟨C⟩π. Q

      else
          return  ⟨C⟩π. IMPL(P')
```

Table 4.8: Definition of function $Impl$.

The application *Impl(P)* propagates *Impl* recursively to all the subprocesses of P. For each subprocess of the form $\langle C \rangle \pi.P'$, the function controls if the recursive invocation $Impl(P')$ results in a process that contains an unguarded capability $P'' = *\langle C' \rangle \pi'.R$ such that $\langle C \rangle \pi.Q \equiv_p^{\mathbf{rep}} \langle C' \rangle \pi'.R$, with $Q = \prod_{Par(Impl(P'))\setminus P''}$. If it is the case, this means that the subterm $\langle C' \rangle \pi'.R$ corresponds to a replication expansion and hence $\langle C \rangle \pi.P'$ can be substituted with the imploded process $\langle C \rangle \pi.Q$. Obviously, the complexity of the control depends on the number of parallel components of $Impl(P')$ and on the complexity of $\equiv_p^{\mathbf{rep}}$. In particular, note that if the congruence $\equiv_p^{\mathbf{rep}}$ is efficiently solvable, then also the function $Impl(P)$ is efficiently solvable, i.e., the complexity is polynomial in the size of P.

Lemma 4.3.2. *Let $P \in \mathcal{P}$. Then $Impl(P) \equiv_p P$.*

Proof sketch. By induction on the structure of P. Every modification that the function $Impl$ carries out on the structure of P comes from the recursive invocation of $Impl(\langle C\rangle\pi.P')$. These rearrangements and modifications are equivalent to the application of a sequence of axioms (a.1-3) and (a.12). These axioms are a subset of the structural axioms of the congruence $\equiv_p$. Hence $Impl(P) \equiv_p P$. □

Now consider the subclass of guarded replication processes that does not contain expanded replications. We call this subclass $\mathcal{P}_{rp}$.

Definition 4.3.3. *Let $\mathcal{P}_{rp}$ to be the subset of processes in $\mathcal{P}$ such that $P \in \mathcal{P}_{rp}$ if and only if P does not contain subterms $\equiv_p^{\mathbf{rep}}$ congruent to processes of the form $\langle C\rangle\pi.\,(P \mid *\langle C\rangle\pi.\,P)$.*

Lemma 4.3.4. *Let $P \in \mathcal{P}$. Then $Impl(P) \in \mathcal{P}_{rp}$.*

Proof. Immediate from $Impl$ definition. □

Lemma 4.3.5. *Let $P, Q \in \mathcal{P}_{rp}$. Then $P \equiv_p Q$ iff $P \equiv_p^{\mathbf{rep}} Q$.*

Proof. ($\Rightarrow$) To show this implication we prove that in $P \equiv_p Q$ the law (a.12) is never used. Assume that P is obtainable from Q by applying, for some subterm of Q, the law (a.12). This means that one of the two processes has a subterm in the form $\langle C\rangle\pi.(R \mid *\langle C\rangle\pi.R)$. But the subterm $\langle C\rangle\pi.(R \mid *\langle C\rangle\pi.R)$ expands the replication $*\langle C\rangle\pi.R$ and this contradicts our initial assumption that $P \in \mathcal{P}_{rp}$. Therefore, the law (a.12) is never used and the implication is true.
($\Leftarrow$) Since the structural axioms of the congruence $\equiv_p^{\mathbf{rep}}$ are a subset of the structural axioms of the congruence $\equiv_p$ then $P \equiv_p^{\mathbf{rep}} Q$ implies $P \equiv_p Q$. □

Lemma 4.3.6. *Let $P, Q \in \mathcal{P}$. Then $P \equiv_p Q$ iff $Impl(P) \equiv_p^{\mathbf{rep}} Impl(Q)$.*

Proof. ($\Rightarrow$) Since $Impl(P) \equiv_p P \equiv_p Q \equiv_p Impl(Q)$ (using Lemma 4.3.2) we obtain that $Impl(P) \equiv_p Impl(Q)$. Since the processes $Impl(P)$ and $Impl(Q)$ does not contain expanded replication, we have that $Impl(P) \equiv_p Impl(Q)$ (using Lemma 4.3.5) implies $Impl(P) \equiv_p^{\mathbf{rep}} Impl(Q)$.
($\Leftarrow$) The structural laws of congruence $\equiv_p^{\mathbf{rep}}$ are a subset of the structural laws of the congruence $\equiv_p$. For this reason $Impl(P) \equiv_p^{\mathbf{rep}} Impl(Q)$ implies $Impl(P) \equiv_p Impl(Q)$. For the Lemma 4.3.2 we have that $P \equiv_p Impl(P) \equiv_p Impl(Q) \equiv_p Q$ and therefore, for transitivity, we have that $P \equiv_p Q$. □

Having reduced $\equiv_p$ to $\equiv_p^{\mathbf{rep}}$, the next step is to show that $\equiv_p^{\mathbf{rep}}$ is decidable and efficiently solvable. In order to do this we proceed similarly by reducing the problem

of determining if two processes are $\equiv_p^{\mathbf{rep}}$ to a tree isomorphism problem. We start by providing a polynomial procedure that, given a process, constructs a corresponding n-ary tree representing its nameless tree representation. The main characteristics of this procedure are:

a) following the De Bruijn indices approach [29], all the binders in the inputs and all the bound names are substituted with integers representing the binding depth in the process;

b) multiple composition of binary parallels and choice in processes and binary operations in conditions are compressed in a unique level;

c) empty processes in parallel and choice compositions are eliminated.

Before starting a formal presentation of the tree construction we give a small intuitive example. Consider the process:

$$\langle (x,\Delta)\rangle x?y.\mathsf{nil} \mid \langle (x,\Delta)\rangle x?y.\langle (y,\Delta)\rangle y?z.\langle (z,\Delta)\rangle z!y.\mathsf{nil} + \langle true\rangle x?k.\mathsf{nil} + \langle (x,\Delta)\rangle y!z.\mathsf{nil}$$

Its tree representation is reported in Fig.4.2; input binders are substitute with integers, representing the in-depth binding, and all the binary operators are compressed.

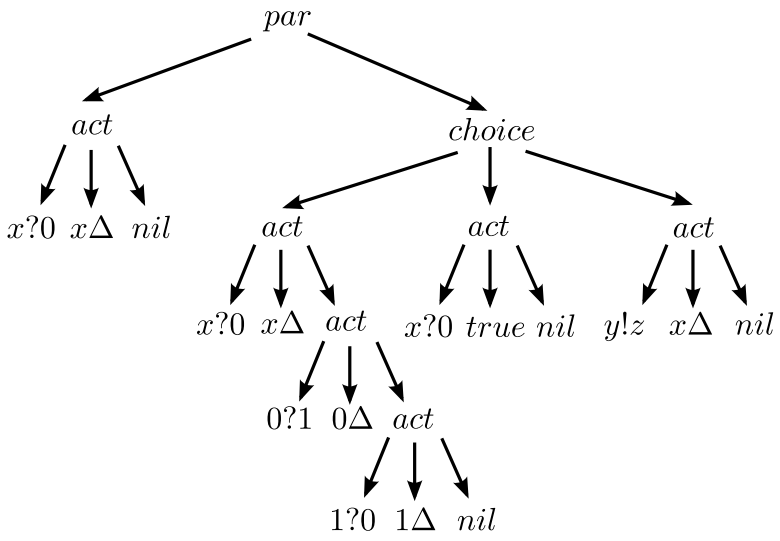


Figure 4.2: Example of process tree representation.

Axioms and rules reported in Tab.4.9 and Tab.4.10 allow for the construction of process tree representations. With ψ we indicate a sequence of binding names and natural numbers, called also *scope*. We use the *comma* operator to extend a scope ψ by adding a new binding on the right. Note that by extending we mean that the instances on the right

overwrite instances on the left: if we have $\psi = x : 3, x : 2$, then $\psi(x) = 2$. Moreover, with $\hat{\psi}(x)$ we indicate a function that returns $\psi(x)$ if $x \in dom(\psi)$ and returns x otherwise. The empty scope is denoted by $\emptyset$. The set of all the possible scopes (ranged over by ψ, ψ', ψ_1, $\cdots$) is denoted by Ψ.

The class of trees considered here are n-ary labeled trees. Tree *nodes* are denoted with $(label; T_1, \cdots, T_m)$, where *label* is a string and denotes the label associated with the node, while $T_1, \cdots, T_m$ are the immediate subtrees of the node. With $(label)$ we denote a *leaf*. The *size* of T is denoted with $|T|$ and indicates the number of nodes composing the tree T. Given a tree T, we denote with $\mathsf{label}(T)$ the function that returns the label of its root node.

Definition 4.3.7. *Let T and T' be trees. We define $T \simeq T'$ (T isomorphic to T') by induction on the number of nodes in T and T' by defining that $T \simeq T'$ holds if and only if:*
(a) $|T| = |T'| = 1$ and $\mathsf{label}(T) = \mathsf{label}(T')$;
(b) T and T' are such that $\mathsf{label}(T) = \mathsf{label}(T')$ and both have the same number, m, of immediate subtrees, and there is some ordering $T_1, \ldots, T_m$ of the immediate subtrees of T and some ordering $T'_1, \ldots, T'_m$ of the immediate subtrees of T' such that $T_i \simeq T'_i$ for all $1 \leq i \leq m$.

Definition 4.3.8. *Given a process P, an initial scope ψ and a natural number n, the tree T we obtain by deriving $\psi \vdash_n P : T$ using the axioms and rules reported in Tab. 4.9 and Tab. 4.10 is a nameless tree representation of P, and is denoted also by $st(P)^{\psi}_{n}$.*

It is clear that depending on ψ and n we obtain different tree representations. Since the goal of the tree construction is to satisfy the points a), b) and c) previously mentioned, all these representations are valid representations. When comparing tree representations of different processes we will always specify the scopes and the numbers we use in their generation.

Tab.4.9 refers to processes. Rules (t1-4) allow the compression in a unique level of multiple parallel compositions. Note that in order to capture all the possible syntactical constructions we need four rules in which we render explicit all the possible cases. Rules (t5-6) allow the elimination of empty capabilities in parallel compositions. Rules (t7-12) are similar to the previous ones and allow the compression in a unique level of multiple choices with the elimination of empty capabilities. Rule (t13) introduces a node representing replication, while rule (t14) introduces a leaf representing the nil process. Rules (t15-17), instead, are used to represent input, output and change capabilities. Note that the first immediate node subtree introduced by all these three rules is a leaf containing a nameless representation of the capability. Moreover, in order to propagate the nameless

$$\text{(t1)}\ \frac{\psi \vdash_n P_1 : (par; T_1, \cdots, T_m) \quad \psi \vdash_n P_2 : T_0 \quad \mathsf{label}(T_0) \notin \{par, nil\}}{\psi \vdash_n P_1 \,|\, P_2 : (par; T_1, \cdots, T_m, T_0)}$$

$$\text{(t2)}\ \frac{\psi \vdash_n P_1 : T_0 \quad \psi \vdash_n P_2 : (par; T_1, \cdots, T_m) \quad \mathsf{label}(T_0) \notin \{par, nil\}}{\psi \vdash_n P_1 \,|\, P_2 : (par; T_0, T_1, \cdots, T_m)}$$

$$\text{(t3)}\ \frac{\psi \vdash_n P_1 : (par; T_1, \cdots, T_m) \quad \psi \vdash_n P_2 : (par; T_{m+1}, \cdots, T_k)}{\psi \vdash_n P_1 \,|\, P_2 : (par; T_1, \cdots, T_k)}$$

$$\text{(t4)}\ \frac{\psi \vdash_n P_1 : T_1 \quad \psi \vdash_n P_2 : T_2 \quad \mathsf{label}(T_1) \notin \{par, nil\} \quad \mathsf{label}(T_2) \notin \{par, nil\}}{\psi \vdash_n P_1 \,|\, P_2 : (par; T_1, T_2)}$$

$$\text{(t5)}\ \frac{\psi \vdash_n P_i : T_i \quad \mathsf{label}(T_2) = nil}{\psi \vdash_n P_1 \,|\, P_2 : T_1}, i = 1, 2 \qquad \text{(t6)}\ \frac{\psi \vdash_n P_i : T_i \quad \mathsf{label}(T_1) = nil}{\psi \vdash_n P_1 \,|\, P_2 : T_2}, i = 1, 2$$

$$\text{(t7)}\ \frac{\psi \vdash_n M_1 : (choice; T_1, \cdots, T_m) \quad \psi \vdash_n M_2 : T_0 \quad \mathsf{label}(T_2) \notin \{choice, nil\}}{\psi \vdash_n M_1 + M_2 : (choice; T_1, \cdots, T_m, T_0)}$$

$$\text{(t8)}\ \frac{\psi \vdash_n M_1 : T_0 \quad \psi \vdash_n M_2 : (choice; T_1, \cdots, T_m) \quad \mathsf{label}(T_1) \notin \{choice, nil\}}{\psi \vdash_n M_1 + M_2 : (choice; T_0, T_1, \cdots, T_m)}$$

$$\text{(t9)}\ \frac{\psi \vdash_n M_1 : (choice; T_1, \cdots, T_m) \quad \psi \vdash_n M_2 : (choice; T_{m+1}, \cdots, T_k)}{\psi \vdash_n M_1 + M_2 : (choice; T_1, \cdots, T_k)}$$

$$\text{(t10)}\ \frac{\psi \vdash_n M_1 : T_1 \quad \psi \vdash_n M_2 : T_2 \quad \mathsf{labe}(T_1) \notin \{choice, nil\} \quad \mathsf{label}(T_2) \notin \{choice, nil\}}{\psi \vdash_n M_1 + M_2 : (choice; T_1, T_2)}$$

$$\text{(t11)}\ \frac{\psi \vdash_n M_i : T_i \quad \mathsf{label}(T_2) = nil}{\psi \vdash_n M_1 + M_2 : T_1}, i = 1, 2 \quad \text{(t12)}\ \frac{\psi \vdash_n M_i : T_i \quad \mathsf{label}(T_1) = nil}{\psi \vdash_n M_1 + M_2 : T_2}, i = 1, 2$$

$$\text{(t13)}\ \frac{\psi \vdash_n \langle C \rangle \pi.\, P : T}{\psi \vdash_n *\langle C \rangle \pi.\, P : (rep; T)} \quad \text{(t14)}\ \psi \vdash_n \mathsf{nil} : (nil)$$

$$\text{(t15)}\ \frac{\psi \vdash_n C : T_1 \quad \psi, x : n \vdash_{n+1} P : T_2}{\psi \vdash_n \langle C \rangle x?y.\, P : (act; (\hat{\psi}(x)?n), T_1, T_2)}$$

$$\text{(t16)}\ \frac{\psi \vdash_n C : T_1 \quad \psi \vdash_n P : T_2}{\psi \vdash_n \langle C \rangle \mathsf{ch}(x, \Delta, p).\, P : (act; (\hat{\psi}(x)\Delta), T_1, T_2)}$$

$$\text{(t17)}\ \frac{\psi \vdash_n C : T_1 \quad \psi \vdash_n P : T_2}{\psi \vdash_n \langle C \rangle x!y.\, P : (act; (\hat{\psi}(x)!\hat{\psi}(y)), T_1, T_2)}$$

Table 4.9: Axioms and rules to generate nameless tree representations of processes.

$$\text{(t18)}\ \frac{\psi \vdash_n C_1 : (and; T_1, \cdots, T_m) \quad \psi \vdash_n C_2 : T_0 \quad \mathsf{label}(T_0) \neq and}{\psi \vdash_n C_1 \wedge C_2 : (and; T_1, \cdots, T_m, T_0)}$$

$$\text{(t19)}\ \frac{\psi \vdash_n C_1 : T_0 \quad \psi \vdash_n C_2 : (and; T_1, \cdots, T_m) \quad \mathsf{label}(T_0) \neq and}{\psi \vdash_n C_1 \wedge C_2 : (and; T_0, T_1, \cdots, T_m)}$$

$$\text{(t20)}\ \frac{\psi \vdash_n C_1 : (par; T_1, \cdots, T_m) \quad \psi \vdash_n C_2 : (par; T_{m+1}, \cdots, T_k)}{\psi \vdash_n C_1 \wedge C_2 : (and; T_1, \cdots, T_k)}$$

$$\text{(t21)}\ \frac{\psi \vdash_n C_1 : T_1 \quad \psi \vdash_n C_2 : T_2 \quad \mathsf{label}(T_1) \neq and \quad \mathsf{label}(T_2) \neq and}{\psi \vdash_n C_1 \wedge C_2 : (and; T_1, T_2)}$$

$$\text{(22)}\ \frac{\psi \vdash_n C_1 : (and; T_1, \cdots, T_m) \quad \psi \vdash_n C_2 : T_0 \quad \mathsf{label}(T_0) \neq or}{\psi \vdash_n C_1 \vee C_2 : (or; T_1, \cdots, T_m, T_0)}$$

$$\text{(t23)}\ \frac{\psi \vdash_n C_1 : T_0 \quad \psi \vdash_n C_2 : (and; T_1, \cdots, T_m) \quad \mathsf{label}(T_0) \neq or}{\psi \vdash_n C_1 \vee C_2 : (or; T_0, T_1, \cdots, T_m)}$$

$$\text{(t24)}\ \frac{\psi \vdash_n C_1 : (par; T_1, \cdots, T_m) \quad \psi \vdash_n C_2 : (par; T_{m+1}, \cdots, T_k)}{\psi \vdash_n C_1 \vee C_2 : (or; T_1, \cdots, T_k)}$$

$$\text{(t25)}\ \frac{\psi \vdash_n C_1 : T_1 \quad \psi \vdash_n C_2 : T_2 \quad \mathsf{label}(T_1) \neq or \quad \mathsf{label}(T_2) \neq or}{\psi \vdash_n C_1 \vee C_2 : (or; T_1, T_2)}$$

$$\text{(t26)}\ \psi \vdash_n (x, \Gamma) : (\hat{\psi}(x)\Gamma) \quad \text{(t27)}\ \psi \vdash_n (x, K) : (\hat{\psi}(x)K) \quad \text{(t28)}\ \psi \vdash_n true : (true)$$

Table 4.10: Axioms and rules to generate nameless tree representations of conditions.

representation in the guarded process P, the input invokes the construction of the subtree corresponding to P by adding a new binding in the scope with actual binding depth size and by increasing the binding depth. In this way, the binding variable y (which can be change at hand by α-conversion) is abstractly represented by the number n indicating the actual binding depth. Below, indeed, we prove that α-convertible processes have the same tree representation. This fact simplifies our treatment of $\equiv_p^{\mathbf{rep}}$ in terms of trees comparison.

Tab. 4.10 refers to conditions. Rules (t18-21) allow the compression in a unique level of multiple and compositions. Rules (t22-25), instead, allow the compression in a unique level of multiple or compositions. Finally, rules (t26-28) are used to represent atoms of conditions.

Note that, given a process P, a ψ and an n, the three representation $st(P)_n^\psi$ is unique.

Lemma 4.3.9. *Let $C \in \mathcal{C}$, $\psi \in \Psi$ and $n \in \mathbb{N}$. Then $\psi \vdash_n C : T_1$ and $\psi \vdash_n C : T_2$ implies*

$T_1 = T_2$.

Proof. By induction on the derivation of $\psi \vdash_n C : T_1$. □

Lemma 4.3.10. *Let $P \in \mathcal{P}$, $\psi \in \Psi$ and $n \in \mathbb{N}$. Then $\psi \vdash_n P : T_1$ and $\psi \vdash_n P : T_2$ implies $T_1 = T_2$.*

Proof. By induction on the derivation of $\psi \vdash_n P : T_1$. □

Now, we prove a series of Lemmas that allow to prove that tree representations are equal up-to α-conversion of corresponding processes.

Lemma 4.3.11. *Let $C \in \mathcal{C}$, and $\psi, \psi' \in \Psi$ and $x, y \in \mathcal{N}$. Then $\psi(z) = \psi'(z)$ for each $z \in \mathsf{fn}(C)$ implies $st(C)^{\psi}_n = st(C)^{\psi'}_n$.*

Proof. By induction on the structure of C. Case *true* is immediate. Since $\mathsf{fn}((x, \Delta)) = \mathsf{fn}((x, K)) = \{x\}$ then these two cases follow immediately from the fact that $\psi(z) = \psi'(z)$ for each $z \in \mathsf{fn}(C)$. Moreover, cases $C_1 \wedge C_2$ and $C_1 \vee C_2$ follow immediately from the inductive hypothesis and free name definition. □

Lemma 4.3.12. *Let $P \in \mathcal{P}$, and $\psi, \psi' \in \Psi$ and $x, y \in \mathcal{N}$. Then $\psi(z) = \psi'(z)$ for each $z \in \mathsf{fn}(P)$ implies $st(P)^{\psi}_n = st(P)^{\psi'}_n$.*

Proof. By induction on the structure of P. Case nil is immediate while cases $*\langle C\rangle\pi.\, Q$, $Q_1 \,|\, Q_2$ and $M_1 + M_2$ follow immediately from the inductive hypothesis and free name definition.

(case $P = \langle C\rangle x?y.Q$)

We have $st(P)^{\psi}_n = (\hat{\psi}(x)?n; T_1, T_2)$ and $st(P)^{\psi'}_n = (\hat{\psi'}(x)?n; T'_1, T'_2)$ where $T_2 = st(Q)^{\psi, y:n}_{n+1}$ and $T'_2 = st(Q)^{\psi', y:n}_{n+1}$. By hypothesis $\hat{\psi}(x)?n = \hat{\psi'}(x)?n$ because either $\hat{\psi}(x) = x = \hat{\psi'}(x)$ (in case $x \notin dom(\psi)$) or $\hat{\psi}(x) = m = \hat{\psi'}(x)$ with $m \in \mathbb{N}$. Moreover, by Lemma 4.3.11, $T_1 = T'_1$ and since $(\psi, y : n)(z) = (\psi', y : n)(z)$ for each $z \in \mathsf{fn}(Q)$, then by inductive hypothesis $T_2 = T'_2$. Hence $st(P)^{\psi}_n = st(P)^{\psi'}_n$;

(case $P = \langle C\rangle x!y.Q$)

We have that $st(P)^{\psi}_n = (\hat{\psi}(x)!\hat{\psi}(y); T_1, T_2)$ and $st(P)^{\psi'}_n = (\hat{\psi'}(x)!\hat{\psi}(y); T'_1, T'_2)$ where $T_2 = st(Q)^{\psi}_n$ and $T'_2 = st(Q)^{\psi'}_n$. By hypothesis we have $\hat{\psi}(x)!\hat{\psi}(y) = \hat{\psi'}(x)!\hat{\psi'}(y)$ because either $\hat{\psi}(x) = x = \hat{\psi'}(x)$ (in case $x \notin dom(\psi)$) or $\hat{\psi}(x) = m = \hat{\psi'}(x)$ with $m \in \mathbb{N}$; the same holds for y. Moreover, by Lemma 4.3.11, $T_1 = T'_1$ and since $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P)$, then by inductive hypothesis $T_2 = T'_2$. Hence $st(P)^{\psi}_n = st(P)^{\psi'}_n$;

(case $P = \langle C\rangle \mathsf{ch}(x, \Delta, p).Q$)

We have that $st(P)^{\psi}_n = (\hat{\psi}(x)\Delta; T_1, T_2)$ and $st(P)^{\psi'}_n = (\hat{\psi'}(x)\Delta; T'_1, T'_2)$ where $T_2 = st(Q)^{\psi}_n$ and $T'_2 = st(Q)^{\psi'}_n$. By hypothesis $\hat{\psi}(x)\Delta = \hat{\psi'}(x)\Delta$ because either $\hat{\psi}(x) = x = \hat{\psi'}(x)$

(in case $x \notin dom(\psi)$) or $\hat{\psi}(x) = m = \hat{\psi'}(x)$ with $m \in \mathbb{N}$. Moreover, by Lemma 4.3.11, $T_1 = T_1'$ and since $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P)$, then by inductive hypothesis $T_2 = T_2'$. Hence $st(P)_n^{\psi} = st(P)_n^{\psi'}$. □

Lemma 4.3.13. *Let $C \in \mathcal{C}$, and $\psi, \psi' \in \Psi$ and $x, y \in \mathcal{N}$. Then $\psi(z_1) = \psi'(z_1)$ for each $z_1 \in \mathsf{fn}(C) \setminus \{x\}$ and $\psi(x) = \psi'(y) \neq \perp$ and $y \notin \mathsf{n}(C)$ implies $st(C)_n^{\psi} = st(C\{y/x\})_n^{\psi'}$.*

Proof. By induction on the structure of C. Case *true* is immediate. Since $\mathsf{fn}((x, \Delta)) = \mathsf{fn}((x, K)) = \{x\}$, and $(x, \Delta)\{x/y\} = (y, \Delta)$, and $(x, K)\{x/y\} = (y, K)$, then these two cases follow immediately from the fact that $\psi(x) = \psi'(y) \neq \perp$. Moreover, cases $C_1 \wedge C_2$ and $C_1 \vee C_2$ follow immediately from the inductive hypothesis, free name definition and substitution definition. □

Lemma 4.3.14. *Let $P \in \mathcal{P}$, and $\psi, \psi' \in \Psi$ and $x, y \in \mathcal{N}$. Then $\psi(z_1) = \psi'(z_1)$ for each $z_1 \in \mathsf{fn}(P) \setminus \{x\}$, and $\psi(x) = \psi'(y) \neq \perp$, and $y \notin \mathsf{n}(P)$ implies $st(P)_n^{\psi} = st(P\{y/x\})_n^{\psi'}$.*

Proof. By induction on the structure of P.
(case $\langle C \rangle z?k.Q$)
We know $\mathsf{fn}(P) = \mathsf{fn}(C) \cup \{z\} \cup (\mathsf{fn}(Q) \setminus \{k\})$. There are four subcases:
($z = x \neq k$) We have $P\{y/x\} = \langle C\{y/x\} \rangle y?k.Q\{y/x\}$. Given the definition of free names we know $(\psi, k : n)(z_1) = (\psi', k : n)(z_1)$ for each $z_1 \in \mathsf{fn}(Q) \setminus \{x\}$, and $(\psi, k : n)(x) = (\psi', k : n)(y)$, and $y \notin \mathsf{n}(Q)$; hence, by inductive hypothesis we have $st(Q)_{n+1}^{\psi, k:n} = st(Q\{y/x\})_{n+1}^{\psi', k:n}$. Moreover, by hypothesis and Lemma 4.3.13 we have $st(C)_n^{\psi} = st(C\{y/x\})_n^{\psi'}$ and by hypothesis $\hat{\psi}(x)?\hat{\psi}(k) = \hat{\psi'}(y)?\hat{\psi'}(k)$. Hence $st(P)_n^{\psi} = st(\langle C\{y/x\} \rangle y?k.Q\{y/x\})_n^{\psi'}$, that means $st(P)_n^{\psi} = st(P\{y/x\})_n^{\psi'}$ by definition of substitution;
($z \neq x \neq k$) We have $P\{y/x\} = \langle C\{y/x\} \rangle z?k.Q\{y/x\}$. Given the definition of free names we know $(\psi, k : n)(z_1) = (\psi', k : n)(z_1)$ for each $z_1 \in \mathsf{fn}(Q) \setminus \{x\}$, and $(\psi, k : n)(x) = (\psi', k : n)(y)$, and $y \notin \mathsf{n}(Q)$; hence, by inductive hypothesis we have $st(Q)_{n+1}^{\psi, k:n} = st(Q\{y/x\})_{n+1}^{\psi', k:n}$. Moreover, by hypothesis and Lemma 4.3.13 we have $st(C)_n^{\psi} = st(C\{y/x\})_n^{\psi'}$ and by hypothesis $\hat{\psi}(z)?\hat{\psi}(k) = \hat{\psi'}(z)?\hat{\psi'}(k)$. Hence $st(P)_n^{\psi} = st(\langle C\{y/x\} \rangle y?k.Q\{y/x\})_n^{\psi'}$, that means $st(P)_n^{\psi} = st(P\{y/x\})_n^{\psi'}$ by definition of substitution;
($z = x = k$) We have $P\{y/x\} = \langle C\{y/x\} \rangle y?k.Q$. Given the definition of free names we know $(\psi, k : n)(z_1) = (\psi', k : n)(z_1)$ for each $z_1 \in \mathsf{fn}(Q)$, then by Lemma 4.3.12 we have $st(Q)_{n+1}^{\psi, k:n} = st(Q)_{n+1}^{\psi', k:n}$. By hypothesis and Lemma 4.3.13 we have $st(C)_n^{\psi} = st(C\{y/x\})_n^{\psi'}$ and by hypothesis $\hat{\psi}(x)?\hat{\psi}(k) = \hat{\psi'}(y)?\hat{\psi'}(k)$. Hence we can derive $st(P)_n^{\psi} = st(\langle C\{y/x\} \rangle y?k.Q\{y/x\})_n^{\psi'}$, that means $st(P)_n^{\psi} = st(P\{y/x\})_n^{\psi'}$ by definition of substitution;
($z \neq x = k$) We have $P\{y/x\} = \langle C\{y/x\} \rangle z?k.Q$. Given the definition of free names we know $(\psi, k : n)(z_1) = (\psi', k : n)(z_1)$ for each $z_1 \in \mathsf{fn}(Q)$, then by Lemma 4.3.12 we have $st(Q)_{n+1}^{\psi, k:n} = st(Q)_{n+1}^{\psi', k:n}$. By hypothesis and Lemma 4.3.13 we have $st(C)_n^{\psi} =$

$st(C\{y/x\})^{\psi'}_n$ and by hypothesis $\hat{\psi}(z)?\hat{\psi}(k) = \hat{\psi'}(z)?\hat{\psi'}(k)$. Hence we can derive $st(P)^{\psi}_n = st(\langle C\{y/x\}\rangle z?k.Q)^{\psi'}_n$, that means $st(P)^{\psi}_n = st(P\{y/x\})^{\psi'}_n$ by definition of substitution.

The case nil is immediate, cases $\langle C\rangle x?y.\, Q$ and $\langle C\rangle \mathsf{ch}(x, \Delta, p).\, Q$ can be proven similarly to the previous case and all the other cases follow by names and free names definition, substitution definition and inductive hypothesis. □

Lemma 4.3.15. *Let $P_1, P_2 \in \mathcal{P}$, $\psi \in \Psi$ and $n \in \mathbb{N}$. Then $P_1 =_\alpha P_2$ implies $st(P_1)^{\psi}_n = st(P_2)^{\psi}_n$.*

Proof. By α-conversion definition, P_2 can be obtained from P_1 by a finite number of changes of bound variables. We have hence processes $P_1 = P^1, P^2, P^3, \cdots, P^m = P_2$ such that P^i differs from P^{i+1} (with $1 \leq i < m$) only by a change of a bound variable. Each of these changes corresponds to a replacement of a subterm $\langle C\rangle x?y.\, Q^i$ of P^i by $\langle C\rangle x?w.\, Q^i\{w/y\}$ where w does not occur in P^{i+1}. From a tree representation perspective, instead, each of this changes corresponds to a replacement of a subtree $st(\langle C\rangle x?y.\, Q^i)^{\psi'}_m$ by $st(\langle C\rangle x?w.\, Q^i\{w/y\})^{\psi'}_m$ in $st(P^i)^{\psi}_n$. Given the derivation of $st(\langle C\rangle x?y.\, Q^i)^{\psi'}_m$ we know $st(Q^i)^{\psi',y:m}_{m+1}$ and by considering $\psi', w : m$ we notice that $(\psi', y : m)(z) = (\psi', w : m)(z)$ for each $z \in \mathsf{fn}(Q^i) \setminus \{w\}$, and $\psi(w) = \psi'(y) \neq \bot$; moreover we know $w \notin \mathsf{n}(Q^i)$. By Lemma 4.3.14 we have that $st(Q^i)^{\psi',y:m}_{m+1} = st(Q^i\{w/y\})^{\psi',y:m}_{m+1}$ and hence $st(\langle C\rangle x?y.\, Q^i)^{\psi'}_m = st(\langle C\rangle x?w.\, Q^i\{w/y\})^{\psi'}_m$. This means that for each change of bound variable that leads from P^i to P^{i+1} there is no change in the structure of the corresponding $st(P^i)^{\psi}_n$. Hence $st(P^1)^{\psi}_n = \cdots = st(P^m)^{\psi}_n$, which means $st(P_1)^{\psi}_n = st(P_2)^{\psi}_n$. □

Lemma 4.3.16. *Let $C_1, C_2 \in \mathcal{C}$. Then $C_1 \equiv^{\mathbf{rep}}_p C_2$ iff $st(C_1)^{\psi}_n \simeq st(C_2)^{\psi}_n$.*

Proof. ($\Rightarrow$) By induction on the length of the derivation of $C_1 \equiv^{\mathbf{rep}}_p C_2$.

(case Tab. 4.6(a.8))

We have $C_1 = C'_1 \wedge C'_2 \equiv_p C'_2 \wedge C'_1 = C_2$. Depending on the last rule we apply to derive $st(C_1)^{\psi}_n$ we can distinguish five subcases:

(case Tab. 4.10(t18)) We have that $st(C'_1 \wedge C'_2)^{\psi}_n = (and; T_1, \cdots, T_m, T_0)$ where by the rule premises $\psi \vdash_n C'_1 : (and; T_1, \cdots, T_m)$ and $\psi \vdash_n C'_2 : T_0$ and $\mathsf{label}(T_0) \neq and$. Note that this means $st(C'_2 \wedge C'_1)^{\psi}_n = (and; T_0, T_1, \cdots, T_m)$ with $\psi \vdash_n C'_1 : (and; T_1, \cdots, T_m)$ and $\psi \vdash_n C_2 : T_0$ and $\mathsf{label}(T_0) \neq and$. Note that $st(C'_1 \wedge C'_2)^{\psi}_n$ and $st(C'_2 \wedge C'_1)^{\psi}_n$ have identical root labels and that there exists an ordering of their immediate subtrees such that they exactly coincide. Hence the two trees are isomorphic;

All the other cases can be proven similarly

(case Tab. 4.6(a.9))

We have $C_1 = C'_1 \wedge (C'_2 \wedge C'_3) \equiv_p (C'_1 \wedge C'_2) \wedge C'_3 = C_2$. Depending on the rules we apply to derive $st(C'_2 \wedge C'_3)^{\psi}_n$ and $st(C_1)^{\psi}_n$ we can distinguish eight subcases:

(case Tab. 4.10(t18-t19)) We have that $st(C'_1 \wedge (C'_2 \wedge C'_3))^{\psi}_n = (and; T_0, T_1, \cdots, T_m)$ where by the rule premises $\psi \vdash_n C'_1 : T_0$ and $\psi \vdash_n C'_2 \wedge C'_3 : (and; T_1, \cdots, T_m)$ and $\mathsf{label}(T_0) \neq and$. Moreover we know $\psi \vdash_n C'_2 : (and; T_1, \cdots, T_{m-1})$ and $\psi \vdash_n C'_3 : T_m$ with $\mathsf{label}(T_3) \neq and$. This means $\psi \vdash_n C'_1 \wedge C'_2 : (and; T_0, T_1, \cdots, T_{m-1})$ and $\psi \vdash_n C'_3 : T_m$ and $\mathsf{label}(T_3) \neq and$ from which we have $st((C'_1 \wedge C'_2) \wedge C'_3)^{\psi}_n = (and; T_0, T_1, \cdots, T_m)$. Note that $st(C'_1 \wedge (C'_2 \wedge C'_3))^{\psi}_n$ and $st((C'_1 \wedge C'_2) \wedge C'_3)^{\psi}_n$ are identical. Hence the two trees are isomorphic. All the other cases can be proven similarly.

Cases Tab. 4.6(a.10-11) can be proven similarly to the previous cases. Note that since $\equiv_p^{\mathbf{rep}}$ is a congruence, we have also to consider the implicit cases $C_1 \equiv_p^{\mathbf{rep}} C'_1 \Rightarrow C_1 \wedge C_2 \equiv_p^{\mathbf{rep}} C'_1 \wedge C_2$ and $C_1 \equiv_p^{\mathbf{rep}} C'_1 \Rightarrow C_1 \vee C_2 \equiv_p^{\mathbf{rep}} C'_1 \vee C_2$, and the equational cases regarding reflexivity, symmetry and transitivity. All these cases follow immediately by inductive hypothesis and by noticing that also the tree isomorphism $\simeq$ is an equivalence relation.

($\Leftarrow$) By induction on the derivation of $st(C_1)^{\psi}_n$. Most of the cases use implicitly the fact that $\equiv_p^{\mathbf{rep}}$ is a congruence relation.

(case $(and; T_1, \cdots, T_m)$**)**

Since $st(C_1)^{\psi}_n \simeq st(C_2)^{\psi}_n$, we have that $st(C_2)^{\psi}_n = (and; T'_1, \cdots, T'_m)$ and that there exists a permutation γ such that $T_i \simeq T'_{\gamma(i)}$, for $1 \leq i \leq m$. Moreover, given the structure of the derivation of $st(C_1)^{\psi}_n$ and $st(C_2)^{\psi}_n$ we know that $C_1 \equiv_p^{\mathbf{rep}} \bigwedge_{i=1}^m C'_i$ and $C_2 \equiv_p^{\mathbf{rep}} \bigwedge_{i=1}^m C''_i$ where $st(C'_i)^{\psi}_n = T_i$, $st(C''_i)^{\psi}_n = T'_i$ and strings $\mathsf{label}(T_i)$ and $\mathsf{label}(T_i)$ are not equal to *and*. By inductive hypothesis we have that $C'_i \equiv_p^{\mathbf{rep}} C''_{\gamma(i)}$, for $1 \leq i \leq m$, from which we can easily recover $C_1 \equiv_p^{\mathbf{rep}} C_2$.

Cases $(a\Delta)$ and (aK) follow immediately from the fact that their tree representations are identical and are nodes. Moreover, case $(or; T_1, \cdots, T_m)$ can be proven similarly to the *and* case. □

Lemma 4.3.17. *Let $P_1, P_2 \in \mathcal{P}$. Then $P_1 \equiv_p^{\boldsymbol{rep}} P_2$ iff $st(P_1)^{\psi}_n \simeq st(P_2)^{\psi}_n$.*

Proof. ($\Rightarrow$) By induction on the length of the derivation of $P_1 \equiv_p^{\mathbf{rep}} P_2$.

(case Tab. 4.6(a.1))

We have $P_1 =_\alpha P_2$. From Lemma 4.3.15 we have $st(P_1)^{\psi}_n = st(P_2)^{\psi}_n$ and hence the two trees are straightforwardly isomorphic.

(case Tab. 4.6(a.2))

We have $P_1 = Q_1 \,|\, \mathsf{nil} \equiv_p Q_1 = P_2$. We have $st(\mathsf{nil})^{\psi}_n = (nil)$. The only rule we can apply is Tab. 4.9(t5), obtaining $st(Q_1 \,|\, \mathsf{nil})^{\psi}_n = st(Q_1)^{\psi}_n$. Hence $st(P_1)^{\psi}_n = st(P_2)^{\psi}_n$.

(case Tab. 4.6(a.3))

We have $P_1 = Q_1 \,|\, Q_2 \equiv_p Q_2 \,|\, Q_1 = P_2$. Depending on the last rule we apply to derive $st(P_1)^{\psi}_n$ we can distinguish five subcases:

(case Tab. 4.9(t1)) We have that $st(Q_1 \,|\, Q_2)^{\psi}_n = (par; T_1, \cdots, T_m, T_0)$ where by the rule premises $\psi \vdash_n Q_1 : (par; T_1, \cdots, T_m)$ and $\psi \vdash_n Q_2 : T_0$ and $\mathsf{label}(T_0) \notin \{par, nil\}$. Note that this means $st(Q_2 \,|\, Q_1)^{\psi}_n = (par; T_0, T_1, \cdots, T_m)$ with $\psi \vdash_n Q_1 : (par; T_1, \cdots, T_m)$ and $\psi \vdash_n Q_2 : T_0$ and $\mathsf{label}(T_0) \notin \{par, nil\}$. It is easy to see that $st(Q_1 \,|\, Q_2)^{\psi}_n$ and $st(Q_2 \,|\, Q_1)^{\psi}_n$ have identical root labels and that there exists and ordering of their immediate subtrees such that they exactly coincide. Hence the two trees are isomorphic;
(case Tab. 4.9(t5)) We have that $Q_2 = \mathsf{nil}$. Hence $st(Q_2)^{\psi}_n = (nil)$ from which we have $st(Q_1 \,|\, Q_2)^{\psi}_n = st(Q_1)^{\psi}_n$. This means $st(Q_2 \,|\, Q_1)^{\psi}_n = st(Q_1)^{\psi}_n$, and hence $st(P_1)^{\psi}_n = st(P_2)^{\psi}_n$.
All the other subcases can be proven similarly to case Tab. 4.9(t1).
(case Tab. 4.6(a.4))
We have $P_1 = Q_1 \,|\, (Q_2 \,|\, Q_3) \equiv_p (Q_1 \,|\, Q_2) \,|\, Q_3 = P_2$. Depending on the rule we apply to derive $st(Q_2 \,|\, Q_3)^{\psi}_n$ and $st(P_1)^{\psi}_n$ we can distinguish twelve subcases:
(case Tab. 4.9(tr1-tr2)) We have that $st(Q_1 \,|\, (Q_2 \,|\, Q_3))^{\psi}_n = (par; T_0, T_1, \cdots, T_m)$ where by the rule premises $\psi \vdash_n Q_1 : T_0$, $\psi \vdash_n Q_2 \,|\, Q_3 : (par; T_1, \cdots, T_m)$ and $\mathsf{label}(T_0) \neq par$. Moreover, we have $st(Q_2)^{\psi}_n = (par : T_1, \cdots, T_{m-1})$ and $st(Q_3)^{\psi}_n = T_m$ and $\mathsf{label}(T_m) \notin \{par, nil\}$. Hence, we can derive $\psi \vdash_n Q_1 \,|\, Q_2 : (par; T_0, T_1, \cdots, T_{m-1})$ that combined with $\psi \vdash_n Q_3 : T_m$ and $\mathsf{label}(T_m) \notin \{par, nil\}$ allows to conclude $st((Q_1 \,|\, Q_2) \,|\, Q_3)^{\psi}_n = (par; T_1, \cdots, T_m, T_0)$. Note that $st(Q_1 \,|\, (Q_2 \,|\, Q_3))^{\psi}_n$ and $st((Q_1 \,|\, Q_2) \,|\, Q_3)^{\psi}_n$ are identical. Hence the two trees are isomorphic.
All the other subcases can be proven similarly.

Cases Tab. 4.6(a.5-7) can be proven in a similar way. Moreover, since $\equiv^{\mathbf{rep}}_p$ is a congruence, we have also to consider the implicit cases $P_1 \equiv^{\mathbf{rep}}_p P_1' \Rightarrow P_1 \,|\, P_2 \equiv^{\mathbf{rep}}_p P_1' \,|\, P_2$, $M_1 \equiv^{\mathbf{rep}}_p M_1' \Rightarrow M_1 + M_2 \equiv^{\mathbf{rep}}_p M_1' + M_2$, $C \equiv^{\mathbf{rep}}_p C' \wedge P \equiv^{\mathbf{rep}}_p P' \Rightarrow \langle C \rangle \pi . P \equiv^{\mathbf{rep}}_p \langle C' \rangle \pi . P'$ and $\langle C \rangle \pi . P \equiv^{\mathbf{rep}}_p \langle C' \rangle \pi' . P' \Rightarrow * \langle C \rangle \pi . P \equiv^{\mathbf{rep}}_p * \langle C' \rangle \pi' . P'$, and the equational cases regarding reflexivity, symmetry and transitivity. All these cases follow by inductive hypothesis, by considering the previous Lemma and by noticing that also the tree isomorphism $\simeq$ is an equivalence relation.
($\Leftarrow$) By induction on the derivation of $st(P_1)^{\psi}_n$. Most of the cases use implicitly the fact that $\equiv^{\mathbf{rep}}_p$ is a congruence relation.
(case $(par; T_1, \cdots, T_m)$**)**
Since $st(P_1)^{\psi}_n \simeq st(P_2)^{\psi}_n$, we have that $st(P_2)^{\psi}_n = (par; T_1', \cdots, T_m')$ and that there exists a permutation γ such that $T_i \simeq T'_{\gamma(i)}$, for $1 \leq i \leq m$. Moreover, given the structure of the derivation of $st(P_1)^{\psi}_n$ and $st(P_2)^{\psi}_n$ we know that $P_1 \equiv^{\mathbf{rep}}_p \prod^m_{i=1} Q_i$ and $P_2 \equiv^{\mathbf{rep}}_p \prod^m_{i=1} R_i$ such that $st(Q_i)^{\psi}_n = T_i$, $st(R_i)^{\psi}_n = T_i'$ and $\mathsf{label}(T_i)$ and strings $\mathsf{label}(T_i)$ are not equal to par and nil. By inductive hypothesis we have that $Q_i \equiv^{\mathbf{rep}}_p R_{\gamma(i)}$, for $1 \leq i \leq m$, from which we can easily recover $P_1 \equiv^{\mathbf{rep}}_p P_2$;
(case $(rep; T)$**)**

Since $st(P_1)^{\psi}_n \simeq st(P_2)^{\psi}_n$, we have that $st(P_2)^{\psi}_n = (rep : T')$ and that $T \simeq T'$. Moreover, given the structure of the derivation of $st(P_1)^{\psi}_n$ and $st(P_2)^{\psi}_n$ we know that $P_1 = *\langle C\rangle\pi.\, Q$ and $P_2 = *\langle C'\rangle\pi'.\, Q'$ with $st(\langle C\rangle\pi.\, Q)^{\psi}_n = T$ and $st(\langle C'\rangle\pi'.\, Q')^{\psi}_n = T'$. By inductive hypothesis we have $\langle C\rangle\pi.\, Q \equiv^{\mathbf{rep}}_p \langle C'\rangle\pi'.\, Q'$ and consequently $P_1 \equiv^{\mathbf{rep}}_p P_2$;
(case $(act; (\hat{\psi}(x)\Delta), T_1, T_2)$**)**
Since $st(P_1)^{\psi}_n \simeq st(P_2)^{\psi}_n$, we have that $st(P_2)^{\psi}_n = (act; (\hat{\psi}(x)\Delta), T'_1, T'_2)$ and that $T_1 \simeq T'_1$ and $T_2 \simeq T'_2$. Given the derivations of $st(P_1)^{\psi}_n$ and $st(P_2)^{\psi}_n$ we know that $P_1 = \langle C\rangle\mathsf{ch}(x, \Delta, p).\, Q$ and $P_2 = \langle C'\rangle\mathsf{ch}(x, \Delta, p).\, Q'$ with $st(C)^{\psi}_n = T_1$, $st(Q)^{\psi}_n = T_2$, $st(C')^{\psi}_n = T'_1$ and $st(Q')^{\psi}_n = T'_2$. By Lemma 4.3.16 we have $C \equiv^{\mathbf{rep}}_p C'$ and by inductive hypothesis we have $Q \equiv^{\mathbf{rep}}_p Q'$, from which it follows $P_1 \equiv^{\mathbf{rep}}_p P_2$;
(case $(act; (a!b), T_1, T_2)$**)**
Since $st(P_1)^{\psi}_n \simeq st(P_2)^{\psi}_n$, we have that $st(P_2)^{\psi}_n = (act; (a!b), T'_1, T'_2)$ and that $T_1 \simeq T'_1$ and $T_2 \simeq T'_2$. Given the derivations of $st(P_1)^{\psi}_n$ and $st(P_2)^{\psi}_n$ we know that $P_1 = \langle C\rangle x!y.\, Q$ and $P_2 = \langle C'\rangle x!y.\, Q'$ with $a = \hat{\psi}(x)$, $b = \hat{\psi}(y)$, $st(C)^{\psi}_n = T_1$, $st(Q)^{\psi}_n = T_2$, $st(C')^{\psi}_n = T'_1$ and $st(Q')^{\psi}_n = T'_2$. By Lemma 4.3.16 we have $C \equiv^{\mathbf{rep}}_p C'$ and by inductive hypothesis we have $Q \equiv^{\mathbf{rep}}_p Q'$, from which it immediately follows $P_1 \equiv^{\mathbf{rep}}_p P_2$;
(case $(act; (a?n), T_1, T_2)$**)**
Since $st(P_1)^{\psi}_n \simeq st(P_2)^{\psi}_n$, we have that $st(P_2)^{\psi}_n = (act; (a?n), T'_1, T'_2)$ and that $T_1 \simeq T'_1$ and $T_2 \simeq T'_2$. Given the derivations of $st(P_1)^{\psi}_n$ and $st(P_2)^{\psi}_n$ we know that $P_1 = \langle C\rangle x?y.\, Q$ and $P_2 = \langle C'\rangle x?z.\, Q'$ with $a = \hat{\psi}(x)$, $st(C)^{\psi}_n = T_1$, $st(Q)^{\psi,y:n}_{n+1} = T_2$, $st(C')^{\psi}_n = T'_1$ and $st(Q')^{\psi,z:n}_{n+1} = T'_2$. By Lemma 4.3.16, we have $C \equiv^{\mathbf{rep}}_p C'$. Moreover, by Lemma 4.3.15, we know that by considering the process $P'_1 = \langle C\rangle x?z.\, Q\{z/y\} =_{\alpha} P_1$, we have $st(P_1)^{\psi}_n = st(P'_1)^{\psi}_n$, and hence $st(P'_1)^{\psi}_n \simeq st(P_2)^{\psi}_n$. From the premises of the two trees derivations we obtain $T_2 = st(Q\{z/y\})^{\psi,z:n}_{n+1}$ and as a consequence $st(Q\{z/y\})^{\psi,z:n}_{n+1} \simeq T'_2$. By inductive hypothesis we obtain $Q' \equiv^{\mathbf{rep}}_p Q\{z/y\}$ from which we recover $P_2 \equiv^{\mathbf{rep}}_p P'_1$. Since $P_1 =_{\alpha} P'_1$ then $P_1 \equiv^{\mathbf{rep}}_p P'_1$ and, by transitivity, $P_1 \equiv^{\mathbf{rep}}_p P_2$.

Case $(choice; T_1, \cdots, T_m)$ can be proven similarly to $(par; T_1, \cdots, T_m)$ and case (nil) is easy. □

Since the tree isomorphism problem is efficiently solvable [59], then by the Lemma 4.3.17 the congruence $\equiv^{\mathbf{rep}}_p$ is decidable and efficiently solvable, and hence the function $Impl$ works in polynomial time. Now, combining all the obtained results, we can show that the congruence $\equiv_p$ is efficiently solvable.

Theorem 4.3.18. *$\equiv_p$ is decidable and efficiently solvable.*

Proof. Using Lemma 4.3.6 and Lemma 4.3.17 we know that $\equiv_p$ is polynomially reducible to a labeled tree isomorphism problems. Hence, given $P, P' \in \mathcal{P}$, $\psi \in \Psi$ and $n \in \mathbb{N}$ we

have:

$$P \equiv_p P' \Leftrightarrow Impl(P) \equiv_p^{\mathbf{rep}} Impl(P') \Leftrightarrow st(Impl(P))_n^{\psi} \simeq st(Impl(P'))_n^{\psi}$$

Since $\simeq$ is an efficiently solvable problem, then $\equiv_p$ is decidable and efficiently solvable. □

4.3.2 Recognizing species

As previously said, we established a notion of species that relies on the definition of structural congruence of boxes. We say that two boxes represent biological substances of the same species only if they are structurally congruent up-to boxes identifiers.

Consider two boxes $I[\,P\,]_x$ and $I'[\,P'\,]_x$ in $\overline{\mathcal{S}}$. Given $P \equiv_p P'$, axiom Tab. 4.6(b.4) states that the two boxes are structurally congruent only if I is a permutation of I' up-to renaming of interface subjects. The structural comparison between interfaces and processes implemented by structural congruence can be described through a function $Species(I, P, I', P')$, defined by induction on the structure of processes and interfaces (see Tab. 4.11).

```
SPECIES (I, P, I', P'):
  switch (I, P, I', P')
    case (ε, P, ε, P') :
      if P ≡_p P' then
        return true

      else
        return false

    case (ε, P, I', P') :
      return false
    case (I', P, ε, P') :
      return false
    case (⊕(x, Δ)^p I*, P, I', P') :
      if I' = I1* K(y, Δ)^p I2* ∧ z ∉ fn(P | P') ∪ sub(I* I1* I2*) then
        SPECIES(I*, P{z/x}, I1* I2*, P'{z/y})

      else
        return false
```

Table 4.11: Definition of function Box.

Given interfaces I and I', then there are two different cases: (1) if a sort correspondence between the first interface $K(x,\,\Gamma)^p$ of I and one interface $K(y,\,\Gamma)^p$ of I' exists, then the function $Species$ is recursively invoked on the boxes $I_1[P\{z/x\}]$ and $I_2[P'\{z/y\}]$, where

$z \notin \mathsf{fn}(P \mid P') \cup \mathsf{sub}(I^* I_2^* I_1^*)$. I_1 is obtained from I deleting the interface $K(x, \Gamma)^p$, while I_2 is obtained from I' deleting the interface $K(y, \Gamma)^p$; (2) if no correspondence between the first interface of I and one interface of I' exists, then the function returns *false*.

If during the recursive invocations only one of the interfaces I and I' results empty, then the function returns *false*. If both I and I' are empty, then the function returns *true* if $P \equiv_p P'$, and *false* otherwise.

Definition 4.3.19. *Given interfaces I_1, I_2 and processes P_1, P_2 we write $I_1[P_1]_n \sim_s I_2[P_2]_m$, and also $I_1[P_1] \sim_s I_2[P_2]$, iff $Species(I_1, P_1, I_2, P_2) = true$.*

Lemma 4.3.20. *$I[\,P\,]_x \equiv_b I'[\,P'\,]_x$ iff $I[P] \sim_s I'[P']$.*

Definition 4.3.21. *Let $I_1[\,P_1\,]_{n_1}$ and $I_2[\,P_2\,]_{n_2}$. Then $I_1[\,P_1\,]_{n_1}$ and $I_2[\,P_2\,]_{n_2}$ belong to the same species only if $I_1[P_1] \sim_s I_2[P_2]$.*

The problem of deciding whether two boxes belong to the same species or more generally whether they are structurally congruent, can be reduced to the structural congruence of processes. Moreover, function *Species* provides a directly implementable method to check whether two boxes describe entities of the same species.

4.3.3 Recognizing complexes

As said, the notion of species can be used at the level of complexes to check whether two complexes are made up by the same number of boxes and whether they preserve links over interfaces with identical sorts and over boxes belonging to the same species. Although the described problem corresponds to check whether two complexes are structurally congruent, it is easy to see that it can be rephrased in terms of a kind of graph isomorphism problem.

Generally, a pair (B, ξ) is used to describe a set of complexes. However, since we want to focus on the comparison of two complexes, we first introduce a relation on links that allows to extract all the single complexes from the description (B, ξ).

Definition 4.3.22. *Let $\sim_l$ be a binary relation over environment links that satisfies the axiom: $\{n_1, m_1\} \neq \{n, m\} \wedge \{n_1, m_1\} \cap \{n, m\} \neq \emptyset \Rightarrow \{\Delta n, \Gamma m\} \sim_l \{\Delta_1 n_1, \Gamma_1 m_1\}$. Moreover, let $\sim_c$ be the smallest transitive and symmetric relation over environment links generated by $\sim_l$.*

Note that given ξ, we can use relation $\sim_c$ to partition ξ into equivalence classes containing the links of the different complexes. In particular, given two links l and l' we have that $l \sim_c l'$ iff the two links connect boxes belonging to the same complex. Having the link partition, we can use the identifiers information stored in the links to easily extract

from B the boxes composing the different complexes. Moreover, the relation $\sim_c$ gives also a method to check whether a pair (B, ξ) represents a single complex or a set of complexes.

Definition 4.3.23. *Let B and ξ such that $\vdash_b B : (N, C)$ and $\vdash_{en} \xi : C$. We say that (B, ξ) is a complex if and only if $\forall l, l' \in \xi$ we have $l \sim_c l'$, or equally $(\xi/\sim_c) = \{\xi\}$.*

Definition 4.3.24. *Let (B_1, ξ_1) and (B_2, ξ_2) be complexes. Then (B_1, ξ_1) and (B_2, ξ_2) are isomorphic, $(B_1, \xi_1) \cong (B_2, \xi_2)$, iff there is a bijection φ between $Boxes(B_1)$ and $Boxes(B_2)$ such that $I[\,P\,]_n \in Boxes(B_1)$ implies $I'[\,P'\,]_{\varphi(n)} \in Boxes(B_2)$ and $I[P] \sim_s I'[P']$, and such that $\{\Delta m, \Delta' m'\} \in \xi_1$ iff $\{\Delta\varphi(m), \Delta'\varphi(m')\} \in \xi_2$.*

Note that a notion of complexes enables to define a notion of isomorphism between complexes. This definition rephrases the problem of checking whether two complexes are structurally congruent and indeed the two problems are equivalent.

Theorem 4.3.25. *$(B, \xi) \equiv_c (B', \xi')$ iff $(B, \xi) \cong (B', \xi')$.*

Proof. ($\Rightarrow$) By induction on the length of the derivation $(B, \xi) \equiv_c (B', \xi')$.
(case Tab. 4.6(c.1))
From $B \equiv_b B'$ we know that $B \equiv_b \prod_{i=1}^{k} I_i[\,P_i\,]_{n_i}$ and $B \equiv_b \prod_{i=1}^{k} I'_i[\,P'_i\,]_{n'_i}$ such that for each i we have $I_i[\,P_i\,]_{n_i} \equiv_b I'_i[\,P'_i\,]_{n_i}$. But this means that for each $I[\,P\,]_n \in Boxes(B)$ there exists a corresponding structural congruent box $I'[\,P'\,]_n \in Boxes(B')$, and vice-versa. Moreover, we know $\xi = \xi'$. Hence, by considering the identity bijection φ (i.e., $\varphi(n) = n$ for all $n \in \mathcal{N}$) we have that $(B, \xi) \cong (B', \xi')$;
(case Tab. 4.6(c.2))
We know that $B \equiv_b I[\,P\,]_n \mid \prod_{i=1}^{k} I_i[\,P_i\,]_{n_i}$ and that $B\langle m/n\rangle \equiv_b I[\,P\,]_m \mid \prod_{i=1}^{k} I_i[\,P_i\,]_{n_i}$, with $m \neq n$. Note that $Boxes(B) \setminus \{I[\,P\,]_n\}$ is equal to $Boxes(B') \setminus \{I[\,P\,]_m\}$. Moreover, environments ξ and ξ' differ only in the links containing n. In particular, for each link $\{\Delta n, \Delta' y\} \in \xi$ containing n, there is a corresponding link $\{\Delta m, \Delta' y\} \in \xi'$, and vice-versa (note that by well-formedness we are sure $m \neq y \neq n$). Hence, by considering the bijection φ such that $\varphi(n) = m$ and $\varphi(y) = y$ for all $y \in \mathcal{N} \setminus \{n\}$ it is $(B, \xi) \cong (B', \xi')$;
($\Leftarrow$) From $(B, \xi) \cong (B', \xi')$ we know that $B \equiv_b \prod_{i=1}^{k} I_i[\,P_i\,]_{n_i}$ and $B' \equiv_b \prod_{i=1}^{k} I'_i[\,P'_i\,]_{n'_i}$ such that for each i we have $Species(I, P, I', P') = true$ and such that $\varphi(n_i) = n'_i$. This means that for each i we have $I_i[\,P_i\,]_{\varphi(n_i)} \equiv_b I'_i[\,P'_i\,]_{n'_i}$. Hence $B'' \equiv_b \prod_{i=1}^{k} I_i[\,P_i\,]_{\varphi(n_i)} \equiv_b B'$ from which follows $(B'', \xi') \equiv_c (B', \xi')$. Now consider a set of names $\{y_1, \cdots, y_k\} \cap \mathsf{id}(B'') = \emptyset$. We can build the sequence of relations:

$$\begin{array}{rcl} (B'', \xi') & \equiv_c & (B\langle {}^{y_1,\cdots,y_k}/_{\varphi(n_1),\cdots,\varphi(n_k)}\rangle, \xi\langle {}^{y_1,\cdots,y_k}/_{\varphi(n_1),\cdots,\varphi(n_k)}\rangle \\ & \equiv_c & (B\langle {}^{n_1,\cdots,n_k}/_{y_1,\cdots,y_k}\rangle, \xi\langle {}^{n_1,\cdots,n_k}/_{y_1,\cdots,y_k}\rangle = (B, \xi) \end{array}$$

and by transitivity derive $(B, \xi) \equiv_c (B', \xi')$. □

Having this results we can now focus on our complexes isomorphism problem. In particular, the isomorphism interpretation allows us to deal with the implementation of structural congruence for complexes in a more intuitive way.

The pseudocode of the algorithm is reported in Tab.4.12. The function ISOMORPHISM checks if the two input complexes are isomorphic and uses the function DFSMATCHING. The function SETVISITED sets a flag to remember that the box has been visited and the function ISVISITED checks whether a box has already been visited. Moreover, the function CLEANVISITED resets all the flags.

```
ISOMORPHISM ((B1, ξ1), (B2, ξ2)):
  if |Boxes(B1)| ≠ |Boxes(B2)| then
    return false
  if |ξ1| ≠ |ξ2| then
    return false
  take I[ P ]n in Boxes(B1);
  for each I'[ P' ]n' in Boxes(B2) do
    if I[P] ∼s I'[P'] then
      CLEANVISITED((B1, ξ1))
      CLEANVISITED((B2, ξ2))
      res := DFSMATCHING((B1, ξ1),(B2, ξ2),I[ P ]n,I'[ P' ]n') ;
      if res then
        return true

  return false;


DFSMATCHING ((B1, ξ1),(B2, ξ2),I1[ P1 ]n1,I2[ P2 ]n2):
  if ISVISITED(I1[ P1 ]n1) ≠ ISVISITED(I2[ P2 ]n2) then
    return false
  if ISVISITED(I1[ P1 ]n1) then
    return true
  SETVISITED(I1[ P1 ]n1);
  SETVISITED(I2[ P2 ]n2);
  for each {Δn1, Δ'n'1} in ξ1 do
    if ∃ {Δn2, Δ'n'2} in ξ2 s.t. I'1[P'1] ∼s I'2[P'2]
       with I'1[ P'1 ]n'1 in Boxes(B1) and I'2[ P'2 ]n'2 in Boxes(B2)
     then
      res := DFSMATCHING((B1, ξ1),(B2, ξ2),I'1[ P'1 ]n'1,I'2[ P'2 ]n'2);
      if ¬ res then
        return false;

    else
      return false;
  return true;
```

Table 4.12: Pseudocode of the algorithm that verifies if two complexes are isomorphic.

Note that since our problem is in nature equivalent to a graph isomorphism problem we used a procedure based on the graph deep first search (DFS). We recall that in general the graph isomorphism problem [59] defines the complexity class *GI*, which contains all the problems equivalent to it [6]. No polynomial algorithm for the problems in *GI* has still been found and it is not known if they are or not *NP-complete*. For the class of complexes we consider, however, it results that the complexity of the comparison is polynomial in the number and size of the complexes descriptions.

Theorem 4.3.26. $(B, \xi) \equiv_c (B', \xi')$ *is efficiently solvable.*

Proof sketch. We prove it by showing that $\textsc{Isomorphism}((B, \xi), (B', \xi'))$ is polynomial in the size of the complexes descriptions.

Consider two complexes (B, ξ) and (B', ξ'). For establishing whether they are isomorphic, we first check if the number of boxes and links is the same. If it is the case, we then choose two structurally congruent boxes in B and B'. If we chose randomly the box $I[\, P \,]_x$ for (B, ξ), then we have at most $|Boxes(B)|$ possibilities for choosing the box for (B', ξ'). For each of these initial possibilities, the isomorphism problem results to be a parallel visit of the two complexes. Indeed, due to the well-formedness property of boxes, when in a box of (B, ξ) we choose a link for propagating the check, we have at most one link with same sorts in the corresponding box of (B', ξ') that can connect the two boxes with two other structural congruent boxes. If the parallel visit finishes correctly, then the two complexes are isomorphic, otherwise we have to check another initial possibility. Therefore, since the initial root possibilities are at most $|Boxes(B')|$ and the parallel visit of the complexes has linear complexity in the number of complexes boxes, at the end we have a polynomial complexity in the number of boxes links.

Note that if for all the initial choices in (B', ξ') the parallel visit is not successful, then the two complexes are not isomorphic. □

4.4 Reduction semantics

The dynamics of a system is formally specified by the reduction semantics reported in Tab.4.13. It makes use of the structural congruence $\equiv$ over BlenX systems defined in Tab.4.6. Moreover, we assume a total function $\delta : \mathcal{N} \to \mathbb{N}$ that associates priorities to names.

Rule (r1) in Tab.4.13 says that two conditions have to be met for the firing of a $\langle C \rangle \mathsf{ch}(x, \Gamma, p)$ prefix within a box with interface I: the guarding expression C has to evaluate to *true* in I, and I must have an interface with subject x. Under the above hypotheses, and if Γ does not clash with the sorts of the other interfaces in I, then

$$\text{(r1)}\quad \frac{\wr C \int_I = true}{(I[\,\langle C\rangle \mathsf{ch}(x,\Gamma,p).P + G \,|\, P_1\,]_n, E, \xi) \to_p (I_1[\,P \,|\, P_1\,]_n, E, \xi\{\Gamma n/\Delta n\})}$$

$$(c1)\quad \text{where } I = K(x,\,\Delta)^{p'}\, I^* \text{ and } I_1 = K(x,\,\Gamma)^{p'}\, I^* \text{ and } \Gamma \notin \mathsf{sorts}(I^*)$$

$$\text{(r2)}\quad \frac{\wr C_1 \int_I = true \quad \wr\, C_2 \int_I = true}{(I[\,\langle C_1\rangle x!z.P_1 + G_1 \,|\, \langle C_2\rangle x?y.P_2 + G_2 \,|\, P\,]_n, E, \xi) \to_p (I[\,P_1 \,|\, P_2\{z/y\} \,|\, P\,]_n, E, \xi)}$$

$$(c2)\quad \text{provided } K(x,\,\Delta)^p \in I \text{ or } (x \notin \mathsf{sub}(I) \text{ and } \delta(x) = p)$$

$$\text{(r3)}\quad (I_1[\,P_1\,]_n \parallel I_2[\,P_2\,]_m, E, \xi) \to_p (I_1'[\,P_1\,]_n \parallel I_2'[\,P_2\,]_m, E, \xi \cup \{\{\Delta_1 n, \Delta_2 m\}\})$$

$$(c3)\quad \text{where } I_i = \oplus(x_i,\,\Delta_i)^{p_i}\, I_i^* \text{ and } I_i' = \otimes(x_i,\,\Delta_i)^{p_i}\, I_i^* \text{ for } i = 1,2$$
$$\text{and provided } \alpha_b(\Delta_1, \Delta_2) = p$$

$$\text{(r4)}\quad (I_1[\,P_1\,]_n \parallel I_2[\,P_2\,]_m, E, \xi \cup \{\{\Delta_1 n, \Delta_2 m\}\}) \to_p (I_1'[\,P_1\,]_n \parallel I_2'[\,P_2\,]_m, E, \xi)$$

$$(c4)\quad \text{where } I_i = \otimes(x_i,\,\Delta_i)^{p_i}\, I_i^* \text{ and } I_i' = \oplus(x_i,\,\Delta_i)^{p_i}\, I_i^* \text{ for } i = 1,2$$
$$\text{and provided } \alpha_u(\Delta_1, \Delta_2) = p$$

$$\text{(r5)}\quad \frac{\wr C_1 \int_{I_1} = true \wedge \wr C_2 \int_{I_2} = true}{(I_1[\,M_1 \,|\, Q_1\,]_n \parallel I_2[\,M_2 \,|\, Q_2\,]_m, E, \xi) \to_p (I_1[\,R_1 \,|\, Q_1\,]_n \parallel I_2[\,R_2\{z/w\} \,|\, Q_2\,]_m, E, \xi)}$$

$$(c5)\quad \text{where } M_1 = \langle C_1\rangle x_1!z.\, R_1 + M_1' \text{ and } M_2 = \langle C_2\rangle x_2?w.\, R_2 + M_2' \text{ and}$$
$$\text{where } I_i = K(x_i,\,\Delta_i)^{p_i}\, I_i^* \text{ and } z \notin \mathsf{sub}(I_i) \text{ for } i = 1,2 \text{ and provided}$$
$$(K = \otimes \text{ and } \alpha_c(\Delta_1, \Delta_2) = p \text{ and } \{\Delta_1 n, \Delta_2 m\} \in \xi) \text{ or}$$
$$(K = \oplus \text{ and } \alpha_c(\Delta_1, \Delta_2) = p \text{ and } \alpha_b(\Delta_1, \Delta_2) = \alpha_u(\Delta_1, \Delta_2) = 0)$$

$$\text{(r6)}\quad \frac{\mathsf{Map}(\mathsf{CB}(B_1, \xi_1), \mathsf{CB}(B_2, \xi_2)) = \sigma \quad \mathsf{id}(\xi) \cap \mathsf{id}(\xi_2) = \emptyset}{(B_1, (B_1, \xi_1) \rhd_p (B_2, \xi_2) \;\parallel\; E, \xi_1 \cup \xi) \to_p (B_2, (B_1, \xi_1) \rhd_p (B_2, \xi_2) \;\parallel\; E, \xi_2 \cup \xi\sigma)}$$

$$\text{(r7)}\quad \frac{(B, E, \xi) \to_p (B', E, \xi') \wedge (\mathsf{id}(B) \cup \mathsf{id}(B')) \cap \mathsf{id}(B_1) = \emptyset}{(B \parallel B_1, E, \xi) \to_p (B' \parallel B_1, E, \xi')}$$

$$\text{(r8)}\quad \frac{S_1 \equiv S_1' \quad S_1' \to_p S_2' \quad S_2' \equiv S_2}{S_1 \to_p S_2}$$

Table 4.13: Reduction semantics of BlenX.

the sort of x is turned to Γ. Since the interface x could be involved in a binding, the environment is consistently updated by possibly refreshing the previous sort of x with Γ. Here notice that the requirement about the freshness of Γ guarantees the freshness of Γn in the updated environment.

Parallel processes that perform complementary actions on the same channel inside the same box (a process perform and input $x?y$ and the other one an output $x!z$) can synchronize and exchange a message. This communication is also called *intra-communication* and is ruled by (r2). The value z flows from the process performing the output to the one performing the input. The flow of information affects the future behaviour of the system because all the free occurrences of y bound by the input place-holder are replaced in the receiving process by the actual value z.

Change actions and intra-communications are referred as *monomolecular* actions.

Boxes can interact, through *bimolecular* reactions, in various ways: they can bind together, unbind, or just communicate. These interactions are based on the existence of a compatibility function, previously described and here formalized as a symmetric function $\alpha : \mathcal{T}^2 \to \mathbb{N}^3$ which returns a triple of values representing priorities associated with the *binding*, *unbinding* and *communication* compatibilities of the two argument sorts. Values in $\mathbb{N}^+$ represent priorities and we assume that higher is the value higher is the priority. Moreover, with 0 we indicate that no priority is associated with the sort pair, meaning that they are not compatible. We use $\alpha_b(\Delta, \Gamma)$, $\alpha_u(\Delta, \Gamma)$, and $\alpha_c(\Delta, \Gamma)$ to mean, respectively, the first, the second, and the third projection of $\alpha(\Delta, \Gamma)$.

Rules (r3) and (r4) describe the dynamics of binding and unbinding, respectively, which can only take place if the binding/unbinding compatibilities of the values of the involved interfaces is greater than zero. In both cases the modification of the binding state of the relevant interfaces is reflected in the interface markers, which are changed either from $\oplus$ to $\otimes$ or the other way round. Also, the link $\{\Delta_1 n_1, \Delta_2 n_2\}$ recording the actual binding is either added to the environment or removed from it. The third kind of interaction between boxes is ruled by (r5). This involves an input and an output action that can fire in two distinct boxes over interfaces with associated sorts Δ_1 and Δ_2. Information flows from the box containing the sending action to the box enclosing the receiving process. Note that the communication depends on the compatibility of Δ_1 and Δ_2 rather than on the fact that input and output actions occur over exactly the same name. Indeed inter-communication is enabled only if $\alpha_c(\Delta_1, \Delta_2) > 0$, and only under the proviso that either the two interfaces are already bound together or they are free and both $\alpha_b(\Delta_1, \Delta_2)$ and $\alpha_u(\Delta_1, \Delta_2)$ are 0. If the communication happens between interfaces that are not bound, we call it *inter-communication*, while if it happens between bound interfaces, we call it *complex-communication*.

Given the system $(B_1, E, \xi_1 \cup \xi)$, rule (r6) defines reductions corresponding to the occurrence of *events* of the shape $(B_1, \xi_1) \rhd_p (B_2, \xi_2)$ when there is no clash between the names of ξ and those of ξ_2. The application of the rule (r6) involves checking the possibility of substituting B_1 and the portion of environment ξ_1 with the bio-process B_2

$$\mathsf{Map}(a,b) = \begin{cases} \emptyset & \text{if } a = \emptyset \text{ and } b = \emptyset \\ \langle {}^{m}\!/\!_{n} \rangle \uplus \mathsf{Map}(S, S') & \text{if } \exists \Delta \in \mathcal{T} \text{ s.t. } a = \{\Delta n\} \cup S \\ & \text{and } b = \{\Delta m\} \cup S' \\ \bot & \text{otherwise} \end{cases}$$

Table 4.14: Function Map.

and the subenvironment ξ_2. The function Map, defined in Tab. 4.14, serves this goal. Recall from Sec. 4.1 that the function $\mathsf{CB}(B_j, \xi_j)$ allows the collection of data about interfaces of boxes in B_j which are possibly involved in bindings outside ξ_j. Function Map is applied to $\mathsf{CB}(B_1, \xi_1)$ and $\mathsf{CB}(B_2, \xi_2)$ to return a consistent mapping from the hanging bindings of B_1 over ξ_1 to the hanging bindings of B_2 over ξ_2. When such a mapping exists, Map returns a substitution that is applied to ξ to get a fully updated environment after the substitution of B_1 with B_2.

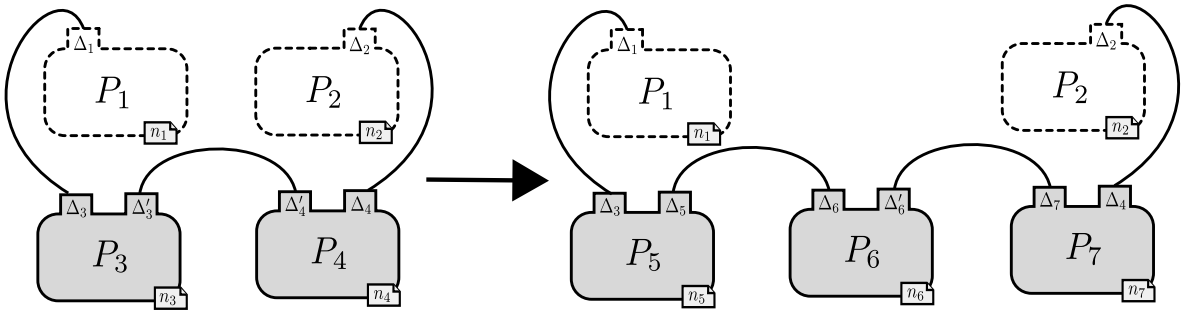


Figure 4.3: Example of an event substituting a subpart of a complex.

Fig. 4.3 actually describes a reduction step of a BlenX system which can be inferred by using (r6). In fact, naming B_j the box containing P_j ($j = 1, \ldots, 7$) and setting:

$$\begin{aligned} \xi_1 &= \{\{\Delta'_3 n_3, \Delta'_4 n_4\}\} \\ \xi_2 &= \{\{\Delta_5 n_5, \Delta_6 n_6\}, \{\Delta'_6 n_6, \Delta_7 n_7\}\} \\ E &= \{(B_3 \parallel B_4, \xi_1) \rhd_p (B_5 \parallel B_6 \parallel B_7, \xi_2)\} \\ \xi &= \{\{\Delta_1 n_1, \Delta_3 n_3\}, \{\Delta_4 n_4, \Delta_2 n_2\}\} \end{aligned}$$

by (r6) we get:

$$(B_3 \parallel B_4, E, \xi_1 \cup \xi) \to (B_5 \parallel B_6 \parallel B_7, E, \xi_2 \cup \xi\sigma)$$

where $\sigma = \{{}^{\Delta_3 n_5}\!/\!_{\Delta_3 n_3}\}, \{{}^{\Delta_4 n_7}\!/\!_{\Delta_4 n_4}\}$.

We conclude the description of the rules in Tab. 4.13 by just observing that, as usual in reduction semantics, the rules (r7) and (r8) are meant to extend reductions over parallel compositions and over structural re-shufflings.

Given the reduction semantics of BlenX we want now to show that it preserves well-formedness of systems. We start by showing that for each reduction $S \rightarrow_p S'$, there exists a *normalized derivation*, where structural congruence is applied only in the last derivation step. We will use then this result to show that if S is well-formed, then also S' is well-formed.

Definition 4.4.1. *A normalized derivation of the reduction $S \rightarrow_p S'$ is of the following form. The first rule applied is an instance of one of the axioms (r1), (r2), (r3), (r4), (r5), (r6), where from a system (B, E, ξ) we obtain a system (B', E, ξ'). The derivation continues (in all the cases is similar) with the application of (r7), yielding $(B' \parallel B_1, E, \xi')$. The last rule applied in the derivation is (r8), so that $S \equiv (B \parallel B_1, E, \xi)$ and $S' \equiv (B' \parallel B_1, E, \xi')$.*

Proposition 4.4.2. *If $S \in \overline{\mathcal{S}}$ and $S \rightarrow_p S'$, then there exist B, I, I_1, I_1', I_2, I_2', P, P_1, P_2, M_1, M_2, R_1, R_2, E and ξ, ξ_1, ξ_2 such that one of the following holds:*

1) $S \equiv S_1 = (K(x, \Delta)^p\, I^*[\,\langle C\rangle \mathsf{ch}(x, \Gamma, p).P + M_1 \mid P_1\,]_n \parallel B, E, \xi)$
   $S' \equiv S_1' = (K(x, \Gamma)^p\, I^*[\,P \mid P_1\,]_n \parallel B, E, \xi\{{}^{\Gamma n}/_{\Delta n}\})$
   where $\wr\, C \int_I = true$ and Tab. 4.13(c1)
2) $S \equiv S_1 = (I[\,\langle C_1\rangle x!z.P_1 + G_1 \mid \langle C_2\rangle x?y.P_2 + G_2 \mid P\,]_n \parallel B, E, \xi)$
   $S' \equiv S_1' = (I[\,P_1 \mid P_2\{{}^{z}/_{y}\} \mid P\,]_n \parallel B, E, \xi)$
   where $\wr\, C_1 \int_I = \wr C_2 \int_I = true$ and Tab. 4.13(c2)
3) $S \equiv S_1 = (I_1[\,P_1\,]_n \parallel I_2[\,P_2\,]_m \parallel B, E, \xi)$
   $S' \equiv S_1' = (I_1'[\,P_1\,]_n \parallel I_2'[\,P_2\,]_m \parallel B, E, \xi \cup \{\{\Delta_1 n, \Delta_2 m\}\})$
   where Tab. 4.13(c3)
4) $S \equiv S_1 = (I_1[\,P_1\,]_n \parallel I_2[\,P_2\,]_m \parallel B, E, \xi \cup \{\{\Delta_1 n, \Delta_2 m\}\})$
   $S' \equiv S_1' = (I_1[\,P_1\,]_n \parallel I_2[\,P_2\,]_m \parallel B, E, \xi)$
   where Tab. 4.13(c4)
5) $S \equiv S_1 = (I_1[\,M_1 \mid Q_1\,]_n \parallel I_2[\,M_2 \mid Q_2\,]_m \parallel B, E, \xi)$
   $S' \equiv S_1' = (I_1[\,R_1 \mid Q_1\,]_n \parallel I_2[\,R_2\{{}^{z}/_{w}\} \mid Q_2\,]_m \parallel B, E, \xi)$
   where $\wr\, C_1 \int_I = \wr C_2 \int_I = true$ and Tab. 4.13(c5)
6) $S \equiv S_1 = (B_1 \parallel B, (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E, \xi_1 \cup \xi)$
   $S' \equiv S_1' = (B_2 \parallel B, (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E, \xi_2 \cup \xi\sigma)$
   where $\mathsf{Map}(\mathsf{CB}(B_1, \xi_1), \mathsf{CB}(B_2, \xi_2)) = \sigma$ and $\mathsf{id}(\xi) \cap \mathsf{id}(\xi_2) = \emptyset$

Proof. We first show that if $S \rightarrow_p S'$ then there always exists a normalized derivation

of $S \rightarrow_p S'$. For the derivations generated by the axioms (r1-5), the statement can be proven by that the normalized derivation can be obtained by repeatedly apply one of the derivation tree rearrangements presented below. In particular, starting from the top of the derivation, we search a pattern where the rearrangements can be applied, we apply the modification and then start again from the top of the derivation. This recursive procedure is continued till the derivation reaches the normalized form. Note that the procedure never increases the depth of derivation tree (which is finite), decreases the number of rules (r7), and pushes down the application of the structural congruence; this facts, combined with the kind of rearrangements we apply, make the normalization procedure always terminating.

The tree rearrangements are:

(a) Two consecutive occurrences of the (r7) rule can be condensed into one. This is achieved by exploiting the transitivity of $\equiv$ and re-arranging derivations as it is done below:

$$\cfrac{\cfrac{\cfrac{\dots}{(B_1,E_1,\xi_1) \rightarrow_p (B_2,E_2,\xi_2)} \quad \wedge\, (\mathsf{id}(B_1)\cup \mathsf{id}(B_2)) \cap \mathsf{id}(B') = \emptyset}{(B_3 = B_1 \parallel B', E, \xi) \rightarrow_p (B_4 = B_2 \parallel B', E', \xi')} \quad \wedge\, (\mathsf{id}(B_3)\cup \mathsf{id}(B_4)) \cap \mathsf{id}(B'') = \emptyset}{S = (B_3 \parallel B'', E, \xi) \rightarrow_p (B_4 \parallel B'', E', \xi') = S'}$$

$$\Downarrow$$

$$\cfrac{\cfrac{\cfrac{\dots}{(B_1,E_1,\xi_1) \rightarrow_p (B_2,E_2,\xi_2)} \quad \wedge\, (\mathsf{id}(B_1)\cup \mathsf{id}(B_2)) \cap \mathsf{id}(B' \parallel B'') = \emptyset}{S \equiv (B_1 \parallel (B' \parallel B''), E_1, \xi_1) \rightarrow_p (B_2 \parallel (B' \parallel B''), E_2, \xi_2) \equiv S'}}{S \rightarrow_p S'}$$

(b) An occurrence of the (r8) rule followed by an occurrence of the (r7) rule can be converted into an instance of (r7) followed by an instance of (r8):

$$\cfrac{\cfrac{(B,E,\xi) \equiv \quad \cfrac{\dots}{(B_1,E_1,\xi_1) \rightarrow_p (B_2,E_2,\xi_2)} \quad \equiv (B',E',\xi')}{(B,E,\xi) \rightarrow_p (B',E',\xi')} \quad \wedge\, (\mathsf{id}(B)\cup \mathsf{id}(B')) \cap \mathsf{id}(B'') = \emptyset}{S = (B \parallel B'', E, \xi) \rightarrow_p (B' \parallel B'', E', \xi') = S'}$$

$$\Downarrow$$

$$\cfrac{S \equiv S'_1 \quad \cfrac{\cfrac{\dots}{(B_1,E_1,\xi_1) \rightarrow_p (B_2,E_2,\xi_2)} \quad \wedge\, (\mathsf{id}(B_1)\cup \mathsf{id}(B_2)) \cap \mathsf{id}(B'') = \emptyset}{S'_1 = (B_1 \parallel B'', E_1, \xi_1) \rightarrow_p (B_2 \parallel B'', E_2, \xi_2) = S'_2} \quad S'_2 \equiv S'}{S \rightarrow_p S'}$$

where we are sure that $(\mathsf{id}(B_1) \cup \mathsf{id}(B_2)) \cap \mathsf{id}(B'') = \emptyset$ holds, because if the rule is

applied starting from the top of the derivation. Indeed, since considered axioms does not change boxes identifiers, by applying the rule starting from the top, we are sure that no clashes are introduced by intermediate identifiers renaming;

(c) Two consecutive occurrences of the (r8) rule can be condensed into one. This is achieved by exploiting the transitivity of $\equiv$ and re-arranging derivations as it is done below:

$$\dfrac{S \equiv S_1 \qquad \dfrac{S_1 \equiv \quad \dfrac{\ldots}{S_1' \to_p S_2'} \quad \equiv S_2}{S_1 \to_p S_2} \qquad S_2 \equiv S'}{S \to_p S'}$$

$$\Downarrow$$

$$\dfrac{S \equiv S_1 \equiv S_1' \quad \dfrac{\ldots}{S_1' \to_p S_2'} \quad S_2' \equiv S_2 \equiv S'}{S \to_p S'}$$

Given this, cases 1, 2, 3, 4 and 5 corresponds to one of the possible distinct axioms driving the normalized derivation of $S \to_p S'$.

Now consider a derivation generated by axiom (r6). We have that rearrangements (a) and (c) are still valid, while rearrangement (b) is not valid in general. Indeed, note how boxes in B_2 can have names that can clash with the context surrounding the bio-process B_1; hence an application of rule (r8) in the derivation cannot be always moved down, because there are cases in which we can generate clashes, making the condition of rule (r7) no more valid. However, considering a derivation:

$$\dfrac{\dfrac{\dfrac{\mathsf{Map}(\mathsf{CB}(B_1,\xi_1),\mathsf{CB}(B_2,\xi_2)) = \sigma \wedge \mathsf{id}(\xi) \cap \mathsf{id}(\xi_2) = \emptyset}{(B_1, E = (B_1,\xi_1) \rhd_p (B_2,\xi_2) \parallel E', \xi \cup \xi_1) \to (B_2, E = (B_1,\xi_1) \rhd_p (B_2,\xi_2) \parallel E', \xi' = \xi_2 \cup \xi\sigma)}}{\ldots}}{S \to_p S'}$$

we can always consider a complex $(B_2', \xi_2') \equiv_c (B_2, \xi_2)$, such that all its boxes identifiers are different to the one used in S, and transform the previous derivation by adding at the beginning a step of rule (r8) in the following way:

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{Map}(\mathsf{CB}(B_1,\xi_1),\mathsf{CB}(B_2',\xi_2')) = \sigma' \wedge \mathsf{id}(\xi) \cap \mathsf{id}(\xi_2') = \emptyset}{(B_1,E,\xi) \equiv (B_1,(B_1,\xi_1) \rhd_p (B_2',\xi_2')||E', \xi \cup \xi_1) \to (B_2',(B_1,\xi_1) \rhd_p (B_2',\xi_2')||E', \xi_2' \cup \xi\sigma') \equiv (B_2,E,\xi')}}{(B_1,E,\xi) \to_p (B_2,E',\xi')}}{\ldots}}{S \to_p S'}$$

and then use the procedure explained before to obtain a normalized derivation of $S \rightarrow_p S'$. Note that we are sure that $(B_2', (B_1, \xi_1) \rhd_p (B_2', \xi_2') \parallel E', \xi_2' \cup \xi\sigma') \equiv (B_2, E, \xi_2 \cup \xi\sigma)$, because the substitutions taken by σ and σ' on ξ regards identifiers of boxes present in B_2 and B_2'. Note, moreover, that by adding this initial derivation, we guarantee that the axiom application does not produces a bio-process that can generate clashes in the top-down rearrangement procedure. □

Theorem 4.4.3. *Let $S \in \overline{\mathcal{S}}$. Then $S \rightarrow_p S'$ implies $S' \in \overline{\mathcal{S}}$.*

Proof. By Proposition 4.4.2 we can distinguish six cases, depending on the normalized derivation corresponding to $S \rightarrow_p S'$.

(case 1)

By hypothesis we have $S \equiv S_1$ and, by Theorem 4.2.7, we know $S_1 \in \overline{\mathcal{S}}$. Thus, combining it with the hypothesis, we have $\vdash_b K(x,\, \Delta)^p\, I^*[\, \langle C \rangle \mathsf{ch}(x, \Gamma, p).P + M_1 \mid P_1 \,]_n \parallel B : (N, L)$, $\vdash_{ev} E : ok$ and $\vdash_{en} \xi : L$. Since the bio-process is well-formed, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\vdash_i K(x,\, \Delta)^p : T} \;\wedge\; \dfrac{\dots}{\vdash_i I^* : T'}}{\vdash_i K(x,\, \Delta)^p\, I^* : T \cup T'}}{\vdash_b K(x,\, \Delta)^p\, I^*[\, \langle C \rangle \mathsf{ch}(x, \Gamma, p).P + M_1 \mid P_1 \,]_n : (\{n\}, \underbrace{\{(\Delta', n) \mid \Delta' \in T \cup T'\}}_{L'})} \;\wedge\; \dfrac{\dots}{\vdash_b B : (N', L'')}}{\vdash_b K(x,\, \Delta)^p\, I^*[\, \langle C \rangle \mathsf{ch}(x, \Gamma, p).P + M_1 \mid P_1 \,]_n \parallel B : (\{n\} \cup N', \underbrace{L' \cup L''}_{L})}$$

where T' is equal $\emptyset$ if I^* coincides with ϵ and where $T = \{\Delta\}$ if $S = \otimes$ and $T = \emptyset$ otherwise. Considering that by hypothesis $\Gamma \notin \mathsf{sorts}(I^*)$ we can derive:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\vdash_i K(x,\, \Gamma)^p : T''} \;\wedge\; \dfrac{\dots}{\vdash_i I^* : T'}}{\vdash_i K(x,\, \Gamma)^p\, I^* : T'' \cup T'}}{\vdash_b K(x,\, \Gamma)^p\, I^*[\, P \mid P_1 \,]_n : (\{n\}, \underbrace{\{(\Delta', n) \mid \Delta' \in T'' \cup T'\}}_{L'''})} \;\wedge\; \dfrac{\dots}{\vdash_b B : (N', L'')}}{\vdash_b K(x,\, \Gamma)^p\, I^*[\, P \mid P_1 \,]_n \parallel B : (\{n\} \cup N', \underbrace{L''' \cup L''}_{L_1})}$$

where all the side conditions of the applied rules are respected. At this point we can distinguish between two cases:

(1) if $S = \otimes$ then the only difference between the signatures $(\{n\} \cup N', L)$ and $(\{n\} \cup N', L_1)$ is that $L' = \{(\Delta, n)\} \cup L''''$, while $L''' = \{(\Gamma, n)\} \cup L''''$ (remember that by well-formedness each pair in L' and L''' compares only once); hence $L = \{(\Delta, n)\} \cup L_2$ while $L_1 = \{(\Gamma, n)\} \cup L_2$, where $L_2 = L'''' \cup L''$. Moreover, since ξ is well-formed,

we know that each occurrence $\Delta' n'$ in ξ is unique, meaning that there exists a link such that $\xi = \{\Delta n, \Delta' n'\} \cup \xi'$ where the occurrence Δn is unique in all the links of ξ. But this means that $\xi\{^{\Gamma n}/_{\Delta n}\} = \{\Gamma n, \Delta' n'\} \cup \xi'$ and by considering that $\vdash_{en} \xi : L$ we immediately have that $\vdash_{en} \xi\{^{\Gamma n}/_{\Delta n}\} : L_1$. Hence $(K(x,\, \Gamma)^p\, I^*[\, P \,|\, P_1 \,]_n \parallel B, E, \xi\{^{\Gamma n}/_{\Delta n}\}) = S_1'$ is well-formed and by Lemma 4.2.7 we finally obtain $S' \in \overline{\mathcal{S}}$;

(2) if $S = \oplus$ then both T and T'' are equal to $\emptyset$. Moreover, there is not a link component Δn in ξ and hence $\xi\{^{\Gamma n}/_{\Delta n}\} = \xi$. Therefore, it results that $(K(x,\, \Gamma)^p\, I^*[\, P \,|\, P_1 \,]_n \parallel B, E, \xi\{^{\Gamma n}/_{\Delta n}\}) = S_1'$ is well-formed and by Lemma 4.2.7 we have $S' \in \overline{\mathcal{S}}$.

(case 3)
By hypothesis we have $S \equiv S_1$ and, by Theorem 4.2.7, we know that $S_1 \in \overline{\mathcal{S}}$. Thus, $\vdash_b I_1[\, P_1 \,]_n \parallel I_2[\, P_2 \,]_m \parallel B : (N, L)$, $\vdash_{ev} E : ok$ and $\vdash_{en} \xi : L$. Since the bio-process is well-formed, we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\vdash_i \oplus(x_1,\, \Delta_1)^p : \emptyset} \quad \dfrac{\dots}{\vdash_i I_1^* : T_1'}}{\vdash_i \oplus(x_1,\, \Delta_1)^p\, I_1^* : T_1'}}{\vdash_b I_1[\, P_1 \,]_{n_1} : (\{n_1\}, L_1)} \quad \dfrac{\dfrac{\dfrac{\dots}{\vdash_i \oplus(x_2,\, \Delta_2)^p : \emptyset} \quad \dfrac{\dots}{\vdash_i I_2^* : T_2'}}{\vdash_i \oplus(x_2,\, \Delta_2)^p\, I_2^* : T_2'}}{\vdash_b I_1[\, P_1 \,]_{n_2} : (\{n_2\}, L_2)}}{\vdash_b I_1[\, P_1 \,]_{n_1} \parallel I_2[\, P_2 \,]_{n_2} : (\{n_1, n_2\} : L_1 \cup L_2)} \quad \dfrac{\dots}{\vdash_b B : (N', L')}}{\vdash_b I_1[\, P_1 \,]_{n_1} \parallel I_2[\, P_2 \,]_{n_2} \parallel B : (\{n_1, n_2\} \cup N', \underbrace{L_1 \cup L_2 \cup L'}_{L})}$$

where T_1 and T_2 are equal to $\emptyset$ if I_1^* and I_2^* coincide to ϵ. We can now prove that $I_1'[\, P_1 \,]_n \parallel I_2'[\, P_2 \,]_m \parallel B'$ is well-formed by constructing the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\vdash_i \otimes(x_1,\, \Delta_1)^p : \{\Delta_1\}} \quad \dfrac{\dots}{\vdash_i I_1^* : T_1'}}{\vdash_i \otimes(x_1,\, \Delta_1)^p\, I_1^* : T_1}}{\vdash_b I_1'[\, P_1 \,]_{n_1} : (\{n_1\}, \{(\Delta_1, n_1)\} \cup L_1)} \quad \dfrac{\dfrac{\dfrac{\dots}{\vdash_i \otimes(x_2,\, \Delta_2)^p : \{\Delta_2\}} \quad \dfrac{\dots}{\vdash_i I_2^* : T_2}}{\vdash_i \otimes(x_2,\, \Delta_2)^p\, I_2^* : T_2}}{\vdash_b I_2'[\, P_2 \,]_{n_2} : (\{n_2\}, \{(\Delta_2, n_2)\} \cup L_2)}}{\vdash_b I_1'[\, P_1 \,]_{n_1} \parallel I_2'[\, P_2 \,]_{n_2} : (\{n_1, n_2\} : \{(\Delta_1, n_1), (\Delta_2, n_2)\} \cup L_1 \cup L_2)} \quad \dfrac{\dots}{\vdash_b B : (N', L')}}{\vdash_b I_1'[\, P_1 \,]_{n_1} \parallel I_2'[\, P_2 \,]_{n_2} \parallel B : (\{n_1, n_2\} \cup N', \{(\Delta_1, n_1), (\Delta_2, n_2)\} \cup L_1 \cup L_2 \cup L')}$$

where all the rule side conditions are respected and where, by reasoning on the well-formedness of interfaces I_1 and I_2 and their differences with respect to I_1' and I_2', we obtain $\{(\Delta_1, n_1), (\Delta_2, n_2)\} \cap (L_1 \cup L_2 \cup L') = \emptyset$. The latter observation can be combined with the fact that $\vdash_{en} \xi : L$ and that $n_1 \neq n_2$ for constructing the following derivation:

$$\frac{\vdash_{en} \xi : L \ \wedge \ \vdash_{en} \{\Delta_1 n_1, \Delta_2 n_2\} : \{(\Delta_1, n_1), (\Delta_2, n_2)\}}{\vdash_{en} \xi \cup \{\{\Delta_1 n_1, \Delta_2 n_2\}\} : L \cup \{(\Delta_1, n_1), (\Delta_2, n_2)\}}$$

Hence $(I_1'[\,P_1\,]_n \parallel I_2'[\,P_2\,]_m \parallel B, E, \xi \cup \{\{\Delta_1 n, \Delta_2 m\}\}) = S_1'$ is well-formed and, since $S_1 \equiv S'$, by using the Theorem 4.2.7 it results $S' \in \overline{\mathcal{S}}$.

(case 6)

By hypothesis and Lemma 4.2.7 we know that $(B_1 \parallel B, (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E, \xi_1 \cup \xi)$ is in $\overline{\mathcal{S}}$ and hence $\vdash_b (B_1 \parallel B, (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E, \xi_1 \cup \xi) : (N, L)$, $\vdash_{ev} (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E : ok$ and $\vdash_{en} \xi : L$. From this we know that the well-formedness derivations of the three elements have the following forms:

$$\frac{\dfrac{\dots}{\vdash_b B_1 : (N_1, L_1)} \ \wedge \ \dfrac{\dots}{\vdash_b B : (N', L')}}{\vdash_b B_1 \parallel B : (N_1 \cup N', L_1 \cup L')}$$

and

$$\frac{\dfrac{\dfrac{\dfrac{\dots}{\vdash_b B_1 : (N_1, L_1)} \ \wedge \ \dfrac{\dots}{\vdash_b B_2 : (N_2, L_2)} \ \wedge \ \dfrac{\dots}{\vdash_{en} \xi_1 : L_1'} \ \wedge \ \dfrac{\dots}{\vdash_{en} \xi_2 : L_2'}}{\vdash_{ev} (B_1, \xi_1) \rhd_p (B_2, \xi_2) : ok} \ \wedge \ \dfrac{\dots}{\vdash_{ev} E : ok}}{\vdash_{ev} (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E : ok}$$

and

$$\frac{\dfrac{\dots}{\vdash_{en} \xi_1 : L_1'} \ \wedge \ \dfrac{\dots}{\vdash_{en} \xi : L''}}{\vdash_{en} \xi_1 \cup \xi : L_1' \cup L''}$$

Note that $L_1' \subseteq L_1$ and $L_2' \subseteq L_2$ and that for each $\Delta \in \mathcal{T}$ it is:

$$|\{(\Delta, n) \in L_1 \setminus L_1' \mid n \in \mathcal{N}\}| = |\{(\Delta, n) \in L_2 \setminus L_2' \mid n \in \mathcal{N}\}|$$

This means firstly that $\mathsf{Map}(\mathsf{CB}(B_1, \xi_1), \mathsf{CB}(B_2, \xi_2)) \neq \bot$, and hence that the function returns a valid sequence σ (eventually empty) of identifier substitutions. Moreover, by well-formedness of B_1 and B_2 we are sure that the single substitutions composing σ (if any) are all different, i.e., $\sigma = \langle {}^{x_1, \cdots, x_n}/_{y_1, \cdots, y_n} \rangle$ is such that for each pairs (x_i, x_j) and (y_i, y_j) (with $i \neq j$ and $i, j \in \{1, \cdots, n\}$) we have $x_i \neq x_j$ and $y_i \neq y_j$. Hence, the substitution σ is a bijection between the sets $L_1 \setminus L_1'$ and $L_2 \setminus L_2'$, meaning that $L_2 \setminus L_2' = \{(\Delta, \sigma(n)) \mid (\Delta, n) \in L_1 \setminus L_1'\}$, and vice-versa (considering $\sigma^{-1} = \langle {}^{y_1, \cdots, y_n}/_{x_1, \cdots, x_n} \rangle$ the inverse of the substitution).

From the fact that $L_1 \uplus L' = L_1' \uplus L''$ and $L_1' \subseteq L_1$ we have $L'' = (L_1 \setminus L_1') \cup L'$. This indirectly means that each link $\{\Delta n, \Delta' n'\}$ with $n \in \mathsf{id}(B_1)$ and $n' \notin \mathsf{id}(B_1)$ is in ξ with

$n \in dom(\sigma)$.

Now we want to use all this facts to prove that S' is well-formed. By hypothesis we know that the derivation $(B_1 \parallel B, (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E, \xi_1 \cup \xi) \rightarrow_p (B_2 \parallel B, (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E, \xi_2 \cup \xi\sigma)$ is normalized and hence surely $(\mathsf{id}(B_1) \cup \mathsf{id}(B_2)) \cap \mathsf{id}(B) = \emptyset$, guaranteeing $\vdash_b B_2 \parallel B : (N_2 \cup N', L_2 \cup L')$.

Now consider $\xi\sigma$. By previous facts we have $\xi = \xi_a \cup \xi_b$ where $\xi_a = \{\{\Delta n, \Delta' n'\} \mid n \in \mathsf{id}(B_1)\}$, $\vdash_{en} \xi_a : L_a$ and $\vdash_{en} \xi_b : L_b$. This itself allows to observe that $L_a = L_{a'} \cup L_{a''}$ where $L_{a'} = \{(\Delta, n) \mid n \in \mathsf{id}(B_1)\}$. Hence $L_{a'}$ corresponds to $L_1 \setminus L_1'$ and $L' = L_{a''} \cup L_b$. Now, by previous facts and by definition of substitution we can say that $\xi\sigma = \xi_a\sigma \cup \xi_b = \{\{\Delta\sigma(n), \Delta' n'\} \mid n \in \mathsf{id}(B_1)\} \cup \xi_b$. Hence $\vdash_{en} \xi_a\sigma : L_{a'''} \cup L_{a''}$ where $L_{a'''} = \{(\Delta, \sigma(n)) \mid (\Delta, n) \in L_{a'}\}$. Since $L_{a'}$ corresponds to $L_1 \setminus L_1'$ then $L_{a'''}$ corresponds to $L_2 \setminus L_2'$ and hence $\vdash_{en} \xi\sigma : (L_2 \setminus L_2') \cup (L_{a''} \cup L_b) = (L_2 \setminus L_2') \cup L'$ from which follows $\vdash_{en} \xi_2 \cup \xi\sigma : L_2' \cup (L_2 \setminus L_2') \cup L' = L_2 \cup L'$.

This means that $(B_2 \parallel B', (B_1, \xi_1) \rhd_p (B_2, \xi_2) \parallel E, \xi_2 \cup \xi\sigma) = S_1'$ is well-formed and hence, since $S_1 \equiv S'$, by Lemma 4.2.7 we can conclude $S' \in \overline{\mathcal{S}}$.

(case 2)

This case is immediate by observing that an intra-communication does not change any structural element involved in the well-formedness derivation.

(case 4)

This case is close to the (case 3) and can be proven similarly.

(case 5)

This case is immediate by observing that an inter-communication does not change any structural element involved in the well-formedness derivation. □

Given the reduction relation $\rightarrow_p$ over BlenX systems, we now give a constructive method to generate a relation $\rightarrow$, that is used to define the transition system $(\overline{\mathcal{S}}, \rightarrow)$. The relation $\rightarrow$ considers priorities associated with reductions. In particular, since we use a model of priorities which is *global* we have to ensure that given a system S, the reductions of S with higher priority have the precedence with respect to the ones with lower priority. Moreover, we do not consider reductions with priority zero.

Definition 4.4.4. *Relation $\rightarrow \subset \overline{\mathcal{S}} \times \overline{\mathcal{S}}$ is obtained with the following procedure:*

$$\rightarrow = \{(S, S') \mid S \rightarrow_p S' \wedge p > 0 \wedge \nexists p' > p \text{ s.t. } S \rightarrow_{p'} S''\}$$

As usual, $\rightarrow^+$ indicates the transitive closure and $\rightarrow^*$ the reflexive and transitive closure of $\rightarrow$. Moreover, $S \rightarrow^k S'$ with $k \geq 1$ indicates that S' can be reached from S with k transitions $S \rightarrow^1 S_1 \rightarrow^2 \cdots \rightarrow^{k-1} S_{k-1} \rightarrow^k S'$. Moreover, we write $S \not\rightarrow$ if there does not exists S' such that $S \rightarrow S'$.

Corollary 4.4.5. *Let $S \in \overline{\mathcal{S}}$. Then $S \to S'$ implies $S' \in \overline{\mathcal{S}}$.*

Proof. Immediate from Theorem 4.4.3 and Definition 4.4.4. □

Chapter 5

Expressive power

Some of the bio-inspired languages introduced in Chapt. 2 differ from classical process calculi because they are devised from the beginning for biology and aim to overcome some limitations by adding or deleting primitives and operators, and by developing new conceptual tools. An interesting question is whether and how those modifications affect the ability of these languages to act as computational devices. Some examples of these investigations can be found in [7, 22, 14].

The main goal of this chapter is to investigate the computational power of BlenX. Turing equivalence results for well-known process calculi like the π-calculus [72, 99] and Mobile Ambients [8, 69] rely on encodings of Turing equivalent formalisms using some high-powered features like restriction operator and name passing in combination with operators like replication, recursion or recursive definitions. In BlenX, the restriction operator is not present and replication is guarded by an action; hence none of the classic results can be directly applied. For these reasons, we decided to start by first developing on a core subset denoted by B_{core}, which considers only primitives for communication. By using the theory of *well-structured transition systems* [36], we show that for B_{core} the termination is decidable. Then we add specific features of BlenX and show that the resulting languages are Turing equivalent. In particular, we prove that by adding either priorities to B_{core} (we denote this subset with B^{p}_{core}) or events (we denote this subset with B^{e}_{core}) we obtain Turing equivalence. We show this by providing encodings of Random Access Machines (RAMs), a well known Turing equivalent formalism, into B^{p}_{core} and B^{e}_{core}.

5.1 The core subset

Here we introduce B_{core}, the core subset on which we base our investigation of the computational power of BlenX. To render easier the presentation of our results we consider both a simplified syntax and semantics with respect to the one of full BlenX. However, all

is done in such a way that all the results we will show can be easily rephrased in terms of the original syntax and semantics.

The syntax of B_{core} is defined in the following way:

$$\begin{array}{lcl} B & ::= & \mathsf{Nil} \mid I[P] \mid B \parallel B \\ I & ::= & \oplus(x, \Delta) \mid \oplus(x, \Delta)\, I \\ P & ::= & M \mid P|P \\ M & ::= & \mathsf{nil} \mid \pi.\, P \mid *\,\pi.\, P \mid M + M \\ \pi & ::= & x?y \mid x!y \end{array}$$

We use here the same symbols and terminology we used for the full BlenX in Chapt. 4. We confine the new interpretations of symbols and definitions only to this chapter.

With $\mathcal{B}_{core}$ and $\mathcal{I}_{core}$ we denote the set of all the possible bio-processes and interfaces of B_{core}, respectively. Moreover, in this case the set of systems $\mathcal{S}_{core}$ coincides with $\mathcal{B}_{core}$ because no complexes and events are considered. Note that since we do not consider here complexes, we also avoid identifiers in the definition of boxes.

Definition 5.1.1. *The function* $\mathsf{sub_t} : \mathcal{B}_{core} \to 2^{\mathcal{N}}$ *is defined as follows*

$$\begin{array}{lcl} \mathsf{sub_t}(I[P]) & = & \mathsf{sub}(I) \\ \mathsf{sub_t}(B_1 \parallel B_2) & = & \mathsf{sub_t}(B_1) \cup \mathsf{sub_t}(B_2) \end{array}$$

Function $\mathsf{sub_t}(\cdot)$ returns the total set of subjects present in all the boxes interfaces composing a bio-process. We simplify the definition of well-formedness by simply saying that a well-formed interface I is a non-empty string of interfaces where subjects and sorts are all distinct and by stating the following definition.

Definition 5.1.2. *Let* $B = I_1[P_1] \parallel \cdots \parallel I_n[P_n]$ *be a bio-process. We say that* B *is well-formed if* $\forall i \in \{1, ..., n\}$ *the interface* I_i *is well-formed.*

We denote with $\overline{\mathcal{S}}_{core}$ the set of all well-formed systems. In particular, for the B_{core} subset we have that $\overline{\mathcal{S}}_{core} \subset \mathcal{B}_{core}$. Definitions of free names for bio-processes and processes result immediately by adapting the ones defined in Tab. 4.3.

The dynamics of a B_{core} system is formally specified through the reduction semantics in Tab. 5.1 which uses a notion of structural congruence $\equiv$.

Definition 5.1.3. *Structural congruence on processes, denoted* $\equiv_p$*, is the smallest congruence on processes that satisfies the axioms in Tab. 5.1(a).*

Structural congruence on bio-processes, denoted $\equiv_b$*, is the smallest congruence that satisfies the axioms in Tab. 5.1(b).*

Structural congruence over systems, denoted $\equiv$, coincides with $\equiv_b$.

(intra) $I[\,x!z.\,P_1 + M_1 \,|\, x?w.\,P_2 + M_2 \,|\, P_3\,] \to I[\,P_1 \,|\, P_2\{z/w\} \,|\, P_3\,]$

(inter) $$\frac{P_1 \equiv_p x!z.\,R_1 + M_1 \,|\, Q_1 \quad P_2 \equiv_p y?w.\,R_2 + M_2 \,|\, Q_2}{I_1[\,P_1\,] \parallel I_2[\,P_2\,] \to I_1[\,R_1 \,|\, Q_1\,] \parallel I_2[\,R_2\{z/w\} \,|\, Q_2\,]}$$
where $I_1 = \oplus(x,\,\Delta)\,I_1^*$ and $I_2 = \oplus(y,\,\Gamma)\,I_2^*$
and provided $\alpha(\Gamma,\Delta) > 0$ and $z \notin \mathsf{sub}(I_1) \cup \mathsf{sub}(I_2)$

(struct) $$\frac{B_1 \equiv_b B_1' \quad B_1' \to B_2' \quad B_2' \equiv_b B_2}{B_1 \to B_2}$$

(redex) $$\frac{B \to B'}{B \parallel B_1 \to B' \parallel B_1}$$

---

a) axioms for processes

1. $P =_\alpha P' \Rightarrow P \equiv_p P'$
2. $P \,|\, \mathsf{nil} \equiv_p P$
3. $P_1 \,|\, P_2 \equiv_p P_2 \,|\, P_1$
4. $P_1 \,|\, (P_2 \,|\, P_3) \equiv_p (P_1 \,|\, P_2) \,|\, P_3$
5. $M + \mathsf{nil} \equiv_p M$
6. $M_1 + M_2 \equiv_p M_2 + M_1$
7. $M_1 + (M_2 + M_3) \equiv_p (M_1 + M_2) + M_3$
8. $*\pi.\,P \equiv_p \pi.\,(P \mid *\pi.\,P)$

b) axioms for bio-processes

1. $B \parallel \mathsf{Nil} \equiv_b B$
2. $B_1 \parallel B_2 \equiv_b B_2 \parallel B_1$
3. $B_1 \parallel (B_2 \parallel B_3) \equiv_b (B_1 \parallel B_2) \parallel B_3$
4. $P_1 \equiv_p P_2 \Rightarrow I_1^* I_2 I_3 I_4^*[\,P_1\,] \equiv_b I_1^* I_3 I_2 I_4^*[\,P_2\,]$
5. $P_1 \equiv_p P_2 \wedge y \notin \mathsf{fn}(P_2) \cup \mathsf{sub}(I^*) \Rightarrow$
   $\oplus(x,\,\Delta)^p\,I^*[\,P_1\,] \equiv_b \oplus(y,\,\Delta)^p\,I^*[\,P_2\{y/x\}\,]$

Table 5.1: Reduction semantics of B_{core}.

It is evident how the reduction semantics in Tab. 4.13 is a simplification of the one presented in Tab.4.13, that highlights only the behavioural aspects of the subset here considered. Note that being not interested in complexes, here we assume to have a compatibility function $\alpha : \mathcal{T}^2 \to \mathbb{N}$ that, given a pair of sorts Δ and Γ returns only a value; if the resulting value is greater than zero, then inter-communications over interfaces having sorts Δ and Γ are allowed.

All the results we proved in the previous chapter for the full BlenX are still valid for the subset B_{core}.

5.2 A decidability result

In this section we show that termination is decidable for the B_{core} subset. With respect to the methods used in this section, we take inspiration from [7, 8] in which decidability results for π-calculus, Pure Mobile Ambients and Brane Calculi have been presented and we rely on the theory of well-structured transition systems [36]. In particular, the existence of an infinite computation starting from a given state is decidable for finitely branching transition systems, provided that the set of states can be equipped with a *well-quasi-ordering*. The main differences with the results contained in [7, 8] are that in our language we have no static hierarchies of ambients and nested restrictions, but we have a two level hierarchy of boxes and processes and a form of name-passing over finite sets of names.

The decidability of termination for B_{core} is proved by first providing an alternative labelled transition semantics for a subset $\overline{\mathcal{S}}_{safe}$ of B_{core} bio-processes, that we call *safe*, and then by showing that there is a correspondence of this semantics with the reduction semantics presented in Sec. 5.1. In particular, we show that it is always possible and easy to transform a generic B_{core} bio-process into an equivalent *safe* one and that a bio-process admits an infinite computation according to the reduction semantics if and only if one of its corresponding safe bio-processes admits an infinite sequence of τ transitions according to the new labelled transition semantics.

Then, we define a quasi-ordering $\preceq_b$ on bio-processes which is strongly compatible with $\xrightarrow{\tau}$, we show that the relation $\preceq_b$ is a well-quasi-ordering and finally we prove that the termination of bio-processes in $\overline{\mathcal{S}}_{safe}$ is decidable.

5.2.1 Well-structured transition systems

Here we recall some basic definitions and results from [36, 47]. A quasi-ordering (qo) is a reflexive and transitive relation.

Definition 5.2.1. *A well-quasi-ordering (wqo) on a set X is a qo $\leq$ such that any infinite sequence of elements $x_0, x_1, x_2, ...$ from X contains an increasing pair $x_i \leq x_j$ with $i < j$. The set X is said to be well-quasi-ordered, or wqo for short.*

Note that if $\leq$ is a wqo then any infinite sequence $x_0, x_1, x_2, \ldots$ contains an infinite increasing subsequence $x_{i_0}, x_{i_1}, x_{i_2}, \ldots$ (with $i_0 \leq i_1 \leq i_2 \leq \ldots$). Thus well-quasi-orders exclude the possibility of having infinite strictly decreasing sequences.

Definition 5.2.2. *A transition system is a tuple $TS = (S, \rightarrow)$ where S is a set of states and $\rightarrow \subseteq S \times S$ is a set of transitions. If $p, q \in S$, then $(p, q) \in \rightarrow$ is usually written as $p \rightarrow q$.*

The set $\{s' \in S \mid s \rightarrow s'\}$ of immediate successors of a state $s \in S$ is denoted with $Succ(s)$. TS is finite branching if $Succ(s)$ is finite for all $s \in S$.

Definition 5.2.3. *A well-structured transition system with strong compatibility, denoted with $TS = (S, \rightarrow, \leq)$ is a transition system equipped with a qo $\leq$ on S such that the following conditions hold:*

- *$\leq$ is a well-quasi-ordering*
- *$\leq$ is (upward) compatible with $\rightarrow$, i.e., for all $s_1 \leq t_1$ and all transitions $s_1 \rightarrow s_2$, there exists a state t_2 such that $t_1 \rightarrow t_2$ and $s_2 \leq t_2$ (strong compatibility).*

Our decidability result is based on the following theorem [36]:

Theorem 5.2.4. *Let $TS = (S, \rightarrow, \leq)$ be a finitely branching, well-structured transition system with decidable $\leq$ and computable Succ. The existence of an infinite computation starting from a state $s \in S$ is decidable.*

In order to prove that the qo we will define on bio-processes is a wqo, we need to introduce some important results proved by Higman in [47]. First of all we recall that given a set S, the set S^* denotes the set of finite sequences of elements in S.

Definition 5.2.5. *Let S be a set and $\leq$ a wqo over S. The relation $\leq_*$ over S^* is defined as follows. Let $t, u \in S^*$, with $t = t_1...t_m$ and $u = u_1...u_n$. We have that $t \leq_* u$ iff there exists an injection f from $\{1, ..., m\}$ to $\{1, ..., n\}$ such that $t_i \leq u_{f(i)}$ and $i \leq f(i)$ for $i = 1, ..., m$.*

Theorem 5.2.6. *[Higman] Let S be a set and $\leq$ be a wqo over S. Then, the relation $\leq_*$ is a wqo over S^*.*

Lemma 5.2.7. *Let S be a finite set. Then equality is a wqo over S.*

5.2.2 A labelled transition semantics for the core subset

Here we define a labelled transition semantics for B_{core} (Tab. 5.2) to get rid of structural congruence. Axioms and rules for processes are in the style of the transition semantics reported in [99] (page 38) for the π-calculus, and hence some results there reported can be reused.

$$\text{(pi\_in) } x?w.\,P \xrightarrow{x?y} P\{y/w\} \qquad \text{(rep\_in) } *x?w.\,P \xrightarrow{x?y} P\{y/w\} \mid *x?w.\,P$$

$$\text{(pi\_out) } x!y.\,P \xrightarrow{x!y} P \qquad \text{(rep\_out) } *x!y.\,P \xrightarrow{x!y} P \mid *x!y.\,P$$

$$\text{(l\_pi\_sum) } \frac{M_0 \xrightarrow{\theta} M_0'}{M_0 + M_1 \xrightarrow{\theta} M_0'} \qquad \text{(l\_pi\_par) } \frac{P_0 \xrightarrow{\theta} P_0'}{P_0 \mid P_1 \xrightarrow{\theta} P_0' \mid P_1} \qquad \text{(l\_intra) } \frac{P \xrightarrow{x!y} P' \quad Q \xrightarrow{x?y} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\text{(bio\_out) } \frac{P \xrightarrow{x!y} P'}{I[\,P\,] \xrightarrow{\Delta x!y} I[\,P'\,]}, \text{ provided } y \notin \mathsf{sub}(I) \text{ and } \oplus(x,\,\Delta) \in I$$

$$\text{(bio\_in) } \frac{P \xrightarrow{x?y} P'}{I[\,P\,] \xrightarrow{\Delta x?y} I[\,P'\,]}, \text{ provided } y \notin \mathsf{sub}(I) \text{ and } \oplus(x,\,\Delta) \in I$$

$$\text{(pi\_bio) } \frac{P \xrightarrow{\tau} P'}{I[\,P\,] \xrightarrow{\tau} I[\,P'\,]} \qquad \text{(l\_bio\_par) } \frac{B_0 \xrightarrow{\theta} B_0'}{B_0 \parallel B_1 \xrightarrow{\theta} B_0' \parallel B_1}$$

$$\text{(l\_inter) } \frac{B_0 \xrightarrow{\Delta x!z} B_0' \quad B_1 \xrightarrow{\Gamma y?z} B_1'}{B_0 \parallel B_1 \xrightarrow{\tau} B_0' \parallel B_1'}, \text{ provided } \alpha(\Delta,\Gamma) > 0$$

Table 5.2: Labelled transition semantics of B_{core} (we omit the symmetric rules r_pi_sum, r_pi_par, r_intra, r_bio_par and r_inter).

We use the meta-variable θ to range over $x?y$, $x!y$, $\Delta x?y$, $\Delta x!y$, and τ. The set of names, $\mathsf{n}(\theta)$, of θ is $\mathsf{fn}(\theta) \cup \mathsf{bn}(\theta)$. Table 5.3 contains the terminology and notation for labels.

Note that the semantics we define is not equivalent to the one presented in Sect. 5.1, because of the absence of rules for managing α-conversion. We do not explicitly consider α-convertible bio-processes to get rid of the infinite names over *intra-communication* that α-conversion introduces. This fact will be used to obtain the wqo over bio-processes.

However, we will show that there is a correspondence between the labelled transition semantics over the *safe* subset of B_{core} bio-processes and the reduction semantics over B_{core} bio-processes. Moreover, although the labelled transition semantics is not finite branching, we will show that the transition systems constructed over safe bio-processes by only considering $\xrightarrow{\tau}$ transitions are finite branching. This fact is essential to use the theory of well-structured transition systems.

Safe bio-processes are introduced to guarantee that no behaviours are lost when we get

θ	kind	$\mathsf{fn}(\theta)$	$\mathsf{bn}(\theta)$	$\mathsf{n}(\theta)$
$x?y$	process input	$\{x, y\}$	$\emptyset$	$\{x, y\}$
$x!y$	process output	$\{x, y\}$	$\emptyset$	$\{x, y\}$
$\Delta x?y$	box input over Δ	$\{x, y\}$	$\emptyset$	$\{x, y\}$
$\Delta x!y$	box output over Δ	$\{x, y\}$	$\emptyset$	$\{x, y\}$
τ	internal	$\emptyset$	$\emptyset$	$\emptyset$

Table 5.3: Terminology and notation for action labels.

rid of structural congruence. Suppose $\alpha(\Delta, \Delta) > 0$ and consider the following bio-process:

$$B = \oplus(x,\, \Delta)[\, x!y.\mathsf{nil}\,] \parallel \oplus(x,\, \Delta)\oplus(y,\, \Gamma)[\, x?z.z!k.\mathsf{nil} \mid y?z.\mathsf{nil}\,]$$

To avoid captures in inter-communications, rule (inter) in Tab. 5.1 requires that $y \notin \mathsf{sub}(\oplus(x,\, \Delta)) \cup \mathsf{sub}(\oplus(x,\, \Delta)\oplus(y,\, \Gamma))$, which in this case does not hold. Hence, in order to consume the inter-communication we have to consider (one among infinitely many others) the bio-process:

$$B' = \oplus(x,\, \Delta)[\, x!y.\mathsf{nil}\,] \parallel \oplus(x,\, \Delta)\oplus(n,\, \Gamma)[\, x?z.z!k.\mathsf{nil} \mid n?z.\mathsf{nil}\,]$$

and derive the transition through the rule struct, i.e., structural law (b.5) in Tab. 5.1 implies $B \equiv_b B'$. Safe bio-processes guarantee that we never need to apply the structural law (b.5) in order to derive an inter-communication, simplifying the definition of our labelled transition semantics.

Definition 5.2.8. *The bio-process B is safe iff $\mathsf{fn}(B) \cap \mathsf{sub_t}(B) = \emptyset$.*

We denote with $\overline{\mathcal{S}}_{safe} \subset \overline{\mathcal{S}}_{core}$ the set of safe well-formed bio-processes.

Lemma 5.2.9. *Let $B \in \overline{\mathcal{S}}_{core}$. There exists $B' \in \overline{\mathcal{S}}_{safe}$ such that $B \equiv_b B'$.*

Proof. Immediate from the structural congruence rules reported in Tab. 5.1 (group b). □

We denote with $safe(B) \subset \overline{\mathcal{S}}_{safe}$ the set of safe bio-processes structurally congruent to a generic bio-process $B \in \overline{\mathcal{S}}_{core}$. Given a bio-process $B \in \overline{\mathcal{S}}_{core}$, the problem of finding an equivalent safe bio-process B', is decidable and efficiently solvable. Indeed, considering the finite set $\mathsf{fn}(B)$ of free names in B and the number m of interfaces of B (both the set and the value can be computed linearly in the size of B), a safe bio-process B' structurally congruent to B can be obtained simply by substituting all the interface subjects of B with names contained in a set $M \subset \mathcal{N}$ such that $M \cap \mathsf{fn}(B) = \emptyset$ and $|M| = m$. The problem of finding this set M is efficiently computable.

Now, we show that the transition system $(\overline{\mathcal{S}}_{safe}, \xrightarrow{\tau})$ is finite branching. In order to do this we first have to show that the safe property of bio-processes is preserved over τ transitions. However, some preliminary results are needed. The first result describes how, given a transition $P \xrightarrow{\theta} P'$ over processes or a transition $B \xrightarrow{\theta} B'$ over bio-processes, the free names of P' and B' are the finite set made up of the free names of P and B and the names in θ.

Lemma 5.2.10. *Let $P, P' \in \mathcal{P}$. Suppose $P \xrightarrow{\theta} P'$, then*

(a) if $\theta = x!y$ then $x, y \in \mathsf{fn}(P)$ and $\mathsf{fn}(P') \subseteq \mathsf{fn}(P)$

(b) if $\theta = x?y$ then $x \in \mathsf{fn}(P)$ and $\mathsf{fn}(P') \subseteq \mathsf{fn}(P) \cup \{y\}$

(c) if $\theta = \tau$ then $\mathsf{fn}(P') \subseteq \mathsf{fn}(P)$

Proof. See [99], page 44. □

Lemma 5.2.11. *Let $B, B' \in \overline{\mathcal{S}}_{safe}$. Suppose $B \xrightarrow{\theta} B'$. Then*

(a) if $\theta = \Delta x!y$ then $y \in \mathsf{fn}(B)$ and $\mathsf{fn}(B') \subseteq \mathsf{fn}(B)$

(b) if $\theta = \Delta x?y$ then $\mathsf{fn}(B') \subseteq \mathsf{fn}(B) \cup \{y\}$

(c) if $\theta = \tau$ then $\mathsf{fn}(B') \subseteq \mathsf{fn}(B)$

Proof. By induction on the inference of $B \xrightarrow{\theta} B'$. We present only the most relevant cases. The other cases can be proved similarly.
(a)
(Case bio_out) We have $B = I[P]$ and $B' = I[P']$. By definition we have $\mathsf{fn}(B) = \mathsf{fn}(P) \setminus \mathsf{sub}(I)$ and $\mathsf{fn}(B') = \mathsf{fn}(P') \setminus \mathsf{sub}(I)$. By hypothesis we have $P \xrightarrow{x!y} P'$ and hence, by Lemma 5.2.10, $x, y \in \mathsf{fn}(P)$ and $\mathsf{fn}(P') \subseteq \mathsf{fn}(P)$. Therefore $\mathsf{fn}(P') \setminus \mathsf{sub}(I) \subseteq \mathsf{fn}(P) \setminus \mathsf{sub}(I)$, which means $\mathsf{fn}(B') \subseteq \mathsf{fn}(B)$. Moreover, since $y \in \mathsf{fn}(P)$ and $y \notin \mathsf{sub}(I)$ we have $y \in \mathsf{fn}(B)$.
(Case l_bio_par) By definition $\mathsf{fn}(B_0 || B_1) = \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1)$. By inductive hypothesis we have $y \in \mathsf{fn}(B_0)$ and $\mathsf{fn}(B_0') \subseteq \mathsf{fn}(B_0)$. Therefore $\mathsf{fn}(B_0') \cup \mathsf{fn}(B_1) \subseteq \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1)$ which means $\mathsf{fn}(B') \subseteq \mathsf{fn}(B)$. Moreover, since $y \in \mathsf{fn}(B_0)$, we have $y \in \mathsf{fn}(B)$.
(b)
(Case bio_in) We have $B = I[P]$ and $B' = I[P']$. By definition we have $\mathsf{fn}(B) = \mathsf{fn}(P) \setminus \mathsf{sub}(I)$ and $\mathsf{fn}(B') = \mathsf{fn}(P') \setminus \mathsf{sub}(I)$. By hypothesis we have $P \xrightarrow{x?y} P'$ and hence, by Lemma 5.2.10, $x \in \mathsf{fn}(P)$ and $\mathsf{fn}(P') \subseteq (\mathsf{fn}(P) \cup \{y\})$. Therefore $(\mathsf{fn}(P') \setminus \mathsf{sub}(I)) \subseteq (\mathsf{fn}(P) \setminus \mathsf{sub}(I)) \cup \{y\}$, which means $\mathsf{fn}(B') \subseteq \mathsf{fn}(B) \cup \{y\}$.
(c)
(Case l_inter) By definition $\mathsf{fn}(B_0 || B_1) = \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1)$. By inductive hypothesis we

have $z \in \mathsf{fn}(B_0)$, $\mathsf{fn}(B_0') \subseteq \mathsf{fn}(B_0)$, and $\mathsf{fn}(B_1') \subseteq \mathsf{fn}(B_1) \cup \{z\}$. Since $z \in \mathsf{fn}(B_0)$ we have $\mathsf{fn}(B_0) \cup \mathsf{fn}(B_1) \cup \{z\} = \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1)$ and therefore $\mathsf{fn}(B_0') \cup \mathsf{fn}(B_1') \subseteq \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1)$, which means $\mathsf{fn}(B') \subseteq \mathsf{fn}(B)$. □

Lemma 5.2.12. *Let $B \in \overline{\mathcal{S}}_{safe}$. Then $B \xrightarrow{\tau} B'$ implies $B' \in \overline{\mathcal{S}}_{safe}$.*

Proof. By hypothesis we have $\mathsf{fn}(B) \cap \mathsf{sub_t}(B) = \emptyset$. Since $B \in \overline{\mathcal{S}}_{safe} \subset \overline{\mathcal{S}}_{core}$ and no reduction rule can change an interface subject or sort we immediately have $B' \in \overline{\mathcal{S}}_{core}$ and $\mathsf{sub_t}(B) = \mathsf{sub_t}(B')$. Moreover, by Lemma 5.2.11, we have $\mathsf{fn}(B') \subseteq \mathsf{fn}(B)$ and as a consequence $\mathsf{fn}(B') \cap \mathsf{sub_t}(B') = \emptyset$. Hence the Lemma holds. □

Now, we recall some results on image-finiteness of π-calculus processes, reported in [99] (page 45) that are still valid for our processes.

Lemma 5.2.13. *Let $P \in \mathcal{P}$. Then*
(1) There are only finitely many x such that $P \xrightarrow{x?z} P'$ for some z and P'.
(2) There are only finitely many triples x, y, P' such that $P \xrightarrow{x!y} P'$.

These results can be used to show that for any process P there are only finitely many processes Q such that $P \xrightarrow{\tau} Q$ and that for any safe bio-process B there are only finitely many safe bio-processes B' such that $B \xrightarrow{\tau} B'$.

Lemma 5.2.14. *Let $P \in \mathcal{P}$. Then the set $Succ(P) = \{P' \in \mathcal{P} \mid P \xrightarrow{\tau} P'\}$ is finite.*

Proof. By induction on the structure of P.
(Induction base) If P has the form nil, $x?y.P'$, $x!y.P'$, $*x?y.P'$ or $*x!y.P'$ it is simple to see that no τ actions can be derived using the semantics rules. Hence, in this cases the set is $\{P' \in \mathcal{P} \mid P \xrightarrow{\tau} P'\} = \emptyset$.
(Case $P = M_0 + M_1$) By inductive hypothesis the sets $Succ(M_0)$ and $Succ(M_1)$ are finite. Since no intra-communications between processes M_0 and M_1 can be performed, then the set $\{P' \in \mathcal{P} \mid P \xrightarrow{\tau} P'\} = Succ(M_0) \cup Succ(M_1)$ is finite.
(Case $P = P_0 \mid P_1$) By inductive hypothesis the sets $Succ(P_0)$ and $Succ(P_1)$ are finite. However, P_0 and P_1 are parallel processes and hence they can synchronize on inputs and outputs actions and perform intra-communications, generating τ transitions. By Lemma 5.2.13, we obtain that the number of possible input and output in P_0 and P_1 on a channel x is finite and hence only a finite number of τ actions (using (l_intra) and (r_intra) rules) can be derived. Therefore, by inductive hypothesis and Lemma 5.2.13 the set $Succ(P)$ is finite. □

We extend results of Lemma 5.2.13 and Lemma 5.2.14 to bio-processes.

Lemma 5.2.15. *Let $B \in \mathcal{B}_{core}$. Then*
(1) There are only finitely many pairs x, Δ such that $B \xrightarrow{\Delta x?z} B'$ for some z and B'.
(2) There are only finitely many tuples Δ, x, y, B' such that $B \xrightarrow{\Delta x!y} B'$.

Proof. By induction on the structure of B.
(1)
(Case Nil) No couple x, Δ such that $B \xrightarrow{\Delta x?z} B'$ for some z and B' exists.
(Case $I[P]$) For each subject $x \in \mathsf{sub}(I)$, we have (by Lemma 5.2.13) that there are only finitely many x such that $P \xrightarrow{x?z} P'$ for some z and P' and hence only finitely many x such that $P \xrightarrow{x?z} P'$ for some $z \notin \mathsf{sub}(I)$ and P'. Since by definition I is well-formed and the set of interfaces subjects $\mathsf{sub}(I)$ is finite, we obtain (by the application of rule (bio_in)) that there are only finitely many pairs Δ, x such that $I[P] \xrightarrow{\Delta x?z} I[P']$ for some z and $I[P']$.
(Case $B \parallel B'$) By inductive hypothesis on B and B' the Lemma follows immediately.
(2)
(Case Nil) No tuple Δ, x, y, B' such that $B \xrightarrow{\Delta x!y} B'$ exists.
(Case $I[P]$) For each subject $x \in \mathsf{sub}(I)$, we have (by Lemma 5.2.13) that there are only finitely many tuples x, y, P' such that $P \xrightarrow{x!y} P'$ and hence there are only finitely many tuples x, y, P' such that $P \xrightarrow{x!y} P'$ and $y \notin \mathsf{sub}(I)$. Since by definition I is well-formed and the set of interface subjects $\mathsf{sub}(I)$ is finite, we obtain (by the application of rule (bio_out)) that there are only finitely many tuples $\Delta, x, y, I[P']$ such that $I[P] \xrightarrow{\Delta x!y} I[P']$.
(Case $B \parallel B'$) By inductive hypothesis on B and B' the Lemma follows immediately. □

Lemma 5.2.16. *Let $B \in \overline{\mathcal{S}}_{safe}$. Then the set $Succ(B) = \{B' \in \overline{\mathcal{S}}_{safe} \mid B \xrightarrow{\tau} B'\}$ is finite.*

Proof. By induction on the structure of B.
(Case Nil) In this case $Succ(B)$ is obviously $\emptyset$.
(Case $I[P]$) By Lemma 5.2.14 the set $Succ(P)$ is finite. It immediately follows (by application of rule (pi_bio)) that the set $Succ(I[P])$ is finite.
(Case $B_0 \parallel B_1$) By inductive hypothesis the sets $Succ(B_0)$ and $Succ(B_1)$ are finite. However, B_0 and B_1 are parallel bio-processes and hence they can synchronize on inputs and outputs actions over compatible interfaces and perform inter-communications, generating τ transitions. By Lemma 5.2.15, the number of possible input and output over compatible interfaces between B_0 and B_1 is finite and hence only a finite number of τ actions (using (l_inter) and (r_inter) rules) can be derived. Therefore, by inductive hypothesis and Lemma 5.2.13, the set $Succ(B)$ is finite. □

Corollary 5.2.17. *The transition system $(\overline{\mathcal{S}}_{safe}, \xrightarrow{\tau})$ is finite branching.*

Proof. Immediate by Lemma 5.2.16. □

To reason on the new labelled semantics we need to show its correspondence with the reduction semantics. In particular, we show that the τ transition relation over safe bio-processes and the reduction relation over bio-processes agree.

Lemma 5.2.18. *Let $B, B' \in \overline{\mathcal{S}}_{safe}$. If $B \equiv_b B'$ and $B \xrightarrow{\tau} B''$, then for some $B''' \in \overline{\mathcal{S}}_{safe}$ we have that $B' \xrightarrow{\tau} B'''$ and $B'' \equiv_b B'''$.*

Proof. By induction on the length of the derivation $B \equiv_b B'$. □

Theorem 5.2.19. *Let $B \in \overline{\mathcal{S}}_{core}$ and $B'' \in safe(B)$. Then $B \to B'$ iff $B'' \xrightarrow{\tau} B'''$ and $B''' \in safe(B')$.*

Proof. ($\Rightarrow$)
Similarly to Lemma 4.4.2, if $B \to B'$ then there exist bio-processes $B_0 \equiv_b B$ and $B_1 \equiv_b B'$ such that

$$B_0 = I[\, x!z.\, P_1 + M_1 \,|\, x?w.\, P_2 + M_2 \,|\, P_3 \,] \parallel B_2$$
$$\text{and } B_1 = I[\, P_1 \,|\, P_2\{z/w\} \,|\, P_3 \,] \parallel B_2$$

or

$$B_0 = \oplus(x, \Delta)\, I_1^*[\, x!z.\, R_1 + M_1 \,|\, Q_1 \,] \parallel \oplus(y, \Gamma)\, I_2^*[\, y?w.\, R_2 + M_2 \,|\, Q_2 \,] \parallel B_2$$
$$\text{and } B_1 = I_1[\, R_1 \,|\, Q_1 \,] \parallel I_2[\, R_2\{z/w\} \,|\, Q_2 \,] \parallel B_2$$

with $\alpha(\Gamma, \Delta) > 0$ and in the second case $z \notin \mathsf{sub}(I_1^*) \cup \{x\} \cup \mathsf{sub}(I_2^*) \cup \{y\}$. By Lemma 5.2.9, in both cases there exist safe bio-processes $B_0^s \equiv_b B_0$ and $B_1^s \equiv_b B_1$. In particular, in the second case we have that B_0^s is such that $z \notin \mathsf{sub}(I_1^*) \cup \mathsf{sub}(I_2^*) \cup \mathsf{sub_t}(B_2) \cup \{x, y\}$. It is easy to see that in both cases we can derive $B_0^s \xrightarrow{\tau} B_1^s$. Moreover, $B_0^s \equiv_b B_0 \equiv_b B$ means that $B_0^s \in safe(B)$ and hence $B'' \equiv_b B_0^s$. By Lemma 5.2.18, there exists $B''' \in \overline{\mathcal{S}}_{safe}$ such that $B''' \equiv_b B_1^s$ and from $B''' \equiv_b B_1^s \equiv_b B_1 \equiv_b B'$ we obtain $B''' \in safe(B')$.
($\Leftarrow$)
It is enough to show that $B \xrightarrow{\tau} B'$ implies $B \to B'$. The proof is by induction on the inference of $B \xrightarrow{\tau} B'$.
(Case (l_inter)) By hypothesis we know that $B_0 \xrightarrow{\Delta x!z} B_0'$, $B_1 \xrightarrow{\Gamma x?z} B_1'$. Notice that :

1) if $B_0 \xrightarrow{\Delta x!z} B_0'$ then $B_0 \equiv_b \oplus(x, \Delta)\, I_0^*[\, x!z.Q + M \mid R \,] \parallel B_2$, $B_0' \equiv_b \oplus(x, \Delta)\, I_0^*[\, Q \mid R \,] \parallel B_2$ and $z \notin \mathsf{sub}(I_0^*) \cup \{x\}$.

2) if $B_1 \xrightarrow{\Gamma y?z} B_1'$ then $B_1 \equiv_b \oplus(y, \Gamma)\, I_1^*[\, y?w.T' + N \mid T \,] \parallel B_3$, $B_1' \equiv_b \oplus(y, \Gamma)\, I_1^*[\, T'\{z/w\} \mid T \,] \parallel B_3$ and $z \notin \mathsf{sub}(I_1^*) \cup \{y\}$.

Hence, the bio-process $B_0 \parallel B_1$ is structurally congruent ($\equiv_b$) to

$$\oplus(x, \Delta)\, I_0^*[\, x!z.Q + M \mid R \,] \parallel \oplus(y, \Gamma)\, I_1^*[\, y?w.T' + N \mid T \,] \parallel B_2 \parallel B_3$$

and by applying (inter) and (redex) rules of the reduction semantics we can derive the transition

$$B_0 \parallel B_1 \rightarrow \oplus(x, \Delta)\, I_0^*[\, Q \mid R \,] \parallel \oplus(y, \Gamma)\, I_1^*[\, T'\{z/w\} \mid T \,] \parallel B_2 \parallel B_3,$$

where the resulting bio-process is structurally congruent to $B_0' \parallel B_1'$. By applying the rule (struct) of the reduction semantics we are done.
(Case (pi_bio)) By hypothesis we have $P \xrightarrow{\tau} P'$. Notice that if $P \xrightarrow{\tau} P'$ than $P \equiv_p x!z.Q + N \mid x?w.T' + M \mid R$ and $P' \equiv_p Q \mid T'\{z/w\} \mid R$. By applying the (intra) rule of the reduction semantics we can derive the transition $I[P] \rightarrow I[\, Q \mid T'\{z/w\} \mid R \,]$, where the resulting bio-process is structurally congruent to $I[P']$. By applying the rule (struct) of the reduction semantics we end the proof.
(Case (l_bio_par)) By inductive hypothesis we have $B_0 \xrightarrow{\tau} B_0'$ implies $B_0 \rightarrow B_0'$ and hence, by applying the rule (redex) of the reduction semantics we derive $B_0 \parallel B_1 \rightarrow B_0' \parallel B_1$. □

As consequence of Theorem 5.2.19, a bio-process $B \in \overline{\mathcal{S}}_{core}$ admits an infinite computation according to the reduction semantics if and only if a corresponding bio-process $B' \in \mathit{safe}(B)$ admits an infinite sequence of τ transitions according to the labelled transition semantics. In particular, B' is terminating according to the labelled transition semantics if an infinite sequence of τ transitions starting from B' does not exists.

Corollary 5.2.20. *Let $B \in \overline{\mathcal{S}}_{core}$ and $B' \in \mathit{safe}(B)$. The bio-process B terminates according to the reduction semantics iff B' terminates according to the labelled transition semantics.*

In the reminder of the chapter we consider only safe bio-processes in $\overline{\mathcal{S}}_{safe}$.

5.2.3 Decidability of termination for the core subset

Here we show that the existence of an infinite computation over safe bio-processes is computable in B_{core}. We equip the labelled transition system $(\overline{\mathcal{S}}_{safe}, \xrightarrow{\tau})$ with a qo $\preceq_b$ on bio-processes which turns out to be wqo compatible with $\xrightarrow{\tau}$. Then we show that termination is decidable.

Definition 5.2.21. *Let $B \in \overline{\mathcal{S}}_{safe}$. The set of bio-processes reachable from B with a sequence of τ transitions is:*

$$\mathit{Deriv}(B) = \{B' \mid B \xrightarrow{\tau}{}^{*} B'\}$$

In order to define the qo $\preceq_b$, we first introduce a simplified structural congruence relation which is compatible with $\xrightarrow{\theta}$. This relation captures only reordering of sums, the

monoidal laws for the parallel composition of processes and bio-processes and reordering of interfaces.

Definition 5.2.22. *The $\equiv_p^{dec}$ congruence relation over processes, is the smallest relation which satisfies the laws a.2, a.3, a.4, a.5, a.6, a.7 in Tab. 5.1 and the $\equiv_b^{dec}$ congruence relation over bio-processes, is the smallest relation which satisfies the laws b.1, b.2, b.3.*

Since $\equiv_p^{dec} \subset \equiv_p$ and $\equiv_b^{dec} \subset \equiv_b$ all the previous results on safe bio-processes hold also for the simplified structural congruence.

Lemma 5.2.23. *Let $B_0, B_1 \in \overline{\mathcal{S}}_{safe}$. If $B_0 \equiv_b^{dec} B_1$ and $B_1 \xrightarrow{\theta} B_1'$ then there exists B_0' such that $B_0 \xrightarrow{\theta} B_0'$ and $B_0' \equiv_b^{dec} B_1'$.*

Proof. By induction on the length of the derivation $B_0 \equiv_b^{dec} B_1$. □

We can now introduce a quasi-order $\preceq_b$ which will be proven to be a well-quasi-order.

Definition 5.2.24. *Let $B, B' \in \overline{\mathcal{S}}_{safe}$. We define $B \preceq_b B'$ iff there exist $I_1, \ldots, I_n, P_1, \ldots, P_n, Q_1, \ldots, Q_n, P_1', \ldots, P_n'$ such that $B \equiv_b^{dec} \prod_{i=1}^n I_i[P_i]$, $B' \equiv_b^{dec} \prod_{i=1}^n I_i[Q_i]$ and $Q_i = P_i \mid P_i'$ for $i = 1, \ldots, n$.*

The $\preceq_b$ relation is reflexive, transitive and strongly compatible with $\xrightarrow{\theta}$.

Theorem 5.2.25. *Let $B_0, B_0', B_1 \in \overline{\mathcal{S}}_{safe}$. If $B_0 \xrightarrow{\theta} B_0'$ and $B_0 \preceq_b B_1$ then there exists B_1' such that $B_1 \xrightarrow{\theta} B_1'$ and $B_0' \preceq_b B_1'$.*

Proof. By cases on the inference of $\xrightarrow{\theta}$. We present only one case because all the others are similar.
(Case (bio_out)) We have that $B_0 = I[P] \xrightarrow{\Delta x!y} I[P'] = B_0'$. By definition of $\preceq_b$ we have that there exists a process Q such that $B_1 \equiv_b^{dec} I[P|Q]$. By hypothesis we know that $P \xrightarrow{x!y} P'$ and hence, by applying the rules in Tab. 5.2, we can derive the transition $P|Q \xrightarrow{x!y} P'|Q$ from which we have $B_1 \xrightarrow{\Delta x!y} I[P'|Q] = B_1'$. By $\preceq_b$ definition it results $B_0' \preceq_b B_1'$. □

Corollary 5.2.26. *$\preceq_b$ is strongly compatible with $\xrightarrow{\tau}$.*

We now introduce some auxiliary functions that will be used to prove that $\preceq_b$ is a wqo. The *Sub* function generates the set of all *possible* sequential and replicated sub-processes of a given process.

Definition 5.2.27. *Let $P \in \mathcal{P}$ and $Y \subseteq 2^{\mathcal{N}}$ be a finite set of names. The set of* possible *sequential and replicated sub-processes of P over the set of names Y is defined as:*

$$\begin{aligned}
Sub(\mathsf{nil}, Y) &= \emptyset \\
Sub(x!m.P, Y) &= \{x!m.P\} \cup Sub(P, Y) \\
Sub(x?m.P, Y) &= \{x?m.P\} \cup (\textstyle\bigcup_{n \in Y} Sub(P\{n/m\}, Y)) \\
Sub(M + N, Y) &= \{M + N\} \cup Sub(M, Y) \cup Sub(N, Y) \\
Sub(P \mid Q, Y) &= Sub(P, Y) \cup Sub(Q, Y) \\
Sub(*x!m.P, Y) &= \{*x!m.Y\} \cup Sub(P, Y) \\
Sub(*x?m.P, Y) &= \{*x?m.P\} \cup (\textstyle\bigcup_{n \in Y} Sub(P\{n/m\}, Y))
\end{aligned}$$

The set of processes generated by the application of the function Sub on a process P and a finite set of names Y is finite.

Lemma 5.2.28. *Let $P \in \mathcal{P}$ and $Y \subseteq 2^{\mathcal{N}}$ be a finite set of names. Then $Sub(P, Y)$ is finite.*

Proof. By induction on the structure of P.
(case nil) The empty set is finite.
(case $x!m.P'$) By inductive hypothesis $Sub(P', Y)$ is finite and hence by adding the element $x!m.P'$ the set is still finite.
(case $x?m.P'$) By inductive hypothesis for all $n \in Y$ the set $Sub(P'\{n/m\}, Y)$ is finite. Since Y is finite, then the union of a finite number of finite sets is a finite set. Moreover, by adding the process $x?m.P'$ this set remains finite.
(case $M + N$) By inductive hypothesis $Sub(M, Y)$ and $Sub(N, Y)$ are finite set and hence their union with the set $\{M + N\}$ results in a finite set.
The other cases are similar. □

Corollary 5.2.29. *Let $P \in \mathcal{P}$. Then $Sub(P, \mathsf{fn}(P))$ is finite.*

Proof. From the fact that $\mathsf{fn}(P)$ is a finite set of names and by Lemma 5.2.28. □

Lemma 5.2.30. *Let $P, P' \in \mathcal{P}$. Then $P \equiv_p^{dec} P'$ implies $Sub(P, \mathsf{fn}(P)) = Sub(P', \mathsf{fn}(P'))$.*

Proof. By induction on the length of the derivation $P \equiv_p^{dec} P'$. □

We now prove some useful properties of the function Sub.

Lemma 5.2.31. *Let $P \in \mathcal{P}$ and $Y, Y' \subseteq 2^{\mathcal{N}}$. If $Y' \subseteq Y$ then $Sub(P, Y') \subseteq Sub(P, Y)$.*

Proof. By induction on the structure of P.
(Case $x?m.P'$) By inductive hypothesis for all $n \in Y'$ we have that $Sub(P'\{n/m\}, Y') \subseteq Sub(P'\{n/m\}, Y)$. As a consequence, we obtain the condition $\bigcup_{n\in Y'} Sub(P'\{n/m\}, Y') \subseteq \bigcup_{n\in Y} Sub(P'\{n/m\}, Y)$ and hence the Lemma holds.

The other cases are similar. □

Lemma 5.2.32. *Let $P, Q \in \mathcal{P}$ and $Y, Y', Y'' \subseteq 2^{\mathcal{N}}$. If $Y' \subseteq Y''$ and $Sub(P, Y') \subseteq Sub(Q, Y'')$ then $Sub(P, Y' \cup Y) \subseteq Sub(Q, Y'' \cup Y)$.*

Proof. By induction on the structure of P.
(Case $x!m.P'$) By definition $Sub(x!m.P', Y') = \{x!m.P'\} \cup Sub(P', Y')$. By hypothesis and Lemma 5.2.31 we have $Sub(x!m.P', Y') \subseteq Sub(Q, Y'') \subseteq Sub(Q, Y''\cup Y)$ and therefore $x!m.P' \in Sub(Q, Y'' \cup Y)$. By definition of Sub function and by inductive hypothesis, we can derive $Sub(P', Y'' \cup Y) \subseteq Sub(Q, Y'' \cup Y)$ and hence the Lemma holds.
(Case $x?m.P'$) By definition $Sub(x?m.P', Y') = \{x?m.P'\} \cup (\bigcup_{n\in Y'} Sub(P'\{n/m\}, Y'))$. By hypothesis and Lemma 5.2.31 $Sub(x!m.P', Y') \subseteq Sub(Q, Y'') \subseteq Sub(Q, Y'' \cup Y)$ and therefore $x!m.P' \in Sub(Q, Y'' \cup Y)$. By definition of Sub function and by inductive hypothesis, we can derive for all $n \in Y''\cup Y$ we have $Sub(P'\{n/m\}, Y''\cup Y) \subseteq Sub(Q, Y''\cup Y)$ and hence the Lemma follows.
(Case $P_0|P_1$) By hypothesis and Sub function definition we have $Sub(P_0, Y') \subseteq Sub(Q, Y'')$ and $Sub(P_1, Y') \subseteq Sub(Q, Y'')$. By inductive hypothesis we have $Sub(P_0, Y' \cup Y) \subseteq Sub(Q, Y'' \cup Y)$ and $Sub(P_1, Y' \cup Y) \subseteq Sub(Q, Y'' \cup Y)$ and therefore $Sub(P_0, Y' \cup Y) \cup Sub(P_1, Y' \cup Y) \subseteq Sub(Q, Y'' \cup Y)$. The Lemma follows.

The other cases are similar. □

The Sub function definition and its properties can be extended to safe bio-processes. In particular, we want to collect all the *possible* sequential and replicated sub-processes of all the boxes internal processes, with associated the corresponding interfaces (*sub-boxes*).

Definition 5.2.33. *Let $B \in \overline{\mathcal{S}}_{safe}$ and $Y \subseteq 2^{\mathcal{N}}$ be a finite set of names. The set of* possible *sub-boxes of B over the set of names Y is defined as:*

$$\begin{aligned} Sub(\mathsf{Nil}, Y) &= \emptyset \\ Sub(I[P], Y) &= \{I[P'] \mid P' \in Sub(P, Y \cup \mathsf{sub}(I))\} \\ Sub(B \mid\mid B', Y) &= Sub(B, Y) \cup Sub(B', Y) \end{aligned}$$

Lemma 5.2.34. *Let $B \in \overline{\mathcal{S}}_{safe}$ and $Y \subseteq 2^{\mathcal{N}}$ be a finite set of names. Then $Sub(B, Y)$ is finite.*

Proof. By induction on the structure of B.
(case Nil) The empty set is finite.
(case $I[P]$) From the fact that $\mathsf{sub}(I)$ is finite and by Lemma 5.2.28, we have that $Sub(P, Y \cup \mathsf{sub}(I))$ is finite and hence the set $\{I[P'] \mid P' \in Sub(P, Y \cup \mathsf{sub}(I))\}$ is finite.
(case $B \parallel B'$) By inductive hypothesis $Sub(B, Y)$ and $Sub(B', Y)$ are finite sets and therefore their union is finite. □

Corollary 5.2.35. *Let $B \in \overline{\mathcal{S}}_{safe}$. Then $Sub(B, \mathsf{fn}(B))$ is finite.*

Proof. From the fact that $\mathsf{fn}(B)$ is a finite set of names and by Lemma 5.2.34. □

Lemma 5.2.36. *Let $B, B' \in \overline{\mathcal{S}}_{safe}$ and $B \equiv_b^{dec} B'$. Then $Sub(B, \mathsf{fn}(B)) = Sub(B', \mathsf{fn}(B'))$.*

Proof. By induction on the length of the derivation $B \equiv_p^{dec} B'$ and by using Lemma 5.2.30. □

Lemma 5.2.37. *Let $B \in \overline{\mathcal{S}}_{safe}$ and $Y, Y' \subseteq 2^{\mathcal{N}}$. If $Y' \subseteq Y$ then $Sub(B, Y') \subseteq Sub(B, Y)$.*

Proof. By induction on the structure of B.
(Case $I[P]$) By Lemma 5.2.31 we have $Sub(P, Y' \cup \mathsf{sub}(I)) \subseteq Sub(P, Y \cup \mathsf{sub}(I))$. This means $\{I[P'] \mid P' \in Sub(P, Y' \cup \mathsf{sub}(I))\} \subseteq \{I[P'] \mid P' \in Sub(P, Y \cup \mathsf{sub}(I))\}$ and the Lemma follows.

The other cases are similar. □

Lemma 5.2.38. *Let $B, B' \in \overline{\mathcal{S}}_{safe}$ and $Y, Y', Y'' \subseteq 2^{\mathcal{N}}$. If $Y' \subseteq Y''$ and $Sub(I[P], Y') \subseteq Sub(I[P'], Y'')$ then $Sub(I[P], Y' \cup Y) \subseteq Sub(I[P'], Y'' \cup Y)$.*

Proof. By hypothesis and Sub definition we have $Sub(P, Y' \cup \mathsf{sub}(I)) \subseteq Sub(P', Y'' \cup \mathsf{sub}(I))$. By Lemma 5.2.32 we have $Sub(P, Y' \cup \mathsf{sub}(I) \cup Y) \subseteq Sub(P', Y'' \cup \mathsf{sub}(I) \cup Y)$. This means $\{I[P''] \mid P'' \in Sub(P, Y' \cup \mathsf{sub}(I) \cup Y)\} \subseteq \{I[P''] \mid P' \in Sub(P', Y'' \cup \mathsf{sub}(I) \cup Y)\}$, which means $Sub(I[P], Y' \cup Y) \subseteq Sub(I[P'], Y'' \cup Y)$. □

To prove that $\preceq_b$ is a wqo we need some preliminary results. The first result states that, given a transition $P \xrightarrow{\theta} P'$ over processes and a transition $B \xrightarrow{\theta} B'$ over bio-processes, the set of possible sequential and replicated processes of P' over $\mathsf{fn}(P')$ and of B' over $\mathsf{fn}(B')$ are delimited by the set of possible sequential and replicated processes of P over $\mathsf{fn}(P)$ and of B over $\mathsf{fn}(B)$ and the names in θ.

Lemma 5.2.39. *Let $P, P' \in \mathcal{P}$ and $Y \in 2^{\mathcal{N}}$. Suppose $P \xrightarrow{\theta} P'$, then*

(a) if $\theta = x!y$ then $Sub(P', \mathsf{fn}(P')) \subseteq Sub(P, \mathsf{fn}(P))$

(b) if $\theta = x?y$ *then* $Sub(P', \mathsf{fn}(P')) \subseteq Sub(P, \mathsf{fn}(P) \cup \{y\})$

(c) if $\theta = \tau$ *then* $Sub(P', \mathsf{fn}(P')) \subseteq Sub(P, \mathsf{fn}(P))$

Proof. For all the three statements the proof is by induction on the inference of $P \xrightarrow{\theta} P'$. Cases (rep_in) and (rep_out) are similar to (pi_in) and (pi_out). The (pi_par) cases are similar for all the three statements and hence we provide the proof only for (a). The (pi_sum) cases are similar to the (pi_par) ones.
(a)
(Case (pi_out)) We have $P = x!y.P' \xrightarrow{x!y} P'$. By definition we have that $Sub(P, \mathsf{fn}(P)) = \{x!y.P'\} \cup Sub(P', \mathsf{fn}(P))$. By Lemma 5.2.10 we know that $\mathsf{fn}(P') \subseteq \mathsf{fn}(P)$ and hence (by Lemma 5.2.31) $Sub(P', \mathsf{fn}(P')) \subseteq Sub(P', \mathsf{fn}(P))$. As a consequence $Sub(P', \mathsf{fn}(P')) \subseteq Sub(P, \mathsf{fn}(P))$.
(Case (pi_par)) By hypothesis we have $P_0 \xrightarrow{x!y} P_0'$ and hence, by inductive hypothesis, we obtain $Sub(P_0', \mathsf{fn}(P_0')) \subseteq Sub(P_0, \mathsf{fn}(P_0))$. By Lemma 5.2.10 we know $\mathsf{fn}(P_0') \subseteq \mathsf{fn}(P_0)$ and hence by Lemma 5.2.32 $Sub(P_0', \mathsf{fn}(P_0') \cup \mathsf{fn}(P_1)) \subseteq Sub(P_0, \mathsf{fn}(P_0) \cup \mathsf{fn}(P_1))$. Moreover, by Lemma 5.2.31 we obtain $Sub(P_1, \mathsf{fn}(P_0') \cup \mathsf{fn}(P_1)) \subseteq Sub(P_1, \mathsf{fn}(P_0) \cup \mathsf{fn}(P_1))$. By definition of the Sub function the Lemma follows.
(b)
(Case (pi_in)) We have $P = x?w.P' \xrightarrow{x?y} P'\{y/w\}$. By definition we have that $Sub(P, \mathsf{fn}(P) \cup \{y\}) = \{x?w.P'\} \cup (\bigcup_{n \in \mathsf{fn}(P) \cup \{y\}} Sub(P'\{n/w\}, \mathsf{fn}(P) \cup \{y\})$ and therefore contains the subset $Sub(P'\{y/w\}, \mathsf{fn}(P) \cup \{y\})$. By Lemma 5.2.10 we know that $\mathsf{fn}(P'\{y/w\}) \subseteq \mathsf{fn}(P) \cup \{y\}$ and hence (by Lemma 5.2.31) $Sub(P'\{y/w\}, \mathsf{fn}(P'\{y/w\})) \subseteq Sub(P'\{y/w\}, \mathsf{fn}(P) \cup \{y\})$. The Lemma follows.
(c)
(Case (l_intra)) By hypothesis we have $P_0 \xrightarrow{x!z} P_0'$ and $P_1 \xrightarrow{x?z} P_1'$. By (a) and (b) we have that $Sub(P_0', \mathsf{fn}(P_0')) \subseteq Sub(P_0, \mathsf{fn}(P_0))$ and $Sub(P_1', \mathsf{fn}(P_1')) \subseteq Sub(P_1, \mathsf{fn}(P_1) \cup \{z\})$. By applying Lemma 5.2.10, Lemma 5.2.31 and Lemma 5.2.32 we obtain $Sub(P_0', \mathsf{fn}(P_0') \cup \mathsf{fn}(P_1')) \subseteq Sub(P_0, \mathsf{fn}(P_0) \cup \mathsf{fn}(P_1))$ and $Sub(P_1', \mathsf{fn}(P_0') \cup \mathsf{fn}(P_1')) \subseteq Sub(P_1, \mathsf{fn}(P_0) \cup \mathsf{fn}(P_1))$ and, by Sub function definition, the Lemma follows. □

Lemma 5.2.40. *Let* $B, B' \in \overline{\mathcal{S}}_{safe}$. *Suppose* $B \xrightarrow{\theta} B'$. *Then*

(a) if $\theta = \Delta x!y$ *then* $Sub(B', \mathsf{fn}(B')) \subseteq Sub(B, \mathsf{fn}(B))$

(b) if $\theta = \Delta x?y$ *then* $Sub(B', \mathsf{fn}(B')) \subseteq Sub(B, \mathsf{fn}(B) \cup \{y\})$

(c) if $\theta = \tau$ *then* $Sub(B', \mathsf{fn}(B')) \subseteq Sub(B, \mathsf{fn}(B))$

Proof. For all the three statements the proof is by cases on the derivation of $B \xrightarrow{\theta} B'$. We present only the most important cases. The other cases can be proved similarly.

(a)
(Case (bio_out)) By hypothesis we know that $P \xrightarrow{x!y} P'$ and therefore by Lemma 5.2.39 we have $Sub(P', \mathsf{fn}(P')) \subseteq Sub(P, \mathsf{fn}(P))$. By applying Lemma 5.2.32 we have $Sub(P', \mathsf{fn}(P') \cup \mathsf{sub}(I)) \subseteq Sub(P, \mathsf{fn}(P) \cup \mathsf{sub}(I))$. But this means that $\{I[P''] | P'' \in Sub(P', \mathsf{fn}(P') \cup \mathsf{sub}(I))\} \subseteq \{I[P''] | P'' \in Sub(P, \mathsf{fn}(P) \cup \mathsf{sub}(I))\}$ and hence $\{I[P''] | P'' \in Sub(P', (\mathsf{fn}(P') \setminus \mathsf{sub}(I)) \cup \mathsf{sub}(I))\} \subseteq \{I[P''] | P'' \in Sub(P, (\mathsf{fn}(P) \setminus \mathsf{sub}(I)) \cup \mathsf{sub}(I))\}$. Thus, we have that $Sub(I[P'], \mathsf{fn}(P') \setminus \mathsf{sub}(I)) \subseteq Sub(I[P], \mathsf{fn}(P) \setminus \mathsf{sub}(I))$, that leads to $Sub(I[P'], \mathsf{fn}(I[P'])) \subseteq Sub(I[P], \mathsf{fn}(I[P]))$.
(Case (l_bio_par)) By hypothesis we have that $B_0 \xrightarrow{\Delta x?y} B_0'$. Similarly to Lemma 4.4.2, also here there exists a normalized derivation such that $B_0 \equiv_b^{dec} I[P] \parallel B$ and $B_0' \equiv_b^{dec} I[P'] \parallel B$ with $I[P] \xrightarrow{\Delta x?y} I[P']$. From the previous case we know that $Sub(I[P'], \mathsf{fn}(I[P'])) \subseteq Sub(I[P], \mathsf{fn}(I[P]))$. Moreover, from Lemma 5.2.10 we know $\mathsf{fn}(I[P']) \subseteq \mathsf{fn}(I[P])$ and hence, by Lemma 5.2.32, $Sub(I[P'], \mathsf{fn}(I[P']) \cup \mathsf{fn}(B) \cup \mathsf{fn}(B_1)) \subseteq Sub(I[P], \mathsf{fn}(I[P]) \cup \mathsf{fn}(B) \cup \mathsf{fn}(B_1))$. Now, knowing that $\mathsf{fn}(B_0') \cup \mathsf{fn}(B_1) \subseteq \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1)$, we know, by Lemma 5.2.31, that $Sub(B, \mathsf{fn}(B_0') \cup \mathsf{fn}(B_1)) \subseteq Sub(B, \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1))$ and also that $Sub(B_1, \mathsf{fn}(B_0') \cup \mathsf{fn}(B_1)) \subseteq Sub(B_1, \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1))$. But this means

$$Sub(I[P'] \parallel B \parallel B_1, \mathsf{fn}(B_0') \cup \mathsf{fn}(B_1)) \subseteq Sub(I[P] \parallel B \parallel B_1, \mathsf{fn}(B_0) \cup \mathsf{fn}(B_1))$$

and by Lemma 5.2.36 we have $Sub(B_0' \parallel B_1, \mathsf{fn}(B_0' \parallel B_1)) \subseteq Sub(B_0 \parallel B_1, \mathsf{fn}(B_0 \parallel B_1))$.
(b)
(Case (bio_in)) By hypothesis we know that $P \xrightarrow{x?y} P'$ and therefore by Lemma 5.2.39 we have $Sub(P', \mathsf{fn}(P')) \subseteq Sub(P, \mathsf{fn}(P) \cup \{y\})$. By applying Lemma 5.2.32 we obtain $Sub(P', \mathsf{fn}(P') \cup \mathsf{sub}(I)) \subseteq Sub(P, \mathsf{fn}(P) \cup \mathsf{sub}(I) \cup \{y\})$. But this means that $\{I[P''] | P'' \in Sub(P', \mathsf{fn}(P') \cup \mathsf{sub}(I))\} \subseteq \{I[P''] | P'' \in Sub(P, \mathsf{fn}(P) \cup \mathsf{sub}(I) \cup \{y\})\}$ and, similarly to the previous case (bio_out), the Lemma follows. □

Now, we define a superset of the set of derivatives of a bio-process B, denoted with $\mathcal{P}_B$. This set includes all the bio-processes whose possible sequential and replicated sub-processes are contained in the corresponding elements of B.

Definition 5.2.41. *Let $B \in \overline{\mathcal{S}}_{safe}$. Then*

$$\mathcal{P}_B = \{B' \in \overline{\mathcal{S}}_{\text{safe}} \mid Sub(B', \mathsf{fn}(B')) \subseteq Sub(B, \mathsf{fn}(B)) \wedge |Boxes(B')| = |Boxes(B)|\}$$

The following result describes how given a bio-process B and a bio-process $B' \in \mathcal{P}_B$, all the derivatives of B' are contained in $\mathcal{P}_B$.

Lemma 5.2.42. *Let $B \in \overline{\mathcal{S}}_{safe}$ and $B' \in \mathcal{P}_B$. If $B' \xrightarrow{\tau} B''$ then $B'' \in \mathcal{P}_B$.*

Proof. $B' \in \mathcal{P}_B$ implies $Sub(B', \mathsf{fn}(B')) \subseteq Sub(B, \mathsf{fn}(B))$. Since by hypothesis $B' \xrightarrow{\tau} B''$, by Lemma 5.2.40 we have $Sub(B'', \mathsf{fn}(B'')) \subseteq Sub(B', \mathsf{fn}(B'))$. Transitivity of $\subseteq$ and $|Boxes(B')| = |Boxes(B'')|$ (immediate by semantics definition) prove $Sub(B'', \mathsf{fn}(B'')) \subseteq Sub(B, \mathsf{fn}(B))$, which means $B'' \in \mathcal{P}_B$. □

A consequence of this lemma is that all the derivatives of a bio-process B are contained in $\mathcal{P}_B$.

Corollary 5.2.43. *Let $B \in \overline{\mathcal{S}}_{safe}$. Then $Deriv(B) \subseteq \mathcal{P}_B$.*

Proof. Immediate from Lemma 5.2.42. □

The following Lemma shows how a bio-process in $\mathcal{P}_B$ can be rewritten (up-to $\equiv_b^{dec}$) as a parallel composition of boxes that are in relation with boxes in $Sub(B, \mathsf{fn}(B))$.

Lemma 5.2.44. *Let $B \in \overline{\mathcal{S}}_{safe}$ and $B' \in \mathcal{P}_B$. Then we have that*

$$B' \equiv_b^{dec} \prod_{i=1}^{n} I_i[\prod_{j=1}^{m} Q_{i,j}]$$

with $I_i[Q_{i,j}] \in Sub(B, \mathsf{fn}(B))$ for $i = 1, ..., n$ and $j = 1, \ldots, m$.

Proof. Immediate from the bio-processes syntax and from the definition of Sub function. □

In the following Lemma we define the relation $=_*$ over bio-processes according to Def. 5.2.5.

Lemma 5.2.45. *Let $B \in \overline{\mathcal{S}}_{safe}$ and $I[P_1], ..., I[P_n], I'[Q_1], ..., I'[Q_m]$ in $Sub(B, \mathsf{fn}(B))$. If $I[P_1], ..., I[P_n] =_* I'[Q_1], ..., I'[Q_m]$ then $I[\prod_{i=1}^{n} P_i] \preceq_b I'[\prod_{i=1}^{m} Q_i]$.*

Proof. If $I[P_1], ..., I[P_n] =_* I'[Q_1], ..., I'[Q_m]$ then there exists an injection $f : \{1, ..., n\} \to \{1, ..., m\}$ such that $I[P_i] = I'[Q_{f(i)}]$ and $i = f(i)$, with $i = 1, ..., n$. The injection f corresponds to the identity function of the set $\{1, ..., n\}$ and hence for each i in $\{1, ..., n\}$ we have $I[P_i] = I'[Q_i]$. By $\equiv_b^{dec}$ definition we can write $I'[\prod_{i=1}^{m} Q_i] \equiv_b^{dec} I'[\prod_{i=1}^{n} Q_i \mid \prod_{i=n+1}^{m} Q_i]$ and by the equality result we have $I'[\prod_{i=1}^{m} Q_i] \equiv_b^{dec} I'[\prod_{i=1}^{n} P_i \mid \prod_{i=n+1}^{m} Q_i]$ (notice that $I = I'$), which means $I[\prod_{i=1}^{n} P_i] \preceq_b I'[\prod_{i=1}^{m} Q_i]$. □

The following theorem shows that the qo $\preceq_b$ is a wqo.

Theorem 5.2.46. *Let $B \in \overline{\mathcal{S}}_{safe}$. The relation $\preceq_b$ is a wqo over $\mathcal{P}_B$.*

Proof. We take an infinite sequence $B_1, ..., B_j, ...$ such that $B_i \in \mathcal{P}_B$ for $i > 0$. By Lemma 5.2.44, for any i we have that:

$$B_i \equiv_b^{dec} \prod_{j=1}^{n} I_{i,j}[\prod_{k=1}^{m_{i,j}} P_{i,j,k}]$$

Hence, each B_i can be seen as composed of n finite sequences:

$$\begin{array}{l} I_{i,1}[P_{i,1,1}], \ldots, I_{i,1}[P_{i,1,m_{i,1}}] \\ I_{i,2}[P_{i,2,1}], \ldots, I_{i,2}[P_{i,2,m_{i,2}}] \\ \ldots \\ I_{i,n}[P_{i,n,1}], \ldots, I_{i,n}[P_{i,n,m_{i,n}}] \end{array}$$

Note that all the sequences are composed of elements from the finite set $Sub(B, \mathsf{fn}(B))$. Each sequence is hence an element of $Sub(B, \mathsf{fn}(B))^*$ and hence we have n infinite sequences of elements in $Sub(B, \mathsf{fn}(B))^*$. By Corollary 5.2.35 $Sub(B, \mathsf{fn}(B))$ is finite, and by applying Lemma 5.2.7 and Higman's Theorem 5.2.6 we have that $=_*$ is a wqo over $Sub(B, \mathsf{fn}(B))^*$.

Now, we can extract an infinite subsequence from $B_1, ..., B_i, ...$ making the finite sequences $I_{i,1}[P_{i,1,1}], \ldots, I_{i,1}[P_{i,1,m_{i,1}}]$ increasing with respect to $=_*$; then, we continue by extracting an infinite subsequence from the subsequence obtained previously, making the finite sequences $I_{i,2}[P_{i,2,1}], \ldots, I_{i,2}[P_{i,2,m_{i,2}}]$ increasing also in this case with respect to $=_*$. We continue for all the n subsequences.

We end up with an infinite subsequence $B_{n_0}, ..., B_{n_i}, ...$ (with $n_0 < \cdots < n_i < \ldots$) of $B_1, ..., B_i, ...$ such that all the n finite sequences are ordered with respect to $=_*$. By Lemma 5.2.45 we obtain:

$$I_{n_0,j}[\prod_{l=1}^{m_{n_0,j}} P_{n_0,j,l}] \preceq_b \cdots \preceq_b I_{n_i,j}[\prod_{l=1}^{m_{n_i,j}} P_{n_i,j,l}] \preceq_b \cdots \quad \text{for } j = 1, \ldots, n$$

from which we finally obtain $B_{n_0} \preceq_b \cdots \preceq_b B_{n_i} \preceq_b \cdots$ □

The hypothesis of Theorem 5.2.4 are satisfied by the following theorem.

Theorem 5.2.47. *Let $B \in \overline{\mathcal{S}}_{safe}$. The transition system $(Deriv(B), \xrightarrow{\tau}, \preceq_b)$ is a well-structured transition system with decidable $\preceq_b$ and computable Succ.*

Proof. The relation $\preceq_b$ has been proved strong compatible in Theorem 5.2.25. Moreover, the fact that $\preceq_b$ is a wqo on $Deriv(B)$ is a consequence of Corollary 5.2.43 and Theorem 5.2.46.

Given $B, B' \in Deriv(B)$, deciding whether $B \preceq_b B'$ means to find a bijection between boxes of B and B' such that, for each correspondence $(I[P], I[P'])$, a subterm P'' of P' with $P' \equiv_p^{dec} P \mid P''$ exists. This is a decidable problem. □

Corollary 5.2.48. *Let $B \in \overline{\mathcal{S}}_{safe}$. The termination of B is decidable.*

5.3 Undecidability results

In this section we prove that termination is undecidable for B^p_{core} and B^e_{core}. We show this by providing encodings of Random Access Machines (RAMs)[101], a well known Turing-complete formalism, into B^p_{core} and B^e_{core}. First of all we recall the definition of RAMs.

5.3.1 Random access machines

A Random Access Machine (RAM) is an abstract machine in the general class of register machines. RAMs are a computational model based on finite programs acting on a finite set of registers.

A RAM R is composed of a finite set of registers $r_1, \cdots, r_n$ and a sequence of indexed instructions $(1, I_1), \cdots, (m, I_m)$. Registers store natural numbers, one for each register, and can be updated (incremented or decremented) and tested for zero. In [75] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Incr(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;
- $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the instruction indexed with the number 1 and it continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached.

The state of a RAM R is a tuple $(j, k_1, ..., k_n)$ where j is the index of next instruction to be executed and $k_1, ..., k_n$ are the current contents of the registers. The execution is defined by a transition relation among states

$$(j, k_1, ..., k_n) \rightarrow_R (j', k'_1, ..., k'_n)$$

meaning that the state of the RAM changes from $(j, k_1, ..., k_n)$ to $(j', k'_1, ..., k'_n)$, as a consequence of the execution of the j-th instruction.

A state $(j, k_1, ..., k_n)$ is terminated if the program counter j is greater than the number of instructions m. We say that a RAM R terminates if its computation reaches a terminated state.

5.3.2 Encoding with priorities

In this section we enrich B_{core} by adding priorities and we show that the resulting language turns out to be Turing equivalent by providing an encoding of RAMs into it. We call this subset B^p_{core}. Priorities are a frequently used feature of many computational systems and many process calculi have been enriched with some priority mechanism [9, 19, 87]. How priorities affect the expressive power of a language has been previously studied in [87, 108]. As for the full BlenX we use a mechanism based on global priorities [5, 19] where we assume the existence of a function $\delta : \mathcal{N} \to \{0, 1, 2\}$ that associates priorities to communication channels and by limiting the compatibility function to $\alpha : \mathcal{T}^2 \to \{0, 1, 2\}$ that associates priorities to sort pairs. Value 0 means that no communication is enabled, while 1 and 2 represent priority levels. We concentrate on a two-level priority mechanism, showing that it is enough to reach Turing completeness.

The key ingredient in this encoding is the combined use of choice and priorities; boxes and sorts are used only to maintain a certain homogeneity and uniformity with respect to this encoding and the one presented in the next section. Very recently, in [3] the authors show an encoding of RAMs into a subset of CCS with replication and enriched with priorities which is similar in the spirit to the one presented here.

Well-formedness conditions and structural congruence relations $\equiv_p$ and $\equiv_b$ remain unchanged with respect to Sec. 5.1 and the reduction semantics for B^p_{core} is reported in Tab. 5.4. Moreover, we denote with $\overline{\mathcal{S}}_p$ the set of well-formed systems of B^p_{core}. Tab. 5.4 shows that intra-communications and inter-communications of priority 1 can be derived only if no intra-communications and inter-communications with priority 2 are enabled.

It is important to note that the introduction of priorities causes the generation of transition systems which are not well-structured. Indeed, since priorities remove certain possibilities that would have existed without priorities, we are no longer able to define a quasi-ordering over bio-processes (of the kind presented in Sec.5.2) that satisfies the *strong compatibility* property.

Definition 5.3.1. *A B^p_{core} transition system is referred to as $(\overline{\mathcal{S}}_p, \to)$, where $\overline{\mathcal{S}}_p$ is the set of well-formed systems and $\to \subseteq \overline{\mathcal{S}}_p \times \overline{\mathcal{S}}_p$ is the transition relation, constructed using the transition relation $\xrightarrow{p} \subseteq \overline{\mathcal{S}}_p \times \overline{\mathcal{S}}_p$ (see Tab.5.4).*

Consider a RAM R with program $(1, I_1), ..., (m, I_M)$ and state $(j, k_1, ..., k_n)$. The encoding of the RAM is the bio-process B in Tab.5.5 where we assume $\delta(x) = 1$ for all

(intra) $I[\,x!z.\,P_1 + M_1 \,|\, x?w.\,P_2 + M_2 \,|\, P_3\,] \xrightarrow{\delta(x)} I[\,P_1 \,|\, P_2\{z/w\} \,|\, P_3\,]$
provided $\delta(x) > 0$

(inter) $$\frac{P_1 \equiv_p x!z.\,R_1 + M_1 \,|\, Q_1 \quad P_2 \equiv_p y?w.\,R_2 + M_2 \,|\, Q_2}{I_1[\,P_1\,] \parallel I_2[\,P_2\,] \xrightarrow{\alpha(\Gamma,\Delta)} I_1[\,R_1 \,|\, Q_1\,] \parallel I_2[\,R_2\{z/w\} \,|\, Q_2\,]}$$
where $I_1 = \oplus(x,\,\Delta)\,I_1^*$ and $I_2 = \oplus(y,\,\Gamma)\,I_2^*$
and provided $\alpha(\Gamma,\Delta) > 0$ and $z \notin \mathsf{sub}(I_1) \cup \mathsf{sub}(I_2)$

(struct) $$\frac{B_1 \equiv_b B_1' \quad B_1' \xrightarrow{p} B_2' \quad B_2' \equiv_b B_2}{B_1 \xrightarrow{p} B_2}$$

(redex) $$\frac{B \xrightarrow{p} B'}{B \parallel B_1 \xrightarrow{p} B' \parallel B_1}$$

(priority) $$\frac{B \xrightarrow{p} B' \wedge \nexists B_1 \text{ s.t. } B \xrightarrow{p_1} B_1 \text{ with } p_1 > p}{B \to B}$$

Table 5.4: Reduction semantics of B^p_{core}.

$x \in \mathcal{N}$ and:

$$\alpha(\Delta_1, \Delta_2) = \begin{cases} 2 & \text{if } \Delta_1 = \Delta_2 = Test_j^{yes} \text{ for all } j \\ 1 & \text{if } \Delta_1 = \Delta_2 \neq Test_j^{yes} \\ 0 & \text{otherwise} \end{cases}$$

The encoding produces a system in B^p_{core}. This encoding is a parallel composition of a switching box, which controls the activation of the instructions sequence, m boxes encoding instructions and n boxes encoding registers. The two sorts of instructions are encoded in different ways, but in both cases the encoding box is activated by performing an inter-communication on the channel *act* with the switching box $Switch_j$.

Each register r_j is modelled with a box whose internal process structure depends on the content of the register. A register can be incremented and tested for not zero value. The number of parallel unguarded $t_y!e.\,\mathsf{nil}$ processes present in the internal structure of the box represents the content of the register.

The box encoding the instruction $(i, Incr(r_j))$, after its activation, consumes an inter-communication with the box encoding the register r_j (through the interfaces of sorts $IReg_j$), representing a request for its increment. In the register box, these communications produces the replication of the process $t_y!e.\,\mathsf{nil}$, representing the increment of one, while in the instruction box, the inter-communication produces the replication of the internal machinery and the consumption of an inter-communication with the switching box (through the interfaces of sorts Ins) for the activation of instruction $i + 1$.

$$
\begin{array}{rcl}
[\![(j,k_1,...,k_n)]\!]^p_R & = & Switch_j \parallel [\![(1,I_1)]\!]^p_R \parallel \cdots \parallel [\![(m,I_m)]\!]^p_R \parallel \\
 & & [\![r_1 = k_1]\!]^p_R \parallel \cdots \parallel [\![r_n = k_n]\!]^p_R \\
 & & \\
Switch_j & \triangleq_B & \oplus(ins, Ins) \oplus(i_1, Ins_1) \oplus(i_2, Ins_2) \cdots \oplus(i_m, Ins_m) \\
 & & [\ *\, x?e.\, Switch \mid Switch \mid i_j!e.\mathsf{nil}\ ] \\
Switch & \triangleq_P & ins?type.\, (type!e.\, \mathsf{nil} \mid (\sum_{l=1}^{m} ins_l?e.\, x!e.\, i_l!e.\, \mathsf{nil})) \\
 & & \\
[\![(i, Incr(r_j))]\!]^p_R & = & \oplus(act, Ins_i) \oplus(next, Ins) \oplus(inc, IReg_j) \\
 & & [\ *\, x?e.\, Inc_i \mid Inc_i\ ] \\
Inc_i & \triangleq_P & act?e.\, inc!e.\, x!e.\, next!ins_{i+1}.\, \mathsf{nil} \\
 & & \\
[\![(i, DecJump(r_j, s))]\!]^p_R & = & \oplus(act, Ins_i) \oplus(next, Ins) \oplus(t_y, Test_j^{yes}) \\
 & & \oplus(t_n, Test_j^{no})[\ *\, x?e.\, DecJump_i \mid DecJump_i\ ] \\
DecJump_i & \triangleq_P & act?e.\, (t_y?e.\, Dec_i + t_n?e.\, Jump_i) \\
Dec_i & \triangleq_P & x!e.\, next!ins_{i+1}.\, \mathsf{nil} \\
Jump_i & \triangleq_P & x!e.\, next!ins_s.\, \mathsf{nil} \\
 & & \\
[\![r_j = l]\!]^p_R & = & \oplus(t_y, Test_j^{yes}) \oplus(t_n, Test_j^{no}) \oplus(inc, IReg_j) \\
 & & [\ *\, inc?e.\, t_y!e.\, nil \mid *\, t_n!e.\, nil \mid \underbrace{t_y!e.\, nil \mid \cdots \mid t_y!e.\, \mathsf{nil}}_{l}\ ]
\end{array}
$$

Table 5.5: Encoding of RAMs with B^p_{core}.

The box encoding the instruction $(i, DecJump(r_j, s))$, after its activation, presents an alternative behaviour (encoded with the choice operator), which implements the mechanism used for testing the content of the register r_j. In particular, the content of the register is tested with two alternative inter-communications on channels t_y and t_n through interfaces of sorts $Test_j^{yes}$ and $Test_j^{no}$, respectively. In the register box, outputs on channel t_y, if present, generate inter-communications with priority 2.

If the encoded register contains a value $n > 0$, then n parallel compositions of process $t_y!e.\,\mathsf{nil}$ are present and hence inter-communications on output $t_y!e$ have always precedence with respect to the inter-communication that the process $*t_n!e.\,\mathsf{nil}$ offers. In the register box, the consumption of an immediate inter-communication deletes an instance of $t_y!e.\,\mathsf{nil}$ process in its internal structure, representing the decrement of one, while in the instruction box, the consumption of an immediate inter-communication enables the process Dec_i, which replicates its internal box machinery and performs an inter-communication with

the switching box (through the interfaces of sorts Ins) for the activation of instruction $i+1$.

If the encoded register contains the value 0, then no unguarded $t_y!e.\,\mathsf{nil}$ processes are present in the internal structure of the register box and hence the inter-communication that the process $*t_n!e.\,\mathsf{nil}$ offers can be consumed. This causes, in the instruction box, the activation of the process $Jump_i$, which replicates its internal box machinery and performs an inter-communication with the switching box for the activation of instruction s.

A formal proof of the encoding correctness follows.

Lemma 5.3.2. *Let R be a RAM with program $(1, I_1), ..., (m, I_m)$ and state $(j, k_1, ..., k_n)$.*

If $(j, k_1, ..., k_n) \rightarrow_R (j', k'_1, ..., k'_n)$, then there exists a well-formed system $S \in \overline{\mathcal{S}}_p$ such that $[\![(j, k_1, ..., k_n)]\!]^p_R \rightarrow^+ S$ and $S \equiv_b [\![(j', k'_1, ..., k'_n)]\!]^p_R$.

Proof. The proof is by case analysis. There are three different cases: (i) Instruction $I_j = DecJump(r_l, s)$ with r_l content greater than zero; (ii) Instruction $I_j = DecJump(r_l, s)$ with r_l content equal to zero; (iii) Instruction $I_j = Incr(r_l)$. We only prove case (i), because the other cases can be proved similarly.

(i) We consider the computation of $[\![(j, k_1, ..., k_n)]\!]^p_R$. An inter-communication between the component $Switch_j$ and the component $[\![(j, DecJump(r_l, s))]\!]^p_R$ is consumed. In particular, the two boxes synchronize on output $i_j!e$ and input $act?e$ through their interfaces of sorts Ins_j. This causes the activation of the $[\![(j, DecJump(r_l, s))]\!]^p_R$ component. Note that, after the inter-communication, the components codifying for the other instructions, the registers and the switching box are blocked.

The activation of the $[\![(j, DecJump(r_l, s))]\!]^p_R$ box causes the enabling of a choice process. This process is used for testing the content of the register r_j, that is a choice composition of two processes blocked on inputs $t_y?e$ and $t_n?e$, bound to the interfaces with sorts $Test^{yes}_j$ and $Test^{no}_j$, respectively. By hypothesis, the content of the register r_l is greater than 0 and hence the internal process structure of the box $[\![r_l = k_l]\!]^p_R$ is a parallel composition of processes that contain at least one $t_y!e.\,\mathsf{nil}$ processes. Two types of inter-communications between the boxes encoding the instruction and the register are enabled. One inter-communication through interfaces with sorts $Test^{no}_j$ and k_j inter-communications through interfaces with sorts $Test^{yes}_j$. However, since the inter-communications through interfaces with sorts $Test^{yes}_j$ have priority 2, they have precedence with respect to the one through interfaces with sorts $Test^{no}$. The consumption of one of the immediate inter-communications deletes one of the k_l $t_y!e.\,\mathsf{nil}$ processes in $[\![r_l = k_l]\!]^p_R$, resulting (for all the possible inter-communications of this type) in a bio-process structurally congruent to $[\![r_l = k_l - 1]\!]^{gp}_R$, and activates process Dec_j in the instruction box.

At this point, an intra-communication in the instruction box on channel x replicates the internal machinery of the box and enables the process $next!ins_{j+1}.\,\mathsf{nil}$. This produces a synchronization between the instruction box and the switching box, which generates an inter-communication on output $next!ins_{j+1}$ and input $ins?type$ through interfaces of type Ins.

The instruction box is now returned in its form $[\![(j, DecJump(r_l, s))]\!]^p_R$, while in the switching box the process

$$(ins_{j+1}!e.\,\mathsf{nil} \mid (\textstyle\sum_{o=1}^{m} ins_o?e.\,x!e.\,i_o!e.\,\mathsf{nil}))$$

is enabled. An intra-communication on channel ins_{j+1} is consumed, the internal machinery is replicated with an intra-communication on channel x and the switching box is now structurally congruent to the box $Switch_{j+1}$. □

Lemma 5.3.3. *Let R be a RAM with program $(1, I_1), ..., (m, I_m)$ and state $(j, k_1, ..., k_n)$. If the system $S = [\![(j, k_1, ..., k_n)]\!]^p_R$ can produce a transition $S \to S_1$, then there exists a computation $S \to S_1 \to S_2 \to \cdots \to S_l$ such that $S_l = [\![(j', k'_1, ..., k'_n)]\!]^p_R$ and $(j, k_1, ..., k_n) \to_R (j', k'_1, ..., k'_n)$*

Proof. Consider the structure of the system $S = [\![(j, k_1, ..., k_n)]\!]^p_R$. If the system S can perform a first step $S \to S_1$, this corresponds to an inter-communication between the box $Switch_j$ and the box encoding for the instruction (j, I_j), representing the activation of the instruction box. The encoding definition ensures that the instruction (j, I_j) exists; hence the instruction can be executed in the state $(j, k_1, ..., k_n)$ of the RAM R, generating a new state $(j', k'_1, ..., k'_n)$.

There are three cases: (i) instruction $I_j = DecJump(r_l, s)$ with r_l content greater than zero; (ii) instruction $I_j = DecJump(r_l, s)$ with r_l content equal to zero; (iii) instruction $I_j = Incr(r_l)$. In all the cases, it is possible to show that from the moment in which the switch activates an instruction till the moment in which the switch is able to activate a new instruction, the computation proceeds deterministically (up-to structural congruence $\equiv_b$). The encoding is hence deterministic up-to structural congruence. We prove only case (iii), because all the other cases can be proved similarly.

(iii) By encoding definition we have that the structure of the instruction box is

$$\begin{aligned}[\![(j, Incr(r_l))]\!]^p_R \;=\; & \oplus(act,\, Ins_j) \oplus(next,\, Ins) \oplus(inc,\, IReg_l) \\ & [\; *x?e.\,Inc_j \mid act?e.\,inc!e.\,x!e.\,next!ins_{j+1}.\,\mathsf{nil} \;]\end{aligned}$$

This box is the only one able to synchronize with the box $Switch_j$ for an inter-communication through interfaces of sorts Ins_j. After the inter-communication we have

that in S_1 the box encoding the instruction j becomes structurally congruent to

$$\begin{array}{rcl} B' & = & \oplus(act,\, Ins_j) \oplus(next,\, Ins) \oplus(inc,\, IReg_l) \\ & & [\ *x?e.\, Inc_j \mid inc!e.\, x!e.\, next!ins_{j+1}.\, \mathsf{nil}\ ] \end{array}$$

At this point, the only possible action $S_1 \to S_2$ is the inter-communication between the box B' and the box $[\![r_l = k]\!]^p_R$ through their interfaces of sorts $IReg_l$ on output $inc!e$ and input $inc?e$, respectively. After the inter-communication the box encoding for the register r_l becomes structurally congruent to

$$\begin{array}{c} \oplus(t_y,\, Test_l^{yes}) \oplus(t_n,\, Test_l^{no}) \oplus(inc,\, IReg_l) \\ [\ *\, inc?e.\, t_y!e.\, \mathsf{nil} \mid\ *\, t_n!e.\, \mathsf{nil} \mid Val_k \mid t_y!e.\, \mathsf{nil}\ ] \end{array}$$

which corresponds to the box $[\![r_l = k+1]\!]^p_R$. Instead the box encoding the instruction j becomes structurally congruent to

$$\begin{array}{rcl} B'' & = & \oplus(act,\, Ins_j) \oplus(next,\, Ins) \oplus(inc,\, IReg_l) \\ & & [\ *\, x?e.\, Inc_j \mid x!e.\, next!ins_{j+1}.\, \mathsf{nil}\ ] \end{array}$$

Now, the action $S_2 \to S_3$ is the intra-communication of B'' on channel x which becomes a box B''' with internal structure $*x?e.\, Inc_j \mid next!ins_{j+1}.\, \mathsf{nil}$, and the action $S_3 \to S_4$ is the inter-communication between the box B''' and the switching box. After the inter-communication, the instruction box returns in its initial form $[\![(j, Incr(r_l))]\!]^p_R$ and the switching box starts a sequence of intra-communications which produces a box structurally congruent to $Switch_{j+1}$ and representing the sequence of actions $S_4 \to S_5 \to S_6$. It is easy to see that S_6 is structurally congruent to $[\![(j+1, k_1, ..., k_{l-1}, k+1, k_{l+1}, ..., k_n)]\!]^p_R$. □

Lemma 5.3.2 and Lemma 5.3.3 give us the instruments for proving the undecidability of termination for B^p_{core} systems.

Theorem 5.3.4. *Let R be a RAM with a program $(1, I_1), ..., (m, I_m)$ and an initial state $(j, k_1, ..., k_n)$. Then the computation of the RAM R terminates if and only if the computation of the system $S = [\![(j, k_1, ..., k_n)]\!]^p_R$ terminates.*

Proof. ($\Rightarrow$) By hypothesis we have that the RAM R terminates. This means that the computation of R reaches, in a number l of steps, a terminated state $(j', k'_1, ..., k'_n)$, i.e., a state with a program counter greater than the number of instructions. The proof is by contradiction assuming that the system S does not terminate, which means we have an infinite computation $S \to S_1 \to \cdots \to S_i \to \cdots$. By Lemma 5.3.2 we have that there exists a well-formed system $S' \in \overline{\mathcal{S}}_p$ such that $S \to^* S'$ and $S' \equiv_b [\![(j', k'_1, ..., k'_n)]\!]^p_R$. By assumption, S does not terminate and hence there exists S'' such that $S' \to S''$. By

Lemma 5.3.3 we have that there exists a computation $S' \to S'' \to S_2 \to \cdots \to S_l$ such that $S_l = [\![(j'', k_1'', ..., k_n'')]\!]_R^p$ and $(j', k_1', ..., k_n') \to_R (j'', k_1'', ..., k_n'')$. But this contradicts our hypothesis, which states that $(j', k_1', ..., k_n')$ is a terminated state and therefore the implication holds.
($\Leftarrow$) By hypothesis we have that the system S terminates. This means that there exists a computation $S \to^l S'$ such that $S' \not\to$. The proof is by contradiction assuming that the RAM R does not terminate. By applying Lemma 5.3.3 we have that $S' \equiv_b [\![(j', k_1', ..., k_n')]\!]_R^p$ and that $(j, k_1, ..., k_n) \to_R \cdots \to_R (j', k_1', ..., k_n')$. By assumption we have that $(j', k_1', ..., k_n') \to_R (j'', k_1'', ..., k_n'')$ and hence by Lemma 5.3.2 we have that there exists a well-formed system $S'' \in \overline{\mathcal{S}}_p$ such that $S' \to^+ S''$ and $S'' \equiv_b [\![(j'', k_1'', ..., k_m'')]\!]_R^p$. But this contradicts our hypothesis, which states that $S' \not\to$ and therefore the implication holds. □

5.3.3 Encoding with events

In this section, we extend the B_{core} language by adding events. We enrich the syntax of B_{core} by adding the syntactic category of events:

$$\begin{array}{rcl} E & ::= & B \rhd B \\ & | & E \parallel E \end{array}$$

Note that with respect to to the full BlenX we limit the use of events by allowing only the specification of events that do not involve complexes or sub-complexes. In particular we will show in this section that events of the type:

$$I[P] \rhd I'[P'] \parallel I''[P''] \quad \text{and} \quad I[P] \parallel I'[P'] \rhd I''[P'']$$

are enough to recover Turing completeness.

A system becomes therefore a pair (B, E), where B is a bio-process and E is a parallel composition of events. We denote with B_{core}^e this extension and with $\overline{\mathcal{S}}_e$ the set of well-formed systems of B_{core}^e. Well-formedness condition over systems extends the one presented in Sec. 5.1 by simply stating that a system (B, E) is well-formed if B is well-formed and all the bio-processes present in the events are well-formed. Note that also in this case this is a simplification of the well-formedness condition of BlenX and that a formal proof system can be constructed easily starting from Tab. 4.7.

The structural congruence and reduction semantics of B_{core}^e (Tab. 5.6) extend properly the ones of B_{core}. Note that both are obtained by adding rules for events that are similar to the ones of full BlenX.

(intra) $(I[\, x!z.\, P_1 + M_1 \,|\, x?w.\, P_2 + M_2 \,|\, P_3\,], E) \to (I[\, P_1 \,|\, P_2\{z/w\} \,|\, P_3\,], E)$

(inter) $$\frac{P_1 \equiv_p x!z.\, R_1 + M_1 \,|\, Q_1 \quad P_2 \equiv_p y?w.\, R_2 + M_2 \,|\, Q_2}{(I_1[\, P_1\,] \parallel I_2[\, P_2\,], E) \to (I_1[\, R_1 \,|\, Q_1\,] \parallel I_2[\, R_2\{z/w\} \,|\, Q_2\,], E)}$$
where $I_1 = \oplus(x,\, \Delta)\, I_1^*$ and $I_2 = \oplus(y,\, \Gamma)\, I_2^*$
and provided $\alpha(\Gamma, \Delta) > 0$ and $z \notin \mathsf{sub}(I_1) \cup \mathsf{sub}(I_2)$

(event) $(B, B \rhd B' \parallel E) \to (B', B \rhd B' \parallel E)$

(redex) $$\frac{(B, E) \to (B', E)}{(B \parallel B_1, E) \to (B' \parallel B_1, E)}$$

(struct) $$\frac{S_1 \equiv_b S_1' \quad S_1' \to S_2' \quad S_2' \equiv_b S_2}{S_1 \to S_2}$$

...

c) axioms for events

1. $E \parallel \mathsf{Nil} \equiv_e E$
2. $E_1 \parallel E_2 \equiv_e E_2 \parallel E_1$
3. $E_1 \parallel (E_2 \parallel E_3) \equiv_e (E_1 \parallel E_2) \parallel E_3$
4. $B_1 \equiv_b B_1' \Rightarrow B_1 \rhd B_2 \equiv_e B_1' \rhd B_2$
5. $B_2 \equiv_b B_2' \Rightarrow B_1 \rhd B_2 \equiv_e B_1 \rhd B_2'$

d) axioms for systems

1. $B_1 \equiv_b B_2 \wedge E_1 \equiv_e E_2 \Rightarrow (B_1, E_1) \equiv (B_2, E_2)$

Table 5.6: Reduction semantics of B^e_{core}.

Definition 5.3.5. *A B^e_{core} transition system is referred to as $(\overline{\mathcal{S}}_e, \to)$, where $\overline{\mathcal{S}}_e$ is the set of well-formed systems and $\to \subseteq \overline{\mathcal{S}}_e \times \overline{\mathcal{S}}_e$ is the transition relation (see Tab.5.6) .*

Consider a RAM R with program $(1, I_1), ..., (m, I_m)$ and state $(j, k_1, ..., k_n)$. The encoding of the RAM is $[\![(j, k_1, ..., k_n)]\!]^e_R = (B, E)$ where the definition of B and E is reported in Tab. 5.7 and the bio-process $Switch_j$ is equal to the one defined in the previous section. We assume $\delta(x) = 1$ for all $x \in \mathcal{N}$ and:

$$\alpha(\Delta_1, \Delta_2) = \begin{cases} 1 & \text{if } \Delta_1 = \Delta_2 \\ 0 & \text{otherwise} \end{cases}$$

The encoding produces a system $S = (B, E)$ in $\overline{\mathcal{S}}_e$. The bio-process B is a parallel composition of a switching box, which controls the activation of the instructions sequence, of m boxes encoding instructions and of n boxes encoding registers; the two types of

instructions are encoded in different ways, but in both cases the encoding box is activated by performing an inter-communication with the box $Switch_j$. The composition of events E contains a couple of events for each register which controls the transformation of a register with content 0 to a register with content 1, and vice-versa.

The modelling of the register r_j depends on its content. If the content of the register is 0, then the box B_j^0 is used; if the content of the register is greater than 0, then bio-process $B_j^{test} \parallel B_j^l$ is used.

The box encoding the instruction $(i, Incr(r_j))$, after its activation, consumes an inter-communication with the box encoding the register r_j (through the interfaces of sorts $IReg_j$), representing a request for its increment; then the instruction box waits for another inter-communication (through the interfaces of sorts $IAck_j$) with the register box; a kind of acknowledgement indicating that the increment has been executed. Finally, after the acknowledgement, the box replicates its internal machinery and performs an inter-communication with the switching box (through the interfaces of sorts Ins) for the activation of instruction $i+1$. The behaviour of the register box depends on its content. If the content is 0, after consuming the increment inter-communication, the box becomes structurally congruent to the box B_j^{split0}, causing the activation of event $ZeroToOne_j$. This event substitutes B_j^{split0} with the bio-process $B_j^{test} \parallel B_j^{split1}$. After consuming the acknowledgement inter-communication on the channel ack_i, the box becomes structurally congruent to the box B_j^1, indicating that the register has been correctly incremented. Notice that when the event $ZeroToOne_j$ is enabled no other actions in the system are enabled. This guarantees that the register transformation is achieved between the request of the instruction and the acknowledgement of the register.

If the content is greater than zero, the increment inter-communication enables the internal replication of the process $Increment$, representing the addition of 1 on the content of the register. The corresponding acknowledgement is performed after the replication, consuming the acknowledgement inter-communication on the channel ack_i.

The box encoding the instruction $(i, DecJump(r_j, s))$, after its activation, consumes first an inter-communication with the box encoding for the register r_j, in order to test its content (through the interfaces of sorts $Test_j$). In particular, the instruction box receives a name *yes* if the content of the register r_j is greater than zero and receives the name *no* otherwise. With the choice operator two alternative behaviours are encoded, depending on the result of the testing communication. In case of *yes* name reception, the instruction box consumes an inter-communication with the r_j register box, representing a request for its decrement (through the interface of sorts $DReg_j$), then waits for an acknowledgement indicating that the decrement has been executed (through the interfaces of sorts $DAck_j$) and finally replicates its internal machinery and performs an inter-communication with the

$$
\begin{array}{rcl}
B & \triangleq_B & Switch_j \parallel [\![(1, I_1)]\!]^e_R \parallel \cdots \parallel [\![(m, I_m)]\!]^e_R \parallel [\![r_1 = k_1]\!]^e_R \parallel \\
 & & \cdots \parallel [\![r_n = k_n]\!]^e_R \\
 & & \\
E & \triangleq_E & ZeroToOne_1 \parallel OneToZero_1 \parallel \cdots \parallel ZeroToOne_n \parallel \\
 & & OneToZero_n \\
 & & \\
[\![(i, Incr(r_j))]\!]^e_R & = & \oplus(act,\, Ins_i) \oplus(next,\, Ins) \oplus(inc,\, IReg_j) \oplus(ack,\, IAck_j) \\
 & & [\ *\, x?e.\, Inc_i \mid Inc_i\ ] \\
Inc_i & \triangleq_P & act?e.\, inc!e.\, ack?e.\, x!e.\, next!ins_{i+1}.\, \mathsf{nil} \\
 & & \\
[\![(i, DecJump(r_j, s))]\!]^e_R & = & \oplus(act,\, Ins_i) \oplus(next,\, Ins) \oplus(dec,\, DReg_j) \\
 & & \oplus(ack,\, DAck_j) \oplus(test,\, Test_j)^p[\ *\, x?e.\, DecJ_i \mid DecJ_i\ ] \\
DecJ_i & \triangleq_P & act?e.\, test?t.\, (t!e.\, \mathsf{nil} \mid (yes?e.\, Dec_i + no?e.\, Jump_i)) \\
Dec_i & \triangleq_P & dec!e.\, ack?e.\, x!e.\, next!ins_{i+1}.\, \mathsf{nil} \\
Jump_i & \triangleq_P & x!e.\, next!ins_s.\, \mathsf{nil} \\
 & & \\
[\![r_j = l]\!]^e_R & = & \begin{cases} B^0_j & \text{if } l = 0 \\ B^{test}_j \parallel B^l_j & \text{otherwise} \end{cases} \\
B^0_j & \triangleq_B & \oplus(test,\, Test_j) \oplus(inc,\, IReg_j) \oplus(ack_i,\, IAck_j) \\
 & & \oplus(ack_d,\, DAck_j)[\ inc?e.\, \mathsf{nil} \mid\ *\, test!no.\, \mathsf{nil}\ ] \\
B^{test}_j & \triangleq_B & \oplus(test,\, Test_j)[\ *\, test!yes.\, \mathsf{nil}\ ] \\
B^l_j & \triangleq_B & \oplus(inc,\, IReg_j) \oplus(dec,\, DReg_j) \oplus(ack_i,\, IAck_j) \\
 & & \oplus(ack_d,\, DAck_j)[\ *\, inc?e.\, Increment \mid AckL_l \mid DecL_l\ ] \\
DecL_l & \triangleq_P & \underbrace{dec?e.\, \mathsf{nil} \mid \cdots \mid dec?e.\, \mathsf{nil}}_{l} \\
AckL_l & \triangleq_P & \underbrace{ack_d!e.\, \mathsf{nil} \mid \cdots \mid ack_d!e.\, \mathsf{nil}}_{l-1} \\
Increment & \triangleq_P & ack_d!e.\, \mathsf{nil} \mid ack_i!e.\, dec?e.\, \mathsf{nil} \\
 & & \\
ZeroToOne_i & \triangleq_E & B^{split0}_j \rhd B^{test}_j \parallel B^{split1}_j \\
OneToZero_i & \triangleq_E & B^{test}_j \parallel B^{join1}_j \rhd B^{join0}_i \\
 & & \\
B^{split0}_j & \triangleq_B & \oplus(test,\, Test_j) \oplus(inc,\, IReg_j) \oplus(ack_i,\, IAck_j) \\
 & & \oplus(ack_d,\, DAck_j)[\ *\, test!no.\, \mathsf{nil}\ ] \\
B^{join0}_j & \triangleq_B & \oplus(test,\, Test_j) \oplus(inc,\, IReg_j) \oplus(ack_i,\, IAck_j) \\
 & & \oplus(ack_d,\, DAck_j)[\ ack_d!e.\, inc?e.\, \mathsf{nil} \mid\ *\, test!no.\, nil\ ] \\
B^{split1}_j & \triangleq_B & \oplus(inc,\, IReg_j) \oplus(dec,\, DReg_j) \oplus(ack_i,\, IAck_j) \\
 & & \oplus(ack_d,\, DAck_j)[\ *\, inc?e.\, Increment \mid ack_i!e.dec?e.\, \mathsf{nil}\ ] \\
B^{join1}_j & \triangleq_B & \oplus(inc,\, IReg_j) \oplus(dec,\, DReg_j) \oplus(ack_i,\, IAck_j) \\
 & & \oplus(ack_d,\, DAck_j)[\ *\, inc?e.\, Increment\ ]
\end{array}
$$

Table 5.7: Encoding of RAMs with B^e_{core}.

switching box (through the interfaces of sorts Ins) for the activation of instruction $i+1$. Instead, in case of *no* name reception, the box simply replicates its internal machinery and performs an inter-communication with the switching box (through the interfaces of sorts Ins) for the activation of instruction s. The behaviour of the register box in the case of decrement depends on its content.

If a decrement inter-communication is consumed by a box representing a register with content 1, then the box becomes structurally congruent to the box B_j^{join1}. This box activates the event $OneToZero_j$, which substitutes the bio-process $B_j^{test} \parallel B_j^{join1}$ with the box B_j^{join0}. After consuming the acknowledgement inter-communication on the channel ack_d, the box becomes structurally congruent to the box B_j^0, indicating that the register has been correctly decremented. Notice that, also in this case, when the event $OneToZero_j$ is enabled no other actions in the system are enabled.

If a decrement inter-communication is consumed by a box representing a register with content greater than zero, then the acknowledgement inter-communication is then consumed, deleting an instance of the parallel processes composing the $AckList_l$ process and hence representing the decrement of 1.

A formal proof of the encoding correctness follows.

Lemma 5.3.6. *Let R be a RAM with program $(1, I_1), ..., (m, I_m)$ and state $(j, k_1, ..., k_n)$. If $(j, k_1, ..., k_n) \rightarrow_R (j', k_1', ..., k_n')$, then there exists a system S such that $[\![(j, k_1, ..., k_n)]\!]_R^e \rightarrow^+ S$ and $S \equiv [\![(j', k_1', ..., k_n')]\!]_R^e$.*

Proof. The proof is by case analysis. There are five different cases: (i) Instruction $I_j = DecJump(r_l, s)$ and r_l value greater than one; (ii) Instruction $I_j = DecJump(r_l, s)$ and r_l value equal to one; (iii) Instruction $I_j = DecJump(r_l, s)$ and r_l value equal to zero; (iv) Instruction $I_j = Incr(r_l)$ and r_l value greater than zero; (v) Instruction $I_j = Incr(r_l)$ and r_l value equal to zero. We prove only case (ii), because the other cases are similar.

(ii) We consider the computation of the system $[\![(j, k_1, ..., k_n)]\!]_R^e$. As in Lemma 5.3.2 an inter-communication between $Switch_j$ and $[\![(j, DecJump(r_l, s))]\!]_R^e$ is the first consumed action. The two boxes synchronize on output $i_j!e$ and input $act?e$ through their interfaces of sorts Ins_j. This cause the activation of the $[\![(j, DecJump(r_l, s))]\!]_R^e$ component. Also in this case, after the inter-communication the components codifying for the other instructions, the registers and the switching box are blocked.

We have that the content of the register is 1 and hence it is encoded by the bio-process $B_l^{test} \parallel B_l^1$. The activation of the $[\![(j, DecJump(r_l, s))]\!]_R^e$ box causes the consumption of an inter-communication between the box B_l^{test} and the instruction box; since the content of the register is 1, then the instruction box receives the name yes, indicating that the content of the register is greater than zero. After the inter-communication the instruction box performs another intra-communication with the register box, which causes the

activation of the process Dec_j. This process synchronizes with the box B_l^1, consumes an inter-communication on output $dec!e$ through interface of sorts $DReg_l$ and remains blocked on input $ack?e$; after that inter-communication, the box B_l^1 becomes structurally congruent to the box B_l^{join1} and the event $OneToZero_l$ becomes active. The execution of the event substitutes the bio-process $B_l^{test} \parallel B_l^{join1}$ with the box B_l^{join0}, which consumes an intra-communication with the instruction box (on interfaces of sorts $DAck_l$) and becomes structurally congruent to $[\![r_l = 0]\!]_R^e$. Moreover, the last inter-communication unblocks the instruction box, which consumes an intra-communication on channel x, replicating its internal machinery, and enables the process $next!ins_{j+1}.nil$. This produces a synchronization between the instruction box and the switching box. Indeed, the boxes consume an inter-communication on output $next!ins_{j+1}$ and input $ins?type$ through interfaces of sorts Ins.

The instruction box is now returned in its form $[\![(j, DecJump(r_l, s))]\!]_R^e$, while in the switching box the process

$$(ins_{j+1}!e.nil \mid (\textstyle\sum_{o=1}^{m} ins_o?e.x!e.i_o!e.nil))$$

is enabled. An intra-communication on channel ins_{j+1} is consumed, the internal machinery is replicated with an intra-communication on channel x and the switching box is now structurally congruent to the box $Switch_{j+1}$. □

Lemma 5.3.7. *Let R be a RAM with program $(1, I_1), ..., (m, I_m)$ and state $(j, k_1, ..., k_n)$. If the system $S = [\![(j, k_1, ..., k_n)]\!]_R^e$ can produce a transition $S \to S_1$, then there exists a computation $S \to S_1 \to S_2 \to \cdots \to S_l$ such that $S_l = [\![(j', k'_1, ..., k'_n)]\!]_R^e$ and $(j, k_1, ..., k_n) \to_R (j', k'_1, ..., k'_n)$.*

Proof. Consider the structure of the system $S = [\![(j, k_1, ..., k_n)]\!]_R^e$. As in Lemma 5.3.3, if the system S can perform a first step $S \to S_1$, this corresponds to an inter-communication between the box $Switch_j$ and the box encoding for the instruction (j, I_j), representing the activation of the instruction box. By encoding definition this means that the instruction (j, I_j) exists; hence the instruction can be executed in the state $(j, k_1, ..., k_n)$ of the RAM R, generating a new state $(j', k'_1, ..., k'_n)$.

The proof is by case analysis. There are five different cases: (i) Instruction $I_j = DecJump(r_l, s)$ and r_l value greater than one; (ii) Instruction $I_j = DecJump(r_l, s)$ and r_l value equal to one; (iii) Instruction $I_j = DecJump(r_l, s)$ and r_l value equal to zero; (iv) Instruction $I_j = Incr(r_l)$ and r_l value greater than zero; (v) Instruction $I_j = Incr(r_l)$ and r_l value equal to zero. In all the cases, it is possible to show that from the moment in which the switch activates an instruction till the moment in which the switch is able to activate a new instruction, the computation proceeds deterministically (up to structural

congruence $\equiv$). The encoding is hence deterministic up to structural congruence. We prove only case (v), because the other cases can be proved similarly.

(v) By encoding definition we have that the structure of the instruction box is

$$\begin{array}{rcl} [\![(j, Incr(r_l))]\!]^e_R & = & \oplus(act,\, Ins_j) \oplus(next,\, Ins) \oplus(inc,\, IReg_l) \oplus(ack,\, IAck_l) \\ & & [\ *\, x?e.\, Inc_j \mid act?e.\, inc!e.\, ack?e.\, x!e.\, next!ins_{j+1}.\, \mathsf{nil}\ ] \end{array}$$

This box is the only able to synchronize with the box $Switch_j$ for an inter-communication through interfaces of sorts Ins_j. After the communication we have that in S_1 the box encoding the instruction j becomes structurally congruent to

$$\begin{array}{rcl} B' & = & \oplus(act,\, Ins_j) \oplus(next,\, Ins) \oplus(inc,\, IReg_l) \oplus(ack,\, IAck_l) \\ & & [\ *\, x?e.\, Inc_j \mid inc!e.\, ack?e.\, x!e.\, next!ins_{j+1}.\, \mathsf{nil}\ ] \end{array}$$

Now, the only possible action $S_1 \to S_2$ is the inter-communication between the box B' and the box $[\![r_l = 0]\!]^e_R$ through their interfaces of sorts $IReg_l$ on output $inc!e$ and input $inc?e$, respectively. After the inter-communication the instruction box remains blocked on input $ack?e$ over interface of sorts $IAck_l$ and the box encoding for the register r_l becomes structurally congruent to the box B^{split0}; the event $ZeroToOne_j$ is now active. The execution of the event, which correspond to the action $S_2 \to S_3$, substitutes in S_2 the box B_l^{split0} with the bio-process $B_l^{test} \parallel B_l^{split1}$. At this point, the action $S_3 \to S_4$ is an inter-communication between the box B_l^{split1} and the instruction box; the register box becomes structurally congruent to $[\![r_l = 1]\!]^e_R$, while the instruction box is unblocked and structurally congruent to

$$\begin{array}{rcl} B'' & = & \oplus(act,\, Ins_j) \oplus(next,\, Ins) \oplus(inc,\, IReg_l) \oplus(ack,\, IAck_l) \\ & & [\ *\, x?e * Inc_j \mid x!e * next!ins_{j+1}.\, \mathsf{nil}\ ] \end{array}$$

Now, the action $S_4 \to S_5$ is the intra-communication of B'' on channel x which becomes a box B''' with internal structure $*x?e.\, Inc_j \mid next!ins_{j+1}.\, \mathsf{nil}$, and the action $S_5 \to S_6$ is the inter-communication between the box B''' and the switching box. After the inter-communication, the instruction box returns in its initial form $[\![(j, Incr(r_l))]\!]^e_R$ and the switching box starts a sequence of intra-communications which produces a box structurally congruent to $Switch_{j+1}$ and representing the sequence of actions $S_6 \to S_7 \to S_8$. It is easy to see that S_8 is structurally congruent to $[\![(j+1, k_1, ..., k_{l-1}, 1, k_{l+1}, ..., k_n)]\!]^e_R$. $\square$

Lemma 5.3.6 and Lemma 5.3.7 can now be used for proving the undecidability of termination for BS^e_{core} systems.

Theorem 5.3.8. *Let R be a RAM with a program $(1, I_1), ..., (m, I_m)$ and an initial state $(j, k_1, ..., k_n)$. The the computation of the RAM R terminates if and only if the computation of the system $S = [\![(j, k_1, ..., k_n)]\!]^e_R$ terminates.*

Proof. The Theorem can be proved similarly to Theorem 5.3.4 and by using Lemma 5.3.6 and Lemma 5.3.7. □

5.4 Discussion

The framework developed for B_{core}, can be considered as a basis for a further development that can enable us to reason about the decidability of other interesting problems like the covering problem, i.e., decide, given two states s and t, whether starting from s it is possible to reach a state $t' \geq t$. Moreover, we are confident that these decidability results can be extended to B_{core} enriched with complexes, change actions and conditions. Moreover, the recent paper by Cardelli and Zavattaro [14], suggests us that we can obtain Turing equivalence also by enriching the B_{core} subset with complexes and a single type of event that simply replicates a box; we plan to investigate this aspect in the future.

Conversely, due to the absence of restriction, to the static structure of boxes and interfaces and to the fact that the set of names used in a system is finite, we feel that a relation between B_{core} and CCS exists. An encoding of B_{core} into CCS, indeed, would allow us to exploit results already proved for CCS.

Concluding, although this work allows us to conclude that BlenX is a Turing equivalent language, we think that the obtained results represent also an investigation into how the addition of global priorities affects the expressive power of a language and on the role that some high-powered features like restriction operator play in Turing equivalence encodings. Moreover, we think these results are a basis for further investigations and for a better understanding of how different primitives and operators can be added, deleted or combined to obtain classes of languages with different computational power.

Chapter 6

Stochastic extension

In this chapter we introduce the stochastic extension of BlenX, that we call sBlenX. Considering the results of the previous chapter, sBlenX is obtained by first limiting BlenX to two levels of priority and to simplified events that do not involve complexes and subcomplexes. The obtained language is then decorated with quantitative information which, as usual in the stochastic setting [91, 48], are used to derive the speed and probabilities associated with reactions.

We start by introducing the syntax of sBlenX, along with its operational semantics, which is given in a SOS style. We continue by explaining how to compute the probability of executing a certain high priority reaction or a certain low priority reaction. Since the final goal is to provide efficient stochastic simulations of sBlenX system, we introduce then a stochastic abstract machine that compresses the information of systems compressing boxes of the same species and isomorphic complexes. We prove that the abstract machine is correct with respect to the stochastic semantics of sBlenX and that it respects reaction probabilities. We finally present a stochastic simulation algorithm, that works on the stochastic abstract machine and that is based on the NRM (see Sec. 2.2).

6.1 Syntax and semantics

The syntaxes of BlenX and sBlenX are very similar. One of the main differences is that all the values $p \in \mathbb{N}$, representing priority values, are substituted with values $r \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, representing both priorities and stochastic rates. Since sBlenX implements two levels of priorities, we have that values in $\mathbb{R}_{\geq 0}$ represent lower priority values, while ∞ represents the high priority value and is used to identify what we call *immediate actions*. Thus, immediate actions have the precedence with respect to actions associated with rates in $\mathbb{R}_{\geq 0}$. Note that, as usual, values in $\mathbb{R}_{\geq 0}$ represent single parameters defining exponential distributions that drive the stochastic behaviour of actions. We recall that an exponential

distribution with rate s has cumulative distribution function $F(t) = 1 - e^{-st}$, where t represents the time. The parameter s determines the shape of the curve. The greater the s parameter, the faster $F(t)$ approaches its asymptotic value. The probability of performing an action with parameter s within time t is $F(t) = 1 - e^{-st}$, so s determines the time t needed to obtain a probability near to 1. The probability density function is $f(t) = se^{-st}$.

The underlying kinetics assumed in sBlenX is the law of mass action (i.e., the rate of a reaction is proportional to the product of the reactants species population). However, *generic kinetic laws* (i.e., functions that approximate sequences of reactions) can be used. Although the law of mass action is enough to model many biological scenarios, generic kinetic laws are useful when it is difficult to derive certain information from the experiments, e.g., the reaction rates of elementary steps, or when there are different time-scales for the reactions.

In order to enable generic kinetic laws, we first need to extend the syntax with the new syntactic category of functions. Functions describe mathematically a particular generic kinetic law. Since generic kinetic laws are used to approximate sequences of reactions, it is reasonable to confine their use only to the kind of reactions that in sBlenX (and also in BlenX) are used to abstract coarse grained description, i.e., events and inter-communications. Intra-communications, change actions, complexations, decomplexations and complex-communications, can be immediate or governed only by the law of mass action; they can be associated only with rates in $\mathbb{R}_{\geq 0} \cup \{\infty\}$. Inter-communications can be associated with rates in $\mathbb{R}_{\geq 0} \cup \{\infty\}$ and functions. Events can be associated only with ∞ and functions.

	...		...		...	...
I	::=	**Interfaces**	E	::=	**Events**
	$K(x,\Gamma)^r$	interface		$B \rhd_h B$	event
	$K(x,\Gamma)^r\ I$	sequence		...	
π	::=	**Prefixes**	F	::=	**Functions**
	$\mathsf{ch}(x,\Gamma,r)$	change		s	real value
	...			$\lvert I[P]\rvert$	number of species
				$op_m\ F$	unary operator
				$F\ bop_m\ F$	binary operator

Table 6.1: sBlenX syntax.

Tab. 6.1 highlights the differences between the syntaxes of BlenX and sBlenX, where

$r \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, $s \in \mathbb{R}_{\geq 0}$ and $h \in \{\infty\} \cup \mathcal{F}$; we use g to identify values in $\mathbb{R}_{\geq 0} \cup \{\infty\} \cup \mathcal{F}$. The set $\mathcal{F}$ denotes all the possible functions generated by the symbol F, where we assume $op_m \in \{log, sqrt, exp, +, -\}$ and $bop_m \in \{+, -, \times, /\}$. Note that in the definition of interfaces and change actions, priorities are substituted with values in $\mathbb{R}_{\geq 0} \cup \{\infty\}$. Moreover, events are defined over values in $\{\infty\} \cup \mathcal{F}$ and have a form that reflects the fact that here we consider the subset of BlenX where events cannot involve complexes or sub-complexes. Like in Chapt. 5, indeed, we assume $B_1 \rhd_h B_2$ to be an abbreviation of $(B_1, \emptyset) \rhd_h (B_2, \emptyset)$. We denote with $\mathcal{S}_s$ all the possible systems that can be generated by the syntax of sBlenX.

Given a function $f \in \mathcal{F}$ and a systems $S \in \mathcal{S}_s$, we denote with $\wr f \int_S$ the evaluation of the function f in the system S. The evaluation computes the value of the mathematical expression implemented by f where with $|I[P]|$ we denote the number of boxes $I'[\,P'\,]_n$ in the bio-process of S such that $I[P] \sim_s I'[P']$.

In the stochastic setting, we assume a total function $\delta^s : \mathcal{N} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$, used to associate quantitative information to communication channels. Moreover, we assume a stochastic compatibility function α^s, that is a symmetric function:

$$\alpha^s : \mathcal{T}^2 \to (0, 0, (\mathbb{R}_{\geq 0} \cup \{\infty\} \cup \mathcal{F})) \cup (\mathbb{R}_{\geq 0} \cup \{\infty\})^3$$

Note that $\alpha^s(\Delta, \Gamma) = (b, u, c)$ can contain a function in c (the only place in the triple where a function can appear) only if b and u are equal to zero; this reflects the idea that functions can be used as a compatibility value only to describe inter-communications. As before, with 0 we indicate that the two sorts are not compatible. Moreover, we use $\alpha^s_b(\Delta, \Gamma)$, $\alpha^s_u(\Delta, \Gamma)$, and $\alpha^s_c(\Delta, \Gamma)$ to mean, respectively, the first, the second, and the third projection of $\alpha^s(\Delta, \Gamma)$.

The definition of structural congruence and static semantics of BlenX can be easily adapted for sBlenX just by replacing p with r or h in all the rules of Tab. 4.6 and of Tab. 4.7. All the results obtained for BlenX are valid also for sBlenX. We overload symbols when unambiguous.

Definition 6.1.1. *The set of well-formed sBlenX systems, denoted with $\overline{\mathcal{S}}_s$, is the set defined as $\overline{\mathcal{S}}_s = \{S \in \mathcal{S}_s \mid \vdash S : ok\}$.*

Theorem 6.1.2. *Let $S \in \overline{\mathcal{S}}_s$ and $S' \in [S]_{\equiv}$. Then $S' \in \overline{\mathcal{S}}_s$.*

The dynamics of a sBlenX well-formed system is formally specified by the reduction semantics reported in Tab. 6.2. Reduction rules rely on the structural congruence relation

(s1) $$\frac{\wr C \int_I = true}{(I[\,\langle C\rangle \mathsf{ch}(x,\Gamma,r).P + G \mid P_1\,]_n, E, \xi) \to_r (I_1[\,P \mid P_1\,]_n, E, \xi\{{}^{\Gamma n}/_{\Delta n}\})}$$
where $I = K(x,\Delta)^{r'}\, I^*$ and $I_1 = K(x,\Gamma)^{r'}\, I^*$ and $\Gamma \notin \mathsf{sorts}(I^*)$

(s2) $$\frac{\wr C_1 \int_I = true \quad \wr\, C_2 \int_I = true}{(I[\,\langle C_1\rangle x!z.P_1 + G_1 \mid \langle C_2\rangle x?y.P_2 + G_2 \mid P\,]_n, E, \xi) \to_{rate_I(x)} (I[\,P_1 \mid P_2\{{}^z/_y\} \mid P\,]_n, E, \xi)}$$

(s3) $$(I_1[\,P_1\,]_n \parallel I_2[\,P_2\,]_m, E, \xi) \to_r (I_1'[\,P_1\,]_n \parallel I_2'[\,P_2\,]_m, E, \xi \cup \{\{\Delta_1 n, \Delta_2 m\}\})$$
where $I_i = \oplus(x_i,\Delta_i)^{r_i}\, I_i^*$ and $I_i' = \otimes(x_i,\Delta_i)^{r_i}\, I_i^*$ for $i = 1,2$
and provided $\alpha_b^s(\Delta_1,\Delta_2) = r$

(s4) $$(I_1[\,P_1\,]_n \parallel I_2[\,P_2\,]_m, E, \xi \cup \{\{\Delta_1 n, \Delta_2 m\}\}) \to_r (I_1'[\,P_1\,]_n \parallel I_2'[\,P_2\,]_m, E, \xi)$$
where $I_i = \otimes(x_i,\Delta_i)^{r_i}\, I_i^*$ and $I_i' = \oplus(x_i,\Delta_i)^{r_i}\, I_i^*$ for $i = 1,2$
and provided $\alpha_u^s(\Delta_1,\Delta_2) = r$

(s5) $$\frac{\wr C_1 \int_{I_1} = true \quad \wr\, C_2 \int_{I_2} = true}{(I_1[\,M_1 \mid Q_1\,]_n \parallel I_2[\,M_2 \mid Q_2\,]_m, E, \xi) \to_g (I_1[\,R_1 \mid Q_1\,]_n \parallel I_2[\,R_2\{{}^z/_w\} \mid Q_2\,]_m, E, \xi)}$$
where $M_1 = \langle C_1\rangle x_1!z.\, R_1 + M_1'$ and $M_2 = \langle C_2\rangle x_2?w.\, R_2 + M_2'$ and
where $I_i = K(x_i,\Delta_i)^{r_i}\, I_i^*$ and $z \notin \mathsf{sub}(I_i)$ for $i = 1,2$ and provided
$(K = \otimes$ and $\alpha_c^s(\Delta_1,\Delta_2) = g$ and $\{\Delta_1 n, \Delta_2 m\} \in \xi)$ or
$(K = \oplus$ and $\alpha_c^s(\Delta_1,\Delta_2) = g$ and $\alpha_b^s(\Delta_1,\Delta_2) = \alpha_u^s(\Delta_1,\Delta_2) = 0)$

(s6) $$(B_1, B_1 \rhd_g B_2 \parallel E, \xi) \to_g (B_2, B_1 \rhd_h B_2 \parallel E, \xi)$$

(s7) $$\frac{(B, E, \xi) \to_g (B', E, \xi') \quad (\mathsf{id}(B) \cup \mathsf{id}(B')) \cap \mathsf{id}(B_1) = \emptyset}{(B \parallel B_1, E, \xi) \to_g (B' \parallel B_1, E, \xi')}$$

(s8) $$\frac{S_1 \equiv S_1' \quad S_1' \to_g S_2' \quad S_2' \equiv S_2}{S_1 \to_g S_2}$$

Table 6.2: Reduction semantics of sBlenX.

over systems. The semantics, moreover, uses the following function:

$$rate_I(x) = \begin{cases} r & \text{if } \exists\Delta.\ K(x,\Delta)^r \in I \\ \delta^s(x) & \text{otherwise} \end{cases}$$

It is easy to see that the reduction rules in Tab.6.2 are very similar to the ones presented in Tab.4.13. The main difference is in the axiom for events where we are sure that B_1 is not part of a complex and hence no modifications in the structure of ξ are taken.

Theorem 6.1.3. *Let $S \in \overline{\mathcal{S}}_s$. Then $S \rightarrow_g S'$ implies $S' \in \overline{\mathcal{S}}_s$.*

Following [85], we provide a measure to calculate the probability to perform a certain reaction $S \rightarrow_g S'$. Since immediate actions have the precedence wirth respect to actions with rate in $\mathbb{R}_{\geq 0}$ we distinguish the two cases. When we have an *immediate reaction* $S \rightarrow_\infty S'$, we say that the probability of performing the reaction is given by $1/R^0(S)$, where $R^0(S)$ denotes the *immediate apparent probability* of S that is the total number of immediate reactions that S can perform. When instead we have a *stochastic reaction* $S \rightarrow_g S'$, with $g \in \mathbb{R}_{\geq 0} \cup \mathcal{F}$, we say that the probability of performing the reaction is given by $s/R^1(S)$, where s is equal to g if $g \in \mathbb{R}_{\geq 0}$ or is equal to $\wr g \int_S$ if $g \in \mathcal{F}$ and where $R^1(S)$ denotes the *stochastic apparent rate* of S, i.e., the sum of the rates of all the active stochastic reactions in S. In the following we describe how to calculate the values $R^0(S)$ and $R^1(S)$, by considering separately each reaction type.

To simplify the treatment of the rest of the chapter we define a class of systems that we call *safe normal form* systems. Given a system $S \in \overline{\mathcal{S}}_s$, with $\hat{S}$ we represent a *safe normal form* system obtained from S.

Definition 6.1.4. *Let $S = (B, E, \xi) \in \overline{\mathcal{S}}$. We say that S is in safe normal form if :*
1) $\mathsf{fn}(B) \cap \mathsf{sub_t}(B) = \emptyset$;
2) $B = \prod_{i=1}^{n} B_i$ with $B_i = I_i[P_i]_{n_i}$ for all $i < n$ and $B_n = \mathsf{Nil}$;
3) $E = \prod_{i=1}^{n} E_i$ with $E_i = B_i \rhd_{h_i} B_i'$ for all $i < n$ and $E_n = \mathsf{Nil}$.

Function $\mathsf{sub_t}$ (see Def. 5.1.1) allows to obtain the set containing all the subjects defined in all the boxes composing a bio-process. Given a system S, a corresponding system $\hat{S}$ is not unique. However, for all the possible rearrangements in safe normal form, the following Lemma hold.

Proposition 6.1.5. *Let $S \in \overline{\mathcal{S}}$. Then $S \equiv \hat{S}$, for all $\hat{S}$.*

Given a system $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}$, we start by defining the immediate apparent probability and the stochastic apparent rate for change actions, denoted by $R^0_{change}(B)$ and $R^1_{change}(B)$, respectively. These values can be computed by inspecting the structure of all the boxes composing B and finding all the change actions guarded by a condition that evaluates to *true* with respect to the corresponding box interfaces:

$$R^m_{change}(\mathsf{Nil}) = 0$$
$$R^m_{change}(B_0 \parallel B_1) = R^m_{change}(B_0) + R^m_{change}(B_1)$$
$$R^m_{change}(I[P]_n) = \sum_{x \in \mathsf{sub}(I)} Ch^m_x(I, P)$$

$$In_x^I(P_1 \mid P_2) = In_x^I(P_1) + In_x^I(P_2)$$
$$In_x^I(\mathsf{nil}) = 0$$
$$In_x^I(*\langle C\rangle\pi.P) = In_x^I(\langle C\rangle\pi.P)$$
$$In_x^I(\langle C\rangle\pi.P) = \begin{cases} 1 & \text{if } \wr C\int_I = true \\ & \text{and } \mathsf{InSub}(\pi) = \{x\} \\ 0 & \text{otherwise} \end{cases}$$
$$In_x^I(M_1 + M_2) = In_x^I(M_1) + In_x^I(M_2)$$

$$Out_x^{I,k}(P_1 \mid P_2) = Out_x^{I,k}(P_1) + Out_x^{I,k}(P_2)$$
$$Out_x^{I,k}(\mathsf{nil}) = 0$$
$$Out_x^{I,k}(*\langle C\rangle\pi.P) = Out_x^{I,k}(\langle C\rangle\pi.P)$$
$$Out_x^{I,k}(\langle C\rangle\pi.P) = \begin{cases} 1 & \text{if } \wr C\int_I = true \\ & \text{and } \mathsf{OutSub}^\epsilon(\pi) = \{x\} \\ & \text{and } k = 0 \\ 1 & \text{if } \wr C\int_I = true \\ & \text{and } \mathsf{OutSub}^I(\pi) = \{x\} \\ & \text{and } k = 1 \\ 0 & \text{otherwise} \end{cases}$$
$$Out_x^{I,k}(M_1 + M_2) = Out_x^{I,k}(M_1) + Out_x^{I,k}(M_2)$$

$$In_{\Delta n}(\mathsf{Nil}) = 0$$
$$In_{\Delta n}(B_1 \parallel B_2) = In_{\Delta n}(B_1) + In_{\Delta n}(B_2)$$
$$In_{\Delta n}(I[\,P\,]_{n_1}) = \begin{cases} In_x^I(P) & \text{if } n = n_1 \text{ and} \\ & \exists x.\, \otimes(x,\Delta)^r \in I \\ 0 & \text{otherwise} \end{cases}$$

$$Out_{\Delta n}(\mathsf{Nil}) = 0$$
$$Out_{\Delta n}(B_1 \parallel B_2) = Out_{\Delta n}(B_1) + Out_{\Delta n}(B_2)$$
$$Out_{\Delta n}(I[\,P\,]_{n_1}) = \begin{cases} Out_x^{I,1}(P) & \text{if } n = n_1 \text{ and} \\ & \exists x.\, \otimes(x,\Delta)^r \in I \\ 0 & \text{otherwise} \end{cases}$$

$$In_\Delta(\mathsf{Nil}) = 0$$
$$In_\Delta(B_1 \parallel B_2) = In_\Delta(B_1) + In_\Delta(B_2)$$
$$In_\Delta(I[\,P\,]_n) = \begin{cases} In_x^I(P) & \text{if } \exists x.\, \oplus(x,\Delta)^r \in I \\ 0 & \text{otherwise} \end{cases}$$

$$Out_\Delta(\mathsf{Nil}) = 0$$
$$Out_\Delta(B_1 \parallel B_2) = Out_\Delta(B_1) + Out_\Delta(B_2)$$
$$Out_\Delta(I[\,P\,]_n) = \begin{cases} Out_x^{I,1}(P) & \text{if } \exists x.\, \oplus(x,\Delta)^r \in I \\ 0 & \text{otherwise} \end{cases}$$

$$Mix_x^I(P_1 \mid P_2) = Mix_x^I(P_1) + Mix_x^I(P_2)$$
$$Mix_x^I(M) = In_x^I(M) \times Out_x^{I,0}(M)$$

$$Mix_{(\Delta,\Gamma)}(\mathsf{Nil}) = 0$$
$$Mix_{(\Delta,\Gamma)}(B_1 \parallel B_2) = Mix_{(\Delta,\Gamma)}(B_1) + Mix_{(\Delta,\Gamma)}(B_2)$$
$$Mix_{(\Delta,\Gamma)}(I[\,P\,]_n) = In_\Delta(I[\,P\,]_n) \times Out_\Gamma(I[\,P\,]_n)$$

$$Ch_x^m(I, \mathsf{nil}) = Ch_x^m(I, \langle C\rangle x?y.P) = Ch_x^m(I, \langle C\rangle x!y.P) = 0$$
$$Ch_x^m(I, P_1 \mid P_2) = Ch_x^m(I, P_1) + Ch_x^m(I, P_2)$$
$$Ch_x^m(I, *\langle C\rangle\pi.P) = Ch_x^m(I, \langle C\rangle\pi.P)$$
$$Ch_x^m(I, M_1 + M_2) = Ch_x^m(I, M_1) + Ch_x^m(I, M_2)$$
$$Ch_x^m(I, \langle C\rangle\mathsf{ch}(y,\Gamma,r)) = \begin{cases} r & \text{if } x = y \text{ and } I = I_1^* \oplus(x,\Delta)^{r'} I_2^* \text{ and } \Gamma \notin \mathsf{sorts}(I_1^* I_2^*) \\ & \text{and } \wr C\int_I = true \text{ and } r \in \mathbb{R}_{\geq 0} \text{ and } m = 1 \\ 1 & \text{if } x = y \text{ and } I = I_1^* \oplus(x,\Delta)^{r'} I_2^* \text{ and } \Gamma \notin \mathsf{sorts}(I_1^* I_2^*) \\ & \text{and } \wr C\int_I = true \text{ and } r = \infty \text{ and } m = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$Int_\Delta(\mathsf{Nil}) = 0$$
$$Int_\Delta(B_1 \parallel B_2) = Int_\Delta(B_1) + Int_\Delta(B_2)$$
$$Int_\Delta(I[\,P\,]_n) = \begin{cases} 1 & \text{if } \exists x.\ \oplus(x,\Delta)^r \in I \\ 0 & \text{otherwise} \end{cases}$$

$$Sel_{(\Delta,\Gamma)}(\mathsf{Nil}) = 0$$
$$Sel_{(\Delta,\Gamma)}(B_1 \parallel B_2) = Sel_{(\Delta,\Gamma)}(B_1) + Sel_{(\Delta,\Gamma)}(B_2)$$
$$Sel_{(\Delta,\Gamma)}(I[\,P\,]_n) = \begin{cases} 1 & \text{if } \exists x.\ \oplus(x,\Delta)^r \in I \\ & \text{and } \exists y.\ \oplus(y,\Gamma)^{r'} \in I \\ 0 & \text{otherwise} \end{cases}$$

Table 6.3: Counting functions.

Function $Ch_x^m(I,P)$ is defined in Tab.6.3. A similar procedure is used to compute the immediate apparent probability and the stochastic apparent rate for intra-communications. These values can be computed by inspecting the structure of all the boxes composing S and finding all the combinations of inputs and outputs that can synchronize inside each of the boxes:

$$R^m_{intra}(\mathsf{Nil}) = 0$$
$$R^m_{intra}(B_0 \parallel B_1) = R^m_{intra}(B_0) + R^m_{intra}(B_1)$$
$$R^m_{intra}(I[P]_n) = \sum_{x \in \mathsf{fn}(P)} \begin{cases} r \times (In^I_x(P) \times Out^{I,0}_x(P) - Mix^I_x(P)) & \text{if } m = 1 \text{ and } r = rate_I(x) \in \mathbb{R}_{\geq 0} \\ In^I_x(P) \times Out^{I,0}_x(P) - Mix^I_x(P) & \text{if } m = 0 \text{ and } rate_I(x) = \infty \\ 0 & \text{otherwise} \end{cases}$$

The functions $In^I_x(P)$ and $Out^{I,k}_x(P)$ are defined in Tab.6.3 and are here invoked with $k = 0$. They return the number of inputs and outputs on channel x in P guarded only by conditions that evaluate to *true*, respectively, while $Mix^I_x(P)$ returns the sum of $In^I_x(M_i) \times Out^{I,0}_x(M_i)$ for each choice M_i in P. The definition of apparent rate, hence, takes into account the fact that an input and an output in the same choice cannot generate an intra-communication, by subtracting $Mix^I_x(P)$ from the product of the number of inputs and outputs on x. In the definition of these functions we use an utility function $\mathsf{InSub}(\pi)$ that returns $\{x\}$ if $\pi = x?y$ and returns $\emptyset$ otherwise; similarly, we use an utility function $\mathsf{OutSub}^I(\pi)$ that returns $\{x\}$ if $\pi = x!y$ and $y \notin \mathsf{sub}(I)$ and returns $\emptyset$ otherwise; obviously, for intra-communications the function $\mathsf{OutSub}^I(\pi)$ is invoked with $I = \epsilon$.

Given a system $\hat{S} = (B, E, \xi)$, the immediate apparent probability and the stochastic apparent rate for complexations, denoted with $R^0_{bind}(B, \wp_2(\mathcal{T}))$ and $R^1_{bind}(B, \wp_2(\mathcal{T}))$, respectively, are computed by the following function [1]:

$$R^m_{bind}(B, \emptyset) = 0$$
$$R^m_{bind}(B, \{\{\Delta, \Gamma\}\} \cup T') =$$
$$R^m_{bind}(B, T') + \begin{cases} \alpha^s_b(\Delta,\Gamma) \times (Int_\Delta(B) \times Int_\Gamma(B) - Sel_{(\Delta,\Gamma)}(B)) \\ \qquad \text{if } \alpha^s_b(\Delta,\Gamma) \in \mathbb{R}_{\geq 0} \text{ and } m = 1 \\ Int_\Delta(B) \times Int_\Gamma(B) - Sel_{(\Delta,\Gamma)}(B) \\ \qquad \text{if } \alpha^s_b(\Delta,\Gamma) = \infty \text{ and } m = 0 \\ 0 \quad \text{otherwise} \end{cases}$$

The function $Int_\Delta(B)$, defined in Tab.6.3, returns the number of free interfaces with sort Δ in B, while $Sel_{(\Delta,\Gamma)}(P)$ returns the number of boxes that have either an interface with sort Δ and an interface with sort Γ. The definition, hence, takes into account the fact that a box cannot complex with himself.

Given a system $\hat{S} = (B, E, \xi)$, the immediate apparent probability and the stochastic

[1] with $\wp_2(\mathcal{T})$ we denote the set $\{\{\Delta, \Gamma\} \mid \{\Delta, \Gamma\} \in \wp(\mathcal{T})\}$ of subsets in the power set $\wp(\mathcal{T})$ with cardinality 2.

apparent rate for unbindings, denoted with $R^0_{unbind}(B,\xi)$ and $R^1_{unbind}(B,\xi)$, respectively, are computed by the following function:

$$
\begin{aligned}
&R^m_{unbind}(B,\emptyset) = 0\\
&R^m_{unbind}(B,\{\{\Delta n,\Gamma m\}\}\cup\xi) =\\
&\qquad R^m_{unbind}(B,\xi) + \begin{cases} \alpha^s_u(\Delta,\Gamma) & \text{if } \alpha^s_u(\Delta,\Gamma)\in\mathbb{R}_{\geq 0} \text{ and } m=1\\ 1 & \text{if } \alpha^s_u(\Delta,\Gamma)=\infty \text{ and } m=0\\ 0 & \text{otherwise}\end{cases}
\end{aligned}
$$

Given a system $\hat{S}=(B,E,\xi)$, the immediate apparent probability and the stochastic apparent rate for complex-communications, denoted with $R^0_{complex}(B,\xi)$ and $R^1_{complex}(B,\xi)$, respectively, are computed by the following function:

$$
\begin{aligned}
&R^m_{complex}(B,\emptyset) = 0\\
&R^m_{complex}(B,\{\{\Delta n,\Gamma m\}\}\cup\xi) =\\
&\qquad R^m_{complex}(B,\xi) \; + \begin{cases} \alpha^s_c(\Delta,\Gamma)\times(In_{\Delta n}(B)\times Out_{\Gamma m}(B) \; + Out_{\Delta n}(B)\times In_{\Gamma m}(B)) \\ \qquad \text{if } \alpha^s_c(\Delta,\Gamma)\in\mathbb{R}_{\geq 0} \text{ and } m=1\\ In_{\Delta n}(B)\times Out_{\Gamma m}(B) \; + Out_{\Delta n}(B)\times In_{\Gamma m}(B)\\ \qquad \text{if } \alpha^s_c(\Delta,\Gamma)=\infty \text{ and } m=0\\ 0 \quad \text{otherwise}\end{cases}
\end{aligned}
$$

Note that for each link the number of possible complex-communications depends on the number of combinations of inputs and outputs that the processes of the two involved boxes have enabled. Given an interface Γ and a box identifier m, the functions $Out_{\Gamma m}(B)$ and $In_{\Gamma m}(B)$, defined in Tab.6.3, return respectively the number of outputs and inputs associated with the interface with sort Γ in the box with identifier m.

Given a system $\hat{S}$, the immediate apparent probability and the stochastic apparent rate for inter-communications, denoted with $R^0_{inter}(B,\mathcal{T}^2)$ and $R^1_{inter}(B,\mathcal{T}^2)$, respectively, are computed by the following function:

$$
\begin{aligned}
&R^m_{inter}(B,\emptyset) = 0\\
&R^m_{inter}(B,\{(\Delta,\Gamma)\}\cup T') =\\
&\qquad R^m_{inter}(B,T') + \begin{cases} \alpha^s_c(\Delta,\Gamma)\times(In_{\Delta}(B)\times Out_{\Gamma}(B) - Mix_{(\Delta,\Gamma)}(B)) \\ \qquad \text{if } \alpha^s_b(\Delta,\Gamma)=\alpha^s_u(\Delta,\Gamma)=0 \text{ and } \alpha^s_c(\Delta,\Gamma)\in\mathbb{R}_{\geq 0} \text{ and } m=1\\ \wr\alpha^s_c(\Delta,\Gamma)\int_S \times COMB \\ \qquad \text{if } \alpha^s_b(\Delta,\Gamma)=\alpha^s_u(\Delta,\Gamma)=0 \text{ and } \alpha^s_c(\Delta,\Gamma)\in\mathcal{F} \text{ and } m=1\\ (In_{\Delta}(B)\times Out_{\Gamma}(B) - Mix_{(\Delta,\Gamma)}(B)) \\ \qquad \text{if } \alpha^s_b(\Delta,\Gamma)=\alpha^s_u(\Delta,\Gamma)=0 \text{ and } \alpha^s_c(\Delta,\Gamma)=\infty \text{ and } m=0\\ 0 \quad \text{otherwise}\end{cases}
\end{aligned}
$$

Given a sort Δ, the functions $In_\Delta(B)$ and $Out_\Delta(B)$, defined in Tab.6.3, return the number of inputs and outputs enabled on free interfaces of boxes in B and guarded only by conditions that evaluate to true, respectively, while $Mix_{(\Delta,\Gamma)}(B)$ returns the sum of $In_\Delta(I[\,P\,]_n) \times Out_\Gamma(I[\,P\,]_n)$ for each box in B. Note that when $\alpha_c^s(\Delta,\Gamma) \in \mathcal{F}$ we have that the associated apparent rate is not calculated by exploiting all the possible input and output combinations. When $\Delta \neq \Gamma$, the value $COMB$ corresponds to all the possible combinations (X_1, X_2) of species in $(Boxes(B)/\sim_s)$ such that $X_1 \neq X_2$ and X_1 has at least an output on an interface with sort Δ and X_2 has at least an input on an interface with sort Γ; When $\Delta = \Gamma$, instead, $COMB$ corresponds to the number of species X in $(Boxes(B)/\sim_s)$ with cardinality of at least 2 and having at least an input and output on an interface with sort Δ.

Finally, given a system $\hat{S} = (B, E, \xi)$, the immediate apparent probability and the stochastic apparent rate for events, denoted with $R^0_{event}(\hat{S}, B, \xi)$ and $R^1_{event}(\hat{S}, B, \xi)$, respectively, are computed with the following procedure:

$$R^m_{event}(\hat{S},\mathsf{Nil},\xi) = 0 \qquad R^m_{event}(\hat{S}, E_0 \parallel E_1, \xi) = R^m_{event}(\hat{S}, E_0, \xi) + R^m_{event}(\hat{S}, E_1, \xi)$$

$$R^m_{event}(\hat{S}, B_1 \rhd_h B_2 \parallel E, \xi) = \begin{cases} \lbag h \rbag_{\hat{S}} & \text{if } h \in \mathcal{F} \wedge (B,\xi) \equiv_c (B_1 \parallel B', \xi') \text{ and } m = 1 \\ 1 & \text{if } h = \infty \wedge (B,\xi) \equiv_c (B_1 \parallel B', \xi') \text{ and } m = 0 \\ 0 \quad \text{otherwise} & \end{cases}$$

In all the cases, the stochastic interpretation of events does not rely on the law of mass action. Events can be, indeed, only immediate or associated with functions describing generic kinetic laws. Note that functions can be used to recover mass action kinetics for events. Now, we can define formally the notions of immediate apparent probability and stochastic apparent rate for a system.

Definition 6.1.6. *Given a system $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}_s$, the immediate apparent rate of $\hat{S}$, denoted with $R^0(\hat{S})$, is computed by the following formula:*

$$\begin{aligned} R^0(\hat{S}) = \; & R^0_{change}(B) + R^0_{intra}(B) + R^0_{bind}(B, \wp_2(\mathcal{T})) + R^0_{unbind}(B,\xi) + \\ & R^0_{complex}(B,\xi) + R^0_{inter}(B, \mathcal{T}^2) \; + R^0_{event}(\hat{S}, E, \xi) \end{aligned}$$

Moreover, the stochastic apparent rate of S, denoted with $R^1(\hat{S})$, is computed by the following formula:

$$\begin{aligned} R^1(\hat{S}) = \; & R^1_{change}(B) + R^1_{intra}(B) + R^1_{bind}(B, \wp_2(\mathcal{T})) + R^1_{unbind}(B,\xi) \; + \\ & R^1_{complex}(B,\xi) + R^1_{inter}(B, \mathcal{T}^2) + R^1_{event}(\hat{S}, E, \xi) \end{aligned}$$

Definition 6.1.7. *Let $\hat{S} \in \overline{\mathcal{S}}_s$. The probability of performing the reaction $\hat{S} \rightarrow_g S'$ is given by $g/R^0(\hat{S})$ if $g = \infty$, is given by $g/R^1(\hat{S})$ if $g \in \mathbb{R}_{\geq 0}$ and is given by $\lbag g \rbag_{\hat{S}} / R^1(\hat{S})$*

if $g \in \mathcal{F}$.

Proposition 6.1.8. *Let* $\hat{S}, \hat{S}' \in \overline{\mathcal{S}}_s$. $\hat{S} \equiv \hat{S}'$ *implies* $R^m(\hat{S}) = R^m(\hat{S}')$ *with* $m \in \{0, 1\}$.

Proof. By induction on the length of the derivation $\hat{S} \equiv \hat{S}'$. □

6.2 A stochastic abstract machine

Having a stochastic interpretation of BlenX, we want now to simulate sBlenX systems by means of the Gillespie algorithm. The specification of sBlenX systems, however, relies on an individual-based interpretation, where each single substance is represented by a box. The interpretation of different substances as single and distinct boxes, is not the most efficient representation on which to implement the Gillespie approach. The one-to-one correspondence between substances and boxes causes, indeed, an explosion of boxes due to their populations. In other words, we will have many copies of the same box to represent the instances of the same substance species. To overcome the multiple copies problem, it makes sense to instantiate objects that represent species and to maintain for each of them the information about its population. The same can be done for complexes. This is the basic principle on which we ground the definition of the BlenX stochastic abstract machine. The main idea is to define an abstract machine relying on a species-based interpretation, where sBlenX systems are compressed by maintaining classes of structurally congruent boxes and classes of structurally congruent(isomorphic) complexes.

We start by defining formally the abstract machine and proving its correctness with respect to the sBlenX semantics. Then we show how the structure of the machine allows for an efficient implementation of the Gillespie algorithm. A sBlenX system is simulated by first encoding it into a corresponding term of a BlenX machine. Machine terms consist of a set of species (ranged over by $\mathfrak{S}$, $\mathfrak{S}'$, $\cdots$), a set of complexes (ranged over by $\mathfrak{C}$, $\mathfrak{C}'$, $\cdots$) and a set of reactions (ranged over by $\mathfrak{R}$, $\mathfrak{R}'$, $\cdots$). With $\mathbf{S}$, $\mathbf{C}$ and $\mathbf{R}$ we identify all the possible set of species, complexes and reactions, respectively.

Formally, a machine term $\mathfrak{M}$ is a triple $\mathfrak{M} = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$ and with $\mathbf{M}$ we identify all the possible machine terms. A set $\mathfrak{S}$ contains elements of the form (i, j, I, P), where $i \in \mathbb{N}$ represents a species identifier, $j \in \mathbb{N}$ represents the population of the species, and I and P correspond to the interfaces and the internal process of the box identifying the species, called also *box species*. The box species is the representative of the species, hence representing all the boxes $I'[P']_n$ such that $I[P] \sim_s I'[P']$.

A set $\mathfrak{C}$ contains graph specifications of sBlenX complexes in the form (i, j, V, E), where i represents the complex identifier, j the population of the complex and where V and E represent the nodes and the edges of the complex, respectively. Set V contains elements

of the form (i, k), where i is a node identifier and k is a species identifier. Set E, instead, contains edges of the form $\{\Delta i_1, \Gamma i_2\}$ where i_1 and i_2 are node identifiers and Δ and Γ are sorts. The main difference between links and edges is that names are substituted with natural numbers. However, the theory developed for links can be simply adapted for edges. In particular, the relation $\sim_c$ (defined in Def.4.3.22) can be used also in this context. These graph specifications of sBlenX complexes are called *machine complexes*. Given box species with identifiers 1, 2, 3 and 4, a machine complex is represented graphically as depicted in Fig. 6.1. $\mathbf{G}$ denotes the set of all the possible machine complexes.

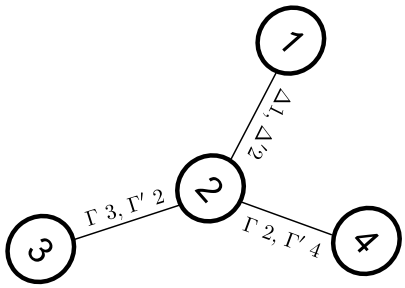


Figure 6.1: Visual representation of a machine complex. Node labels represent the identifiers of the nodes. Edges labels contain the information about the binding; for example, the edge $\Gamma 3, \Gamma' 2$ says that the species associated with the node with identifier 3, and the species associated with the node with identifier 2, are complexed through their interfaces of sorts Γ and Γ', respectively.

A set $\mathfrak{R}$ contains reactions, which can have one up-to four different forms. Each reaction, indeed, can describe:

- an intra-communication if it has the the form (i, r, x, in, out, mix), where i is the species identifier, $r \in \mathbb{R}_{>0} \cup \{\infty\}$ is the rate, $x \in \mathcal{N}$ is the communication channel and and $in, out, mix \in \mathbb{N}$ allow to identify all the combinations of inputs and outputs on that channel;

- a change if it has the form $(i, x, change, type)$, where i is the species identifier, $x \in \mathcal{N}$ is the subject of the change action, $change \in \mathbb{R}_{>0}$ is the sum of the rates of all the change actions on x that the species i can perform and $type \in \{ch, chinf\}$;

- a bind, unbind, complex-communication or inter-communication if it has the form $(i_1, i_2, g, \Delta_1, \Delta_2, k, type)$ where i_1 and i_2 are species identifiers, $g \in \mathbb{R}_{>0} \cup \{\infty\} \cup \mathcal{F}$ is the rate constant, Δ_1 and Δ_2 are sorts, k identifies the number of instances of the reaction and $type \in \{inter, complex, bind, unbind\}$;

- an event if it has the form (i, N, h, N'), where i is the event identifier, N, N' are multisets of elements in $\mathcal{N}$ representing the identifiers of the box species involved in

$$
\mathfrak{S} \boxplus_s^m (I,P) = \begin{cases} (\mathfrak{S} \cup \{(k = |\mathfrak{S}|+1, m, I, P)\}, \{(k, m, I, P)\}, k, \mathfrak{R}) \\ \quad \text{if } \nexists (i,j,I',P') \in \mathfrak{S} \text{ s.t. } I[P] \sim_s I'[P'] \\ (\mathfrak{S} \setminus \{(i,j,I',P')\} \cup \{(i, j+m, I', P')\}, \emptyset, i, \emptyset) \\ \quad \text{if } \exists (i,j,I',P') \in \mathfrak{S} \text{ s.t. } I[P] \sim_s I'[P'] \end{cases}
$$

where $\mathfrak{R} =$

$$
\bigcup_{x \in \mathsf{fn}(P)} \{(k, rate_I(x), x, In_x^I(P), Out_x^{I,0}(P), Mix_x^I(P)) \mid rate_I(x) \neq 0 \text{ and } In_x^I(P) \times Out_x^{I,0}(P) \times Mix_x^I(P) > 0\} \cup
$$

$$
\bigcup_{x \in \mathsf{sub}(I)} (\{(k, x, Ch_x^1(I,P), ch) \mid Ch_x^1(I,P) > 0\} \cup \{(k, x, Ch_x^0(I,P), chinf) \mid Ch_x^0(I,P) > 0\})
$$

$$
\mathfrak{S} \boxminus_s (I,P) = \begin{cases} \mathfrak{S} \setminus \{(i,j,I',P')\} \cup \{(i, j-1, I', P')\} \\ \quad \text{if } \exists (i,j,I',P') \in \mathfrak{S} \text{ s.t. } j > 0 \text{ and } I[P] \sim_s I'[P'] \\ \bot \quad \text{otherwise} \end{cases}
$$

$$
\mathfrak{S} \overline{\boxplus}_s (\{\!| (I,P) |\!\} \cup X) = \mathfrak{S}_2 \text{ where } (\mathfrak{S}_1, \mathfrak{S}'', k, \mathfrak{R}_1) = \mathfrak{S} \boxplus_s^1 (I,P) \text{ and } (\mathfrak{S}_2, \mathfrak{R}_2) = \mathfrak{S}_1 \overline{\boxplus}_s X
$$

$$
\mathfrak{S} \overline{\boxplus}_s \emptyset = \mathfrak{S}
$$

$$
\mathfrak{C} \boxplus_c (V,E) = \begin{cases} \mathfrak{C} \cup \{(|\mathfrak{C}|+1, 1, V, E)\} \\ \quad \text{if } \nexists (i,j,V',E') \in \mathfrak{C} \text{ s.t. } gBox(V,E) \approx gBox(V',E') \\ \mathfrak{C} \setminus \{(i,j,V',E')\} \cup \{(i, j+1, V', E')\} \\ \quad \text{if } \exists (i,j,V',E') \in \mathfrak{C} \text{ s.t. } gBox(V,E) \approx gBox(V',E') \end{cases}
$$

$$
\mathfrak{C} \boxminus_c (V,E) = \begin{cases} \mathfrak{C} \setminus \{(i,j,V',E')\} \cup \{(i, j-1, V', E')\} \\ \quad \text{if } \exists (i,j,V',E') \in \mathfrak{C} \text{ s.t. } j > 0 \text{ and } gBox(V,E) \approx gBox(V',E') \\ \bot \quad \text{otherwise} \end{cases}
$$

Table 6.4: Operators for adding and subtracting species and complexes from sets of species and complexes.

the event substitution[2] and $h \in \{\infty\} \cup \mathcal{F}$ is the rate constant.

Given a well-formed sBlenX system S, the corresponding BlenX machine $\mathfrak{M} = [\![\hat{S}]\!]$ is obtained by applying an encoding $[\![\cdot]\!]$ on $\hat{S}$. Before presenting formally the encoding $[\![\cdot]\!]$, we first introduce some operators we will use in the rest of the chapter.

Tab. 6.4 defines operators for adding and subtracting species and complexes from sets of species and complexes, respectively. Given a set $\mathfrak{S}$ and a pair (I,P), the operator $\boxplus_s^m : \mathbf{S} \times (\mathcal{I} \times \mathcal{P}) \to \mathbf{S} \times \mathbf{S} \times \mathbb{N} \times \mathbf{R}$ searches a pair (I',P') in $\mathfrak{S}$ such that $I[P] \sim_s I'[P']$.

If the search is successful, the corresponding species population is increased by m

[2]Formally, a multiset over a set A is a map $N : A \to \mathbb{N}$, where $N(a)$ denotes the multiplicity of the element $a \in A$ in the multiset N. We use also the explicit notation $N = \{\!| a, a, b, a, b, c, a, c |\!\}$.

and the quadruple $(\mathfrak{S}', \emptyset, k, \emptyset)$ is returned; $\mathfrak{S}'$ is the updated set of species and k the identifier of the species that has been modified. Otherwise, the quadruple $(\mathfrak{S} \cup \{(k = |\mathfrak{S}| + 1, m, I, P)\}, \{(k, m, I, P)\}, k, \mathfrak{R})$ is returned; the first element in the quadruple is the species set with the new species added; the second element is a set containing only the new species; the value k is the new species identifier; the last set $\mathfrak{R}$ is a set containing all the monomolecular actions that the species can perform. The set $\mathfrak{R}$ is constructed by first adding quadruples $(k, rate_I(x), x, In_x^I(P), Out_x^{I,0}(P), Mix_x^I(P))$ representing intra-communications, where x is a free name in P, $rate_I(x)$ is the rate associated with the communication and the last three values allow to infer the total number of possible intra-communications over x that can happen in P. The set $\mathfrak{R}$ is then updated with quadruples $(k, x, Ch_x^1(P), ch)$ and $(k, x, Ch_x^0(P), chinf)$ representing change actions, where x is a subject of an interface in I and values $Ch_x^1(P)$ and $Ch_x^0(P)$ give information about the propensity of the reaction. In the quadruples of both intra-communications and change actions, the number k represents the identifier of the species that can perform the corresponding actions.

Operator $\boxminus_s : \mathbf{S} \times (\mathcal{I} \times \mathcal{P}) \to \mathbf{S} \cup \{\perp\}$ is used to decrease by one the population of a species that is already present. Indeed, the operation $\mathfrak{S} \boxminus_s (I, P)$ is defined only when in $\mathfrak{S}$ exists a pair (I', P') such that $I[P] \sim_s I'[P']$ and such that its population counter j is greater than zero.

Operator $\overline{\boxplus}_s$ generalises operator $\boxplus_s^m$ by adding a multiset of pairs (I, P). It is based on $\boxplus_s^1$ and returns an updated set of species; no reactions are returned, because, as we will see, the operator will be used in a context in which we are sure that no new species are added.

Operators for sets of machine complexes are similar. Operator $\boxplus_c : \mathbf{C} \times \mathbf{G} \to \mathbf{C}$ allows to add a machine complex (V, E) to a set of machine complexes. As before, if the machine complex is already present, then its population counter is increased by one, while if it is not present, then a new machine complex is generated. Note that the presence of a machine complex is verified by using the isomorphism relation $\approx$ on complexes introduced in Def. 4.3.24. In particular, given two machine complexes, the check is done by transforming the machine complexes into corresponding sBlenX complexes, through a function $gBox$. As an example, given a machine complex:

$$(V, E) = (\{(1, i_1), (2, i_2), (3, i_3)\}, \{\{\Delta_1 1, \Delta_2 2\}, \{\Delta_2' 2, \Delta_3 3\}\})$$

the function $gBox(B, \xi)$ generates the corresponding sBlenX complex:

$$(I_1[\,P_1\,]_{x_1} \parallel I_1[\,P_2\,]_{x_2} \parallel I_2[\,P_3\,]_{x_3}, \{\{\Delta_1 x_1, \Delta_2 x_2\}, \{\Delta_2' x_2, \Delta_3 x_3\}\})$$

$$
\begin{aligned}
\mathfrak{S}_0 \bowtie_b \mathfrak{S}_1 &= \{(i_k, i_{1-k}, r, \Delta_k, \Delta_{1-k}, 1, bind) \mid (i_0, j_0, I_0, P_0) \in \mathfrak{S}_0 \wedge (i_1, j_1, I_1, P_1) \in \mathfrak{S}_1 \wedge \\
&\quad \oplus(x_0, \Delta_0)^{r_0} \in I_0 \wedge \oplus(x_1, \Delta_1)^{r_1} \in I_1 \wedge r = \alpha_b^s(\Delta_0, \Delta_1) \neq 0 \wedge \\
&\quad ((k = 0 \text{ if } i_0 \leq i_1) \vee (k = 1 \text{ if } i_1 < i_0))\} \\
\mathfrak{S}_0 \bowtie_u^{\mathfrak{C}} \mathfrak{S}_1 &= \{(i_k, i_m, r, \Delta_k, \Delta_m, 1, unbind) \mid (i_0, j_0, I_0, P_0) \in \mathfrak{S}_0 \wedge (i_1, j_1, I_1, P_1) \in \mathfrak{S}_1 \wedge \\
&\quad \otimes(x_0, \Delta_0)^{r_0} \in I_0 \wedge \otimes(x_1, \Delta_1)^{r_1} \in I_1 \wedge ((k = 0 \text{ if } i_0 \leq i_1) \vee (k = 1 \text{ if } i_1 < i_0)) \wedge \\
&\quad g = \alpha_u^s(\Delta_0, \Delta_1) \neq 0 \wedge \exists(\{(l_0, i_0), (l_1, i_1)\} \cup N, \{\{l_0\Delta_0, l_1\Delta_1\}\} \cup V) \in \mathfrak{C}\} \\
\mathfrak{S}_0 \bowtie_c^{\mathfrak{C}} \mathfrak{S}_1 &= \{(i_0, i_1, r, \Delta_0, \Delta_1, k, complex) \mid (i_0, j_0, I_0, P_0) \in \mathfrak{S}_0 \wedge (i_1, j_1, I_1, P_1) \in \mathfrak{S}_1 \wedge \\
&\quad \otimes(x_0, \Delta_0)^{r_0} \in I_0 \wedge \otimes(x_1, \Delta_1)^{r_1} \in I_1 \wedge k = Out_{x_1}^{I_1,1}(P_1) \times In_{x_2}^{I_2}(P_2) > 0 \\
&\quad r = \alpha_c^s(\Delta_0, \Delta_1) \neq 0 \wedge \exists(\{(l_0, i_0), (l_1, i_1)\} \cup N, \{\{l_0\Delta_0, l_1\Delta_1\}\} \cup V) \in \mathfrak{C}\} \\
\mathfrak{S}_0 \bowtie_i \mathfrak{S}_1 &= \{(i_0, i_1, g, \Delta_0, \Delta_1, k, inter) \mid (i_0, j_0, I_0, P_0) \in \mathfrak{S}_0 \wedge (i_1, j_1, I_1, P_1) \in \mathfrak{S}_1 \wedge \\
&\quad \oplus(x_0, \Delta_0)^{r_0} \in I_0 \wedge \oplus(x_1, \Delta_1)^{r_1} \in I_1 \wedge \alpha_b^s(\Delta_0, \Delta_1) = \alpha_u^s(\Delta_0, \Delta_1) = 0 \wedge \\
&\quad g = \alpha_c^s(\Delta_0, \Delta_1) \neq 0 \wedge k = Out_{x_1}^{I_1,1}(P_1) \times In_{x_2}^{I_2}(P_2) > 0\}
\end{aligned}
$$

Table 6.5: Operators for the generation of all the reactions that pairs of species can perform.

where $(i_1, j_1, I_1, P_1), (i_2, j_2, I_2, P_2), (i_3, j_3, I_3, P_3) \in \mathfrak{S}$ and $x \in \mathcal{N}$. Note that the uniqueness of node identifiers guarantees the well-formedness of the sBlenX complex.

Operator $\boxminus_c : \mathbf{C} \times \mathbf{G} \to \mathbf{C} \cup \{\perp\}$ is used to decrease the population of a machine complex. The operation $\mathfrak{C} \boxminus_c (V, E)$ is defined only when in $\mathfrak{C}$ there exists a pair (V', E') such that $gBox(V, E)$ is isomorphic to $gBox(V', E')$ and such that its population counter j is greater than zero.

Tab. 6.5 introduces some operators that allow the generation of sets containing, respectively, all the bimolecular actions that two sets of species $\mathfrak{S}_0$ and $\mathfrak{S}_1$ can perform together. Given two sets of species $\mathfrak{S}_0$ and $\mathfrak{S}_1$, the operator $\bowtie_b$: $\mathbf{S} \times \mathbf{S} \to \mathbf{R}$ generates a set containing tuples $(i_0, i_1, r, \Delta_0, \Delta_1, 1, bind)$, representing bind reactions. In particular, for each combination of species in the two sets, the operator adds a tuple for each pair of interfaces, of the two species, that are not in bound state and that have associated sorts that can bind with a compatibility greater than zero. Since the intersection of the two species set can be different from $\emptyset$, then by ordering the tuple elements with respect to identifier values, we avoid the generation of multiple copies of the same reaction.

Operator $\bowtie_u$: $\mathbf{S} \times \mathbf{C} \times \mathbf{S} \to \mathbf{R}$ is similar and generates a set containing tuples representing unbind reactions. In particular, for each combination of species in the two sets, the operator adds a tuple for each pair of interfaces, of the two species, that are in bound state (and represent actually a link) and that have associated sorts that can unbind with a compatibility greater than zero. Also in this case by ordering the tuple elements with

respect to identifier values we avoid the generation of multiple copies of the same reaction.

Operator $\bowtie_c$: $\mathbf{S} \times \mathbf{C} \times \mathbf{S} \to \mathbf{R}$ generates a set containing tuples that represent complex-communications. For each combination of species in the two sets, the operator adds a tuple for each pair of interfaces, of the two species, that are in bound state (and represent actually a link) and that have associated sorts that can communicate with a compatibility greater than zero. Moreover, the value k counts all the possible combinations of outputs and inputs of the first and the second species, respectively; we require k to be greater than zero. Note that in this way we implicitly establish an ordering in the tuple elements, where we know that the first identifier always refer to the species that executes the output.

Operator $\bowtie_i$: $\mathbf{S} \times \mathbf{S} \to \mathbf{R}$ generates a set with tuples representing inter-communications. For each combination of species in the two sets, the operator adds a tuple for each pair of interfaces, of the two species, that are not in bound state and that have associated sorts that can communicate with a compatibility greater than 0. Since we are talking about inter-communications we have to be sure that the corresponding sorts have value 0 associated with the binding and unbinding compatibilities. Moreover, also in this case the value k counts all the possible combinations of outputs and inputs of the first and the second species, respectively, which is required to be greater than zero.

The formal definition of the encoding function $[\![\cdot]\!]$, that produces a BlenX machine term from a well-formed sBlenX system in safe normal form, is given in Tab. 6.6. The encoding $[\![\cdot]\!]$ uses an auxiliary encoding $[\![\cdot]\!]^1_i$ for bio-processes, auxiliary encodings $[\![\cdot]\!]^2$ and $[\![\cdot]\!]^3$ for complexes and an auxiliary encoding $[\![\cdot]\!]^4_i$ for events.

Encoding $[\![\cdot]\!]^1_i$ takes a bio-process B in safe normal form and a set of species $\mathfrak{S}$ and returns an updated set of species, a mapping μ from names to natural numbers and a set of reactions representing all the monomolecular reactions that all the species in B can perform. The mapping μ is a sequence of mappings $[n \mapsto i]$ from names to natural numbers. With $\mu[y \mapsto i]$ we extend a mapping on the right; instances on the right overwrite instances on the left; the empty mapping is denoted by $\emptyset$. Consider for example the bio-process:

$$B = \oplus(x, \Delta)^{r_1}[\,\mathsf{ch}(x, \Gamma, s).\mathsf{nil}\,]_{n_1} \parallel I_2[\,x?z.\mathsf{nil} \mid x!y.\mathsf{nil}\,]_{n_2} \parallel I_2[\,y?z.\mathsf{nil} \mid y!y.\mathsf{nil}\,]_{n_3}$$

By applying the encoding $[\![\cdot]\!]^1_1$ on B and an initial empty set of species we obtain:

$$[\![B, \emptyset]\!]^1_1 =$$
$$(\{(1, 1, I_2, y?z.\mathsf{nil} \mid y!y.\mathsf{nil}), (2, 1, I_2, x?z.\mathsf{nil} \mid x!y.\mathsf{nil}), (3, 1, \oplus(x, \Delta)^{r_1}, \mathsf{ch}(x, \Gamma, s).\mathsf{nil})\},$$
$$[n_3 \mapsto 1][n_2 \mapsto 2][n_1 \mapsto 3], \{(3, x, s, ch), (2, \delta^s(x), x, 1, 1, 0), (1, \delta^s(y), y, 1, 1, 0)\})$$

where we assume $\delta^s(x)$ and $\delta^s(y)$ greater than 0. Note that since B is well-formed, we

$$
\begin{array}{rcl}
[\![(B,E,\xi)]\!] & = & (\mathfrak{S},\mathfrak{C},\mathfrak{R}\cup\mathfrak{R}'\cup(\mathfrak{S}\bowtie_b\mathfrak{S})\cup(\mathfrak{S}\bowtie_i\mathfrak{S})\cup(\mathfrak{S}\bowtie_u^{\mathfrak{C}}\mathfrak{S})\cup(\mathfrak{S}\bowtie_c^{\mathfrak{C}}\mathfrak{S})) \\
 & & \text{where } (\mathfrak{S}',\mu,\mathfrak{R}')=[\![B,\emptyset]\!]_1^1 \text{ and } \mathfrak{C}=[\![(\xi/\sim_c),\mu]\!]^2 \\
 & & \text{and } (\mathfrak{S},\mathfrak{R})=[\![E,\mathfrak{S}']\!]_1^4 \\
 & & \\
[\![\mathsf{Nil},\mathfrak{S}]\!]_i^1 & = & (\emptyset,\emptyset,\emptyset) \\
[\![I[\,P\,]_y \parallel B,\mathfrak{S}]\!]_i^1 & = & (\mathfrak{S}'',\mu'[y\mapsto k],\mathfrak{R}'\cup\mathfrak{R}'') \\
 & & \text{where } (\mathfrak{S}',\mu',\mathfrak{R}')=[\![B,\mathfrak{S}]\!]_i^1 \text{ and } (\mathfrak{S}'',k,\mathfrak{R}'')=\mathfrak{S}'\boxplus_s^i(I,P) \\
 & & \\
[\![\emptyset,\mu]\!]^2 & = & \emptyset \\
[\![O\cup\{o\},\mu]\!]^2 & = & [\![O,\mu]\!]^2\boxplus_c[\![o,\mu]\!]^3 \\
 & & \\
[\![\emptyset,\mu]\!]^3 & = & (\emptyset,\emptyset) \\
[\![\xi\cup\{\Delta_1 n_1,\Delta_2 n_2\},\mu]\!]^3 & = & (V\cup\{(k_1,\mu(n_1)),(k_2,\mu(n_2))\},E\cup\{\{\Delta_1 k_1,\Delta_2 k_2\}\}) \\
 & & \text{where } (V,E)=[\![\xi,\mu]\!]^3 \text{ and } k_1=|V|+1 \text{ and } k_2=|V|+2 \text{ and} \\
 & & \\
[\![\mathsf{Nil},\mathfrak{S}]\!]_i^4 & = & (\emptyset,\emptyset) \\
[\![B_0\rhd_h B_1\parallel E,\mathfrak{S}]\!]_i^4 & = & (\mathfrak{S}_3,\mathfrak{R}\cup\mathfrak{R}_1\cup\mathfrak{R}_2\cup\mathfrak{R}_3) \text{ where } (\mathfrak{S}_3,\mathfrak{R}_3)=[\![E,\mathfrak{S}_2]\!]_{i+1}^4 \\
 & & \text{and } (\mathfrak{S}_1,\mu_1,\mathfrak{R}_1)=[\![B_0,\mathfrak{S}]\!]_0^1 \text{ and } (\mathfrak{S}_2,\mu_2,\mathfrak{R}_2)=[\![B_1,\mathfrak{S}_1]\!]_0^1 \\
 & & \text{and } \mathfrak{R}=\{(i,\{\!|\,\mu_1(n)\mid n\in\mathsf{id}(B_0)\,|\!\},h,\{\!|\,\mu_2(n)\mid n\in\mathsf{id}(B_1)\,|\!\})\}
\end{array}
$$

Table 6.6: Encoding from sBlenX systems into BlenX machine terms.

are sure that in the mapping we have no overriding of associations. Indeed, all the boxes identifiers are different.

Encoding $[\![\cdot]\!]^3$ takes an environment ξ and a mapping μ and returns a machine complex (V,E). From the information in the links composing ξ a set of nodes V and a set of edges E are created. In particular, in creating nodes (i,k), node identifiers i are created by enumerating them starting from 1, while species identifiers are obtained with the help of the mapping. Consider for example the environment:

$$\xi=\{\{\Delta_1 n_1,\Delta_2 n_2\},\{\Delta_1' n_1,\Delta_2 n_2\},\{\Delta_2' n_2,\Delta_3 n_3\},\{\Delta_3' n_3,\Delta_1' n_1\}\}$$

and the mapping:

$$\mu=[n_1\mapsto 1][n_2\mapsto 2][n_3\mapsto 3]$$

By applying the encoding $[\![\cdot]\!]^3$ on the elements ξ and μ we can obtain the following tuple of elements:

$$[\![\xi,\mu]\!]^3=(\{(1,1),(2,3),(3,2)\},\{\{\Delta_1 1,\Delta_2 3\},\{\Delta_1' 1,\Delta_2 3\},\{\Delta_2' 3,\Delta_3 2\},\{\Delta_3' 2,\Delta_1' 1\}\})$$

Note that the encoding works only if the mapping μ is defined over all the names contained in ξ. Well-formedness of sBlenX systems guarantees the consistency of $[\![\cdot]\!]^3$ application.

Encoding $[\![\cdot]\!]^2$ takes a set of environments O and a mapping μ and returns a set of machine complexes. In detail, encoding $[\![\cdot]\!]^2$ applies the encoding $[\![\cdot]\!]^3$ on each element of O to generate a machine complex; using the operator $\boxplus_c$, all the generated machine complexes are then combined to create the overall set of machine complexes.

Encoding $[\![\cdot]\!]^4_i$ takes a composition of events E in safe normal form and a set of species $\mathfrak{S}$, and returns an updated set of species and a set of reactions. Given an event $B_0 \rhd_h B_1 \parallel E$, its encoding first uses encoding $[\![B_0, \mathfrak{S}]\!]^1_0$ to add all the species defined in B_0 to the set of species $\mathfrak{S}$, obtaining an updated set of species $\mathfrak{S}_1$; the operation returns also a mapping μ_1 and the set $\mathfrak{R}_1$ of monomolecular reactions that the new species in B_0 can perform. The resulting $\mathfrak{S}_1$ is then used in $[\![B_1, \mathfrak{S}_1]\!]^1_0$ to repeat the previous operations on the second part of the event. The operation, indeed, returns a set of species $\mathfrak{S}_2$, a mapping μ_2 and a set of monomolecular reactions $\mathfrak{R}_2$. Note that by using $[\![\cdot]\!]^1_k$, with $k = 0$, we guarantee that all the species we add have population zero or does not change the population of already present species. This is essential, because species in events do not affect the number of species present in the initial configuration of the system. Mappings μ_1 and μ_2 are then used to generate the reaction $\{(i, \{\!|\, \mu_1(n) \mid n \in \mathsf{id}(B_0) \,|\!\}, h, \{\!|\, \mu_2(n) \mid n \in \mathsf{id}(B_1) \,|\!\})\}$, representing the event. For each box in B_0, the first multiset in the triple contains the corresponding species identifiers. The last element in the tuple is, instead, the multiset regarding the bio-process B_1. We use multisets because more instances of the same species identifier can be present. Finally, the encoding is inductively applied on $[\![E, \mathfrak{S}_2]\!]^4_{i+1}$, obtaining the last sets $\mathfrak{S}_3$ and $\mathfrak{R}_3$ which are combined with all the other sets to obtain the returning pair $(\mathfrak{S}_3, \mathfrak{R} \cup \mathfrak{R}_1 \cup \mathfrak{R}_2 \cup \mathfrak{R}_3)$. By starting from 1 and propagating a value for event identifiers that is increased by one in each recursive invocation, we finally have different identifiers for all the events. An example of event encoding follows:

$$E = \oplus(x, \Delta)^{r_1}[\,\mathsf{ch}(x, \Gamma, s).\mathsf{nil}\,]_{n_1} \rhd_h I_2[\,x?z.\mathsf{nil} \mid x!y.\mathsf{nil}\,]_{n_2}$$

By applying the encoding $[\![\cdot]\!]^4_1$ on the elements E and the empty set of species we obtain the following pair of elements:

$$[\![E, \emptyset]\!]^4_1 = (\{(1, 0, \oplus(x, \Delta)^{r_1}, \mathsf{ch}(x, \Gamma, r).\mathsf{nil}), (2, 0, I_2, x?z.\mathsf{nil} \mid x!y.\mathsf{nil})\},$$
$$\{(1, x, s, ch), (2, \delta^s(x), x, 1, 1, 0), (1, \{\!|\, 1 \,|\!\}, h, \{\!|\, 2 \,|\!\})\})$$

All the presented auxiliary encodings are used by $[\![\cdot]\!]$ to generate a BlenX machine term from a sBlenX system in safe normal form. Given a system $\hat{S} = (B, E, \xi)$, the encoding

first uses $[\![B, \emptyset]\!]^1_1$ to obtain a set $\mathfrak{S}'$ containing all the species defined in B, a mapping μ from box identifiers of B to species identifiers in $\mathfrak{S}'$ and a set $\mathfrak{R}'$ containing all the monomolecular actions that all the species in B can perform. The mapping μ is then used in $[\![(\xi/\sim_c), \mu]\!]^2$ to generate all the machine complexes corresponding to the complexes specified by ξ; indeed, $(\xi/\sim_c)$ returns a partition of ξ such that each of its elements represent a different sBlenX complex. Moreover, since the encoded system is well-formed, all the names in ξ have a corresponding mapping in μ and hence $[\![(\xi/\sim_c), \mu]\!]^2$ returns a valid set $\mathfrak{C}$ of machine complexes. Then, the sBlenX system encoding uses $[\![E, \mathfrak{S}']\!]^4_1$ to update $\mathfrak{S}'$ with all the species defined in the events composing E, generating $\mathfrak{S}$, and to collect all the monomolecular reactions that the new species in E can perform and all the event reactions, generating the set $\mathfrak{R}$. Given the set of species $\mathfrak{S}$, the set of machine complexes $\mathfrak{C}$ and the set of all the monomolecular reaction and events $\mathfrak{R} \cup \mathfrak{R}'$, the last step is to add to the set of all the bimolecular reactions. This is done by adding to $\mathfrak{R} \cup \mathfrak{R}'$ all the sets generated by $\mathfrak{S} \bowtie_b \mathfrak{S}$, $\mathfrak{S} \bowtie^{\mathfrak{C}}_u \mathfrak{S}$, $\mathfrak{S} \bowtie^{\mathfrak{C}}_c \mathfrak{S}$ and $\mathfrak{S} \bowtie_i \mathfrak{S}$. As an example, consider the sBlenX system $\hat{S} = (B, E, \xi)$ with:

$$\begin{aligned}
B &= \otimes(x, \Delta)^r[\, x?y.\mathsf{nil} + x?z.\mathsf{nil} \mid x!z.\mathsf{nil}\,]_{n_1} \parallel \otimes(y, \Gamma)^{r'}[\, \mathsf{nil} \mid y!z.\mathsf{nil}\,]_{n_2} \parallel \\
&\quad \oplus(x, \Delta_1)^{r''}[\, x?z.\mathsf{nil} \mid x!y.\mathsf{nil}\,]_{n_3} \parallel \oplus(x, \Delta_1)^{r''}[\, x!y.\mathsf{nil} \mid x?z.\mathsf{nil}\,]_{n_4} \\
E &= I_1[\, \mathsf{nil}\,]_{n_1} \rhd_h \oplus(x, \Delta_1)^{r''}[\, x!y.\mathsf{nil}\,]_{n_2} \\
\xi &= \{\{\Delta n_1, \Gamma n_2\}\}
\end{aligned}$$

where we assume $\alpha^s(\Delta, \Gamma) = (1.5, 1.5, 1.5)$ and $\alpha^s(\Delta_1, \Delta_1) = (0, 0, 2.5)$; all the others combinations are not compatible. By applying the encoding $[\![\hat{S}]\!]$ we obtain the following BlenX machine term:

$$\begin{aligned}
[\![(B, E, \xi)]\!] = \; & (\{(1, 2, \oplus(x, \Delta_1)^{r''}, x!y.\mathsf{nil} \mid x?z.\mathsf{nil}), (4, 0, I_1, \mathsf{nil}), (5, 0, \oplus(x, \Delta_1)^{r''}, x!y.\mathsf{nil}), \\
& \quad (3, 1, \otimes(x, \Delta)^r, x?y.\mathsf{nil} + x?z.\mathsf{nil} \mid x!z.\mathsf{nil})\}, (2, 1, \otimes(y, \Gamma)^{r'}, \mathsf{nil} \mid y!z.\mathsf{nil}) \\
& \{(1, 1, \{(1, 3), (2, 2)\}, \{\{\Delta 1, \Gamma 2\}\})\} \\
& \{(3, x, \delta^s(x), 2, 1, 0), (1, x, \delta^s(x), 1, 1, 0), (1, \{\!|\ 4\ |\!\}, h, \{\!|\ 5), \\
& \quad (3, 2, 1.5, \Delta, \Gamma, 1, \mathit{unbind}), (2, 3, 1.5, \Delta, \Gamma, 2, \mathit{complex}), \\
& \quad (1, 1, 2.5, \Delta_1, \Delta_1, 2, \mathit{inter}), (5, 1, 2.5, \Delta_1, \Delta_1, 1, \mathit{inter})\})
\end{aligned}$$

There are 5 species, where the one with identifier 1 has population equal to 2 and the ones belonging to events have population 0. Then there is a complex composed by two nodes and one edge and a set of reactions containing two intra-communications, one events, an unbind, a complex-communication and two inter-communications.

Given a machine term obtained from a well-formed sBlenX system in safe normal form,

its dynamics is described by a series of rules that we introduce in the following. Each rule describes how a machine term is modified to reflect the execution of one of its reactions.

Consider a term $\mathfrak{M} = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$ and suppose $(i, r, x, in, out, mix) \in \mathfrak{R}$. The reaction refers to an intra-communication on channel x that the species with identifier i can perform. If the identifier i refers to a species that is part of at least one machine complex, then the dynamics of the intra-communication is described by the following rule:

$$\frac{\begin{array}{c}(i, r, x, in, out, mix) \in \mathfrak{R} \quad (i, j, I, P) \in \mathfrak{S} \\ P \equiv_p \langle C_1 \rangle x!z.Q_1 + M_1 \mid \langle C_2 \rangle x?y.Q_2 + M_2 \mid Q_3 \quad \wr C_1 \int_I = \wr C_2 \int_I = true \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I, P) \quad (\mathfrak{S}_2, \mathfrak{S}_3, k, \mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I, Q_1 \mid Q_2\{z/y\} \mid Q_3) \\ \mathfrak{C}_1 = (\mathfrak{C} \boxminus_c (\{(l, i)\} \cup V, E)) \boxplus_c (\{(l, k)\} \cup V, E)\end{array}}{\mathfrak{M} \mapsto_r (\mathfrak{S}_2, \mathfrak{C}_1, \mathfrak{R} \cup \mathfrak{R}' \cup (\bigcup \mathfrak{S}_2 \bowtie^{\mathfrak{C}_1} \mathfrak{S}_3))} \quad \text{(m1)}$$

where with $\bigcup \mathfrak{S}_2 \bowtie^{\mathfrak{C}_1} \mathfrak{S}_3$ we mean $\mathfrak{S}_2 \bowtie_b \mathfrak{S}_3 \cup \mathfrak{S}_2 \bowtie_i \mathfrak{S}_3 \cup \mathfrak{S}_2 \bowtie_u^{\mathfrak{C}_1} \mathfrak{S}_3 \cup \mathfrak{S}_2 \bowtie_c^{\mathfrak{C}_1} \mathfrak{S}_3$. The rule can be applied for all the combinations of elements Q_1, M_1, Q_2, M_2, Q_3, C_1, C_2, l, V and E that make the rule premises valid. Given the species $(i, j, I, P) \in \mathfrak{S}$, the rule states that for an enabled combination of input/output on x, expressed as $P \equiv_p \langle C_1 \rangle x!z.Q_1 + M_1 \mid \langle C_2 \rangle x?y.Q_2 + M_2 \mid Q_3$ with conditions evaluating to $true$, we first decrease the population of the species (I, P) in $\mathfrak{S}$, obtaining $\mathfrak{S}_1$, and then we add to $\mathfrak{S}_1$ the species $(I, Q_1 \mid Q_2\{z/y\} \mid Q_3)$, generated by executing the intra-communication, obtaining $\mathfrak{S}_2$. Moreover, we decrease the population of a complex $(\{(l, i)\} \cup V, E)$ containing a node associated with species i, and add a new complex $(\{(l, k)\} \cup V, E)$ in which that node is updated by substituting the association to i with an association to k, that is the identifier of the new species. Finally, if $\mathfrak{S}_3$ and $\mathfrak{R}'$ are not $\emptyset$, this means that k is actually a new species and hence we have to add $\mathfrak{R}'$ to the set of reactions and generate all the new bimolecular reactions that $\mathfrak{S}_3$ can perform with all the other species in $\mathfrak{S}_2$. Note that if the species i is not part of a complex, then the operation $\mathfrak{C} \boxminus_c (\{(l, i)\} \cup V, E)$ is not defined (returns $\perp$) and hence the rule cannot be applied. Moreover, note that the reaction is associated with the rate r contained in the selected tuple. More about the quantitative aspects will be explained later.

If the identifier i refers to a species that is not part of any machine complex, then the dynamics of the intra-communication is described by the rule:

$$\frac{\begin{array}{c}(i, r, x, in, out, mix) \in \mathfrak{R} \quad (i, j, I, P) \in \mathfrak{S} \quad \nexists (\{(l, i)\} \cup V, E) \in \mathfrak{C} \\ P \equiv_p \langle C_1 \rangle x!z.Q_1 + M_1 \mid \langle C_2 \rangle x?y.Q_2 + M_2 \mid Q_3 \quad \wr C_1 \int_I = \wr C_2 \int_I = true \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I, P) \quad (\mathfrak{S}_2, \mathfrak{S}_3, k, \mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I, Q_1 \mid Q_2\{z/y\} \mid Q_3)\end{array}}{\mathfrak{M} \mapsto_r (\mathfrak{S}_2, \mathfrak{C}, \mathfrak{R} \cup \mathfrak{R}' \cup (\bigcup \mathfrak{S}_2 \bowtie^{\mathfrak{C}} \mathfrak{S}_3))} \quad \text{(m2)}$$

the explanation of which is similar to the previous one. The main difference is that no modifications in the set of complexes are taken.

Now consider a reaction $(i, x, change, type) \in \mathfrak{R}$. The reaction refers to a change on x that the species with identifier i can perform. If the identifier i refers to a species that is part of at least one machine complex, then the dynamics of the reaction is described by the rule:

$$\frac{\begin{array}{c}(i,x,change,type) \in \mathfrak{R} \quad (i,j,I,P) \in \mathfrak{S} \\ P \equiv_p \langle C \rangle \mathsf{ch}(x,\Gamma,r).Q_1 + M_1 \mid Q_2 \quad \wr\, C \int_I = true \quad I = I_1^* \oplus (x,\Delta)^{r'} I_2^* \quad \Gamma \notin \mathsf{sorts}(I_1^* I_2^*) \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I,P) \quad (\mathfrak{S}_2, \mathfrak{S}_3, k, \mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I_1^* \oplus (x,\Gamma)^{r'} I_2^*, Q_1 \mid Q_2) \\ \mathfrak{C}_1 = (\mathfrak{C} \boxminus_c (\{(l,i)\} \cup V, \{\{\Delta l, \Delta' l'\}\} \cup E)) \boxplus_c (\{(l,k)\} \cup V, \{\{\Gamma l, \Delta' l'\}\} \cup E)\end{array}}{\mathfrak{M} \mapsto_r (\mathfrak{S}_2, \mathfrak{C}_1, \mathfrak{R} \cup \mathfrak{R}' \cup (\bigcup \mathfrak{S}_2 \bowtie^{\mathfrak{C}_1} \mathfrak{S}_3))} \quad \text{(m3)}$$

Given the species $(i, j, I, P) \in \mathfrak{S}$, the rule states that for an enabled instance of change on x in P, expressed as $P \equiv_p \langle C \rangle \mathsf{ch}(x, \Gamma, r).Q_1 + M_1 \mid Q_2$ with condition evaluating to $true$ and such that $\Gamma \notin \mathsf{sorts}(I_1^* I_2^*)$, we first decrease the population of the species (I, P) in $\mathfrak{S}$, obtaining $\mathfrak{S}_1$, and then we add to $\mathfrak{S}_1$ the species $(I_1^* \oplus (x,\Gamma)^{r'} I_2^*, Q_1 \mid Q_2)$, generated by executing the change, obtaining $\mathfrak{S}_2$. Moreover, as for intra-communications, we decrease the population of a complex $(\{(l,i)\} \cup V, \{\{\Delta l, \Delta' l'\}\} \cup E)$ containing a node associated with species i, and add a new complex $(\{(l,k)\} \cup V, \{\{\Gamma l, \Delta' l'\}\} \cup E)$ in which that node is updated by substituting the association to i with an association to k, that is the identifier of the new species; note moreover, that the corresponding edge is modified accordingly. Also in this case, if $\mathfrak{S}_3$ and $\mathfrak{R}'$ are not $\emptyset$, this means that k is actually a new species and hence we have to add $\mathfrak{R}'$ to the set of reactions and generate all the new bimolecular reactions that $\mathfrak{S}_3$ can perform with all the other species in $\mathfrak{S}_2$.

If the identifier i refers to a species that is not part of any machine complex, then the dynamics of the intra-communication is described by the rule:

$$\frac{\begin{array}{c}(i,x,change,type) \in \mathfrak{R} \quad (i,j,I,P) \in \mathfrak{S} \quad \nexists (\{(l,i)\} \cup V, E) \in \mathfrak{C} \\ P \equiv_p \langle C \rangle \mathsf{ch}(x,\Gamma,r).Q_1 + M_1 \mid Q_3 \quad \wr\, C \int_I = true \quad I = I_1^* \oplus (x,\Delta)^{r'} I_2^* \quad \Gamma \notin \mathsf{sorts}(I_1^* I_2^*) \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I,P) \quad (\mathfrak{S}_2, \mathfrak{S}_3, k, \mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I_1^* \oplus (x,\Gamma)^{r'} I_2^*, Q_1 \mid Q_3)\end{array}}{\mathfrak{M} \mapsto_r (\mathfrak{S}_2, \mathfrak{C}, \mathfrak{R} \cup \mathfrak{R}' \cup (\bigcup \mathfrak{S}_2 \bowtie^{\mathfrak{C}} \mathfrak{S}_3))} \quad \text{(m4)}$$

Now, suppose that a binding reaction $(i_1, i_2, r, \Delta_1, \Delta_2, k, bind)$ is selected. We can distinguish among five different cases: the species are part of different complexes (m5); the species are part of the same complex (m6); only one of the two species is part of at least one complex (m7-8); species i_1 and i_2 are not part of complexes (m9). We start from

case (m5), the dynamics of which is described by the following rule:

$$\frac{\begin{array}{c}(i_1,i_2,r,\Delta_1,\Delta_2,1,bind)\in\mathfrak{R}\quad (i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad (i_2,j_2,I_2,P_2)\in\mathfrak{S}\\ \mathfrak{S}_1=\mathfrak{S}\boxminus_s(I_1,P_1)\boxminus_s(I_2,P_2)\\ I_1=I_{11}^*\oplus(x_1,\Delta_1)^{r_1}\,I_{12}^*\quad (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1(I_{11}^*\otimes(x_1,\Delta_1)^{r_1}\,I_{12}^*,P_1)\\ I_2=I_{21}^*\oplus(x_2,\Delta_2)^{r_2}\,I_{22}^*\quad (\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1(I_{21}^*\otimes(x_2,\Delta_2)^{r_2}\,I_{22}^*,P_2)\\ \mathfrak{C}_1=(\mathfrak{C}\boxminus_c(V'=\{(l_1,i_1)\}\cup V_1,E_1))\boxminus_c(\{(l_2,i_2)\}\cup V_2,E_2)\end{array}}{\mathfrak{M}\mapsto_r(\mathfrak{S}_3,\mathfrak{C}'=\mathfrak{C}_1\boxplus_c(V,E),\mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}'}(\mathfrak{S}'\cup\mathfrak{S}')))}\ \text{(m5)}$$

where,

$$V=\{(l_1,k_1)\}\cup V_1\cup\{(l+|V'|,i)\mid(l,i)\in\{(l_2,k_2)\}\cup V_2\}$$
$$E=E_1\cup\{\{\Delta_1 l_1,\Delta_2 l_2+|V'|\}\}\cup\{\{\Delta_1(l_1+|V'|),\Delta_2(l_2+|V'|)\}\mid\{\Delta_1 l_1,\Delta_2 l_2\}\in E_2\}$$

Given species (i_1,j_1,I_1,P_1) and (i_2,j_2,I_2,P_2) in $\mathfrak{S}$, the rule states that first we decrease the population of the species (I_1,P_1) and (I_2,P_2) in $\mathfrak{S}$, obtaining $\mathfrak{S}_1$. Then we add species $(I_{11}^*\otimes(x_1,\Delta_1)^{r_1}\,I_{12}^*,P_1)$ to $\mathfrak{S}_1$, obtaining $\mathfrak{S}_2$, and add species $(I_{21}^*\otimes(x_2,\Delta_2)^{r_2}\,I_{22}^*,P_2)$ to $\mathfrak{S}_2$, obtaining $\mathfrak{S}_3$. Note that the two species we add are obtained from (I_1,P_1) and (I_2,P_2) by rendering complexed the state of the two interfaces involved in the binding. Then, given two complexes that contain nodes associated with species i_1 and i_2, we decrease their population and use them to create a new complex that represents the result of the binding. Note that to maintain a consistent graph representation we have first to update nodes and edges of the second complex by adding to all the node identifiers a value corresponding to the cardinality of the set of nodes of the first complex. A visual example elucidating the complex creation is given in Fig. 6.2.

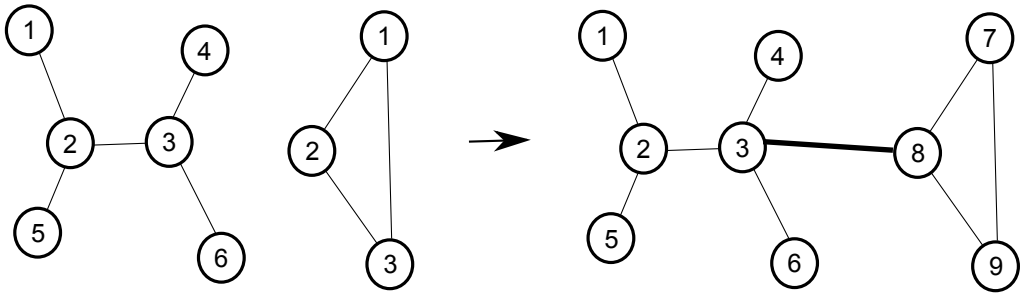


Figure 6.2: Visual representation of the binding of two machine complexes. When the new edge (the bold one on the right) is added, then the node enumeration of the second machine complex is modified to maintain the consistency of the overall nodes enumeration. In particular, to each node identifier we add the cardinality of the first complex.

Now consider the case in which the complexation happens between nodes of the same complex. This case is captured by the following rule:

$$\frac{\begin{array}{c}(i_1, i_2, r, \Delta_1, \Delta_2, 1, bind) \in \mathfrak{R} \quad (i_1, j_1, I_1, P_1) \in \mathfrak{S} \quad (i_2, j_2, I_2, P_2) \in \mathfrak{S} \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I_1, P_1) \boxminus_s (I_2, P_2) \\ I_1 = I_{11}^* \oplus (x_1, \Delta_1)^{r_1} I_{12}^* \quad (\mathfrak{S}_2, \mathfrak{S}', k_1, \mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I_{11}^* \otimes (x_1, \Delta_1)^{r_1} I_{12}^*, P_1) \\ I_2 = I_{21}^* \oplus (x_2, \Delta_2)^{r_2} I_{22}^* \quad (\mathfrak{S}_3, \mathfrak{S}'', k_2, \mathfrak{R}'') = \mathfrak{S}_2 \boxplus_s^1 (I_{21}^* \otimes (x_2, \Delta_2)^{r_2} I_{22}^*, P_2) \\ \mathfrak{C}_1 = \mathfrak{C} \boxminus_c (V = \{(l_1, i_1)\} \cup \{(l_2, i_2)\} \cup V', E)\end{array}}{\mathfrak{M} \mapsto_r (\mathfrak{S}_3, \mathfrak{C}' = \mathfrak{C}_1 \boxplus_c (V, E \cup \{\{\Delta_1 l_1, \Delta_2 l_2)\}\}, \mathfrak{R} \cup \mathfrak{R}' \cup \mathfrak{R}'' \cup (\bigcup \mathfrak{S}_3 \bowtie^{\mathfrak{C}'} (\mathfrak{S}' \cup \mathfrak{S}'')))} \quad \text{(m6)}$$

Note that the rule is similar to the previous one. However, after all the updates on the set of species that lead to $\mathfrak{S}_2$, the operations on the set of complexes are a bit different. In this case, indeed, only the population of a complex containing two nodes associate to species i_1 and i_2, respectively, is decreased. The new complex that is added to the set $\mathfrak{C}_1$ is obtained from the one that is decreased by simply adding a new edge.

The remaining cases differ only with respect to the update of complexes. In rule (m7) only the first species identifier is part of a machine complex, while in rule (m8) only the second species identifier is part of a machine complex. In the last rule (m9), instead, two conditions assures that both species identifiers i_1 and i_2 are not part of any machine complex. All the last tree rule are described by the following rules:

$$\frac{\begin{array}{c}(i_1, i_2, r, \Delta_1, \Delta_2, 1, bind) \in \mathfrak{R} \quad (i_1, j_1, I_1, P_1) \in \mathfrak{S} \quad (i_2, j_2, I_2, P_2) \in \mathfrak{S} \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I_1, P_1) \boxminus_s (I_2, P_2) \\ I_1 = I_{11}^* \oplus (x_1, \Delta_1)^{r_1} I_{12}^* \quad (\mathfrak{S}_2, \mathfrak{S}', k_1, \mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I_{11}^* \otimes (x_1, \Delta_1)^{r_1} I_{12}^*, P_1) \\ I_2 = I_{21}^* \oplus (x_2, \Delta_2)^{r_2} I_{22}^* \quad (\mathfrak{S}_3, \mathfrak{S}'', k_2, \mathfrak{R}'') = \mathfrak{S}_2 \boxplus_s^1 (I_{21}^* \otimes (x_2, \Delta_2)^{r_2} I_{22}^*, P_2) \\ \mathfrak{C}_1 = \mathfrak{C} \boxminus_c (V_1 = \{(l_1, i_1)\} \cup V_1, E_1) \quad \nexists (\{(l_2, i_2)\} \cup V_2, E_2) \in \mathfrak{C}\end{array}}{\begin{array}{r}\mathfrak{M} \mapsto_r (\mathfrak{S}_3, \mathfrak{C}' = \mathfrak{C}_1 \boxplus_c (V_1 \cup \{(|V_1| + 1, i_2)\}, E_1 \cup \{\{\Delta_1 l_1, \Delta_2 (|V_1| + 1)\}\}), \\ \mathfrak{R} \cup \mathfrak{R}' \cup \mathfrak{R}'' \cup (\bigcup \mathfrak{S}_3 \bowtie^{\mathfrak{C}'} (\mathfrak{S}' \cup \mathfrak{S}'')))\end{array}} \quad \text{(m7)}$$

$$\frac{\begin{array}{c}(i_1, i_2, r, \Delta_1, \Delta_2, 1, bind) \in \mathfrak{R} \quad (i_1, j_1, I_1, P_1) \in \mathfrak{S} \quad (i_2, j_2, I_2, P_2) \in \mathfrak{S} \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I_1, P_1) \boxminus_s (I_2, P_2) \\ I_1 = I_{11}^* \oplus (x_1, \Delta_1)^{r_1} I_{12}^* \quad (\mathfrak{S}_2, \mathfrak{S}', k_1, \mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I_{11}^* \otimes (x_1, \Delta_1)^{r_1} I_{12}^*, P_1) \\ I_2 = I_{21}^* \oplus (x_2, \Delta_2)^{r_2} I_{22}^* \quad (\mathfrak{S}_3, \mathfrak{S}'', k_2, \mathfrak{R}'') = \mathfrak{S}_2 \boxplus_s^1 (I_{21}^* \otimes (x_2, \Delta_2)^{r_2} I_{22}^*, P_2) \\ \nexists (\{(l_1, i_1)\} \cup V_1, E_1) \in \mathfrak{C} \quad \mathfrak{C}_2 = \mathfrak{C} \boxminus_c (V_2 = \{(l_2, i_2)\} \cup V', E_2)\end{array}}{\begin{array}{r}\mathfrak{M} \mapsto_r (\mathfrak{S}_3, \mathfrak{C}' = \mathfrak{C}_2 \boxplus_c (V_2 \cup \{(|V_2| + 1, i_1)\}, E_2 \cup \{\{\Delta_1 (|V_2| + 1), \Delta_2 l_2\}\}), \\ \mathfrak{R} \cup \mathfrak{R}' \cup \mathfrak{R}'' \cup (\bigcup \mathfrak{S}_3 \bowtie^{\mathfrak{C}'} (\mathfrak{S}' \cup \mathfrak{S}'')))\end{array}} \quad \text{(m8)}$$

$$\frac{\begin{array}{c}(i_1,i_2,r,\Delta_1,\Delta_2,1,\mathit{bind})\in\mathfrak{R}\quad (i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad (i_2,j_2,I_2,P_2)\in\mathfrak{S}\\ \mathfrak{S}_1=\mathfrak{S}\boxminus_s (I_1,P_1)\boxminus_s (I_2,P_2)\\ I_1=I_{11}^*\oplus(x_1,\Delta_1)^{r_1} I_{12}^*\quad (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1 (I_{11}^*\otimes(x_1,\Delta_1)^{r_1} I_{12}^*,P_1)\\ I_2=I_{21}^*\oplus(x_2,\Delta_2)^{r_2} I_{22}^*\quad (\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1 (I_{21}^*\otimes(x_2,\Delta_2)^{r_2} I_{22}^*,P_2)\\ \nexists(\{(l_1,i_1)\}\cup V_1,E_1)\in\mathfrak{C}\quad \nexists(\{(l_2,i_2)\}\cup V_2,E_2)\in\mathfrak{C}\end{array}}{\begin{array}{l}\mathfrak{M}\mapsto_r (\mathfrak{S}_3,\mathfrak{C}'=\mathfrak{C}\boxplus_c (\{(1,i_1),(2,i_2)\},\{\{\Delta_1 1,\Delta_2 2\}\}),\\ \qquad\qquad \mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}'}(\mathfrak{S}'\cup\mathfrak{S}'')))\end{array}}\ \text{(m9)}$$

We move now to the description of an unbind reaction $(i_1,i_2,r,\Delta_1,\Delta_2,1,\mathit{unbind})$. We can distinguish among three different cases: the unbind generates two complexes (m10); the unbind generates one complex (m11); the unbind breaks a machine complex made up of only one edge and hence no complexes are generated (m12). We start by presenting the first case and the rule describing its dynamics:

$$\frac{\begin{array}{c}(i_1,i_2,r,\Delta_1,\Delta_2,1,\mathit{unbind})\in\mathfrak{R}\quad (i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad (i_1,j_2,I_2,P_2)\in\mathfrak{S}\\ \mathfrak{S}_1=\mathfrak{S}\boxminus_s (I_1,P_1)\boxminus_s (I_2,P_2)\\ I_1=I_{11}^*\otimes(x_1,\Delta_1)^{r_1} I_{12}^*\quad (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1 (I_{11}^*\oplus(x_1,\Delta_1)^{r_1} I_{12}^*,P_1)\\ I_2=I_{21}^*\otimes(x_2,\Delta_2)^{r_2} I_{22}^*\quad (\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1 (I_{21}^*\oplus(x_2,\Delta_2)^{r_2} I_{22}^*,P_2)\\ \mathfrak{C}_1=\mathfrak{C}\boxminus_c (V=\{(l_1,i_1),(l_2,i_2)\}\cup V',\{\{\Delta_1 l_1,\Delta_2 l_2\}\}\cup E)\quad (E/\sim_c)=\{E_1',E_2'\}\end{array}}{\mathfrak{M}\mapsto_r (\mathfrak{S}_3,\mathfrak{C}'=\mathfrak{C}_2\boxplus_c (V_1,E_1)\boxplus_c (V_2,E_2),\mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}'}(\mathfrak{S}'\cup\mathfrak{S}'')))}\ \text{(m10)}$$

where,

$$(V_1,E_1)=\Xi(V_1'\{^{(l_1,k_1)}/_{(l_1,i_1)}\}\{^{(l_2,k_2)}/_{(l_2,i_2)}\},E_1')\text{ with }V_1'=\{(l,k)\in V\mid\exists\{\Delta l,\Gamma l'\}\in E_1'\}$$
$$(V_2,E_2)=\Xi(V_1'\{^{(l_1,k_1)}/_{(l_1,i_1)}\}\{^{(l_2,k_2)}/_{(l_2,i_2)}\},E_2')\text{ with }V_2'=\{(l,k)\in V\mid\exists\{\Delta l,\Gamma l'\}\in E_2'\}$$

Given species (i_1,j_1,I_1,P_1) and (i_2,j_2,I_2,P_2) in $\mathfrak{S}$, the rule states that we first decrease the population of the species (I_1,P_1) and (I_2,P_2) in $\mathfrak{S}$, obtaining $\mathfrak{S}_1$. Then we add species $(I_{11}^*\oplus(x_1,\Delta_1)^{r_1} I_{12}^*,P_1)$ to $\mathfrak{S}_1$, obtaining $\mathfrak{S}_2$, and add species $(I_{21}^*\oplus(x_2,\Delta_2)^{r_2} I_{22}^*,P_2)$ to $\mathfrak{S}_2$, obtaining $\mathfrak{S}_3$. Note that the two species we add are obtained from (I_1,P_1) and (I_2,P_2) by rendering not complexed the state of the two interfaces involved in the unbinding. Then, given a machine complex that contains nodes associated with species i_1 and i_2 and an edge $\{\Delta_1 l_1,\Delta_2 l_2\}$, we decrease its population and use the set of its edges E, obtained by deleting the edge $\{\Delta_1 l_1,\Delta_2 l_2\}$, to generate the result of the unbinding. Indeed, by using relation $\sim_c$ we can partition the set of edges E into equivalence classes. Each class represents the edges of a machine complex. In this case we have two classes E_1' and E_2', meaning that two machine complexes are generated by the unbind execution. These two sets of edges are used to reconstruct the two machine complexes resulting

from the unbind. In particular, we first create the set of nodes V_1', that contains all the nodes that are present in at least on edge in E_1', and use it to create the first complex $(V_1'\{^{(l_1,k_1)}/_{(l_1,i_1)}\}\{^{(l_2,k_2)}/_{(l_2,i_2)}\}, E_1')$, where we apply proper substitutions of species identifiers. Since the machine complex is obtained from a greater complex, then the enumeration of its nodes does probably not respect the correct increasing enumeration from value 1. In order to obtain a machine complex with a correct enumeration, we use a function Ξ, that applied to a machine complex returns the same machine complex with nodes enumerated starting from 1. This function can be implemented in several different ways. One is to substitute step by step all the minimal node identifiers with a value that starts from 1 and that is increased by one after each substitution. The same sequence of operations is taken on the second set of edges E_2. The two resulting complexes are finally added to the set of machine complexes $\mathfrak{C}$. A visual representation of the sequence of operations is given in Fig. 6.3.

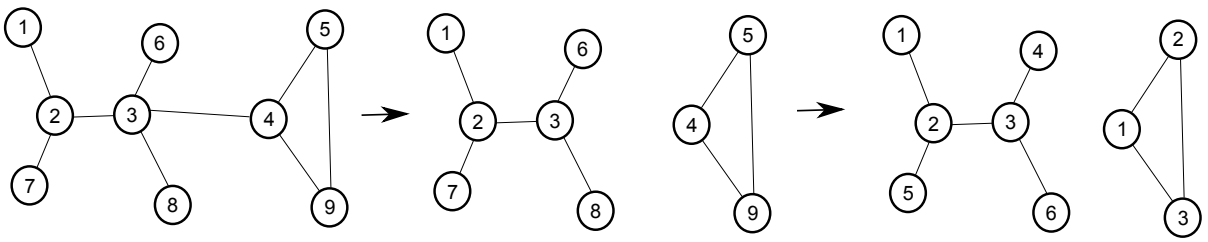


Figure 6.3: Visual representation of the unbinding of two machine complexes. After the unbind we obtain two different machine complexes. The nodes enumeration of the two machine complexes is not correct. Hence, function Ξ is used to obtain the two final machine complexes where the enumeration of the nodes starts from one and is increasing.

The second unbind case occurs when the partitioning of the remaining edges contains only an element, i.e., only one set of edges is generated by partitioning E with $\sim_c$. This means that the unbinding breaks the machine complex in a smaller complex and a single box species. This case is captured by the following rule:

$$\frac{\begin{array}{c}(i_1,i_2,r,\Delta_1,\Delta_2,k,\mathit{unbind})\in\mathfrak{R}\quad(i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad(i_1,j_2,I_2,P_2)\in\mathfrak{S}\\ \mathfrak{S}_1=\mathfrak{S}\boxminus_s(I_1,P_1)\boxminus_s(I_2,P_2)\\ I_1=I_{11}^*\otimes(x_1,\Delta_1)^{r_1}I_{12}^*\quad(\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1(I_{11}^*\oplus(x_1,\Delta_1)^{r_1}I_{12}^*,P_1)\\ I_2=I_{21}^*\otimes(x_2,\Delta_2)^{r_2}I_{22}^*\quad(\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1(I_{21}^*\oplus(x_2,\Delta_2)^{r_2}I_{22}^*,P_2)\\ \mathfrak{C}_1=\mathfrak{C}\boxminus_c(\{(l_1,i_1),(l_2,i_2)\}\cup V,\{\{\Delta_1 l_1,\Delta_2 l_2\}\}\cup E)\quad(E/\sim_c)=\{E\}\end{array}}{\mathfrak{M}\mapsto_r(\mathfrak{S}_3,\mathfrak{C}'=\mathfrak{C}_2\boxplus_c(V_1,E),\mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}'}(\mathfrak{S}'\cup\mathfrak{S}')))}\quad\text{(m11)}$$

where,

$$V_1 = V_1'\{^{(l_1,k_1)}/_{(l_1,i_1)}\}\{^{(l_2,k_2)}/_{(l_2,i_2)}\} \text{ with } V_1' = \{(l,k) \in V \mid \exists\{\Delta l, \Gamma l'\} \in E\}$$

The last unbinding case occurs when the partitioning of the remaining edges generates the empty set. In this case, indeed, we have that the unbinding breaks up a machine complex made of only two box species connected with only one link. The third case is described by the following rule:

$$\frac{\begin{array}{c}(i_1,i_2,r,\Delta_1,\Delta_2,k,\mathit{unbind}) \in \mathfrak{R} \quad (i_1,j_1,I_1,P_1) \in \mathfrak{S} \quad (i_1,j_2,I_2,P_2) \in \mathfrak{S} \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I_1,P_1) \boxminus_s (I_2,P_2) \\ I_1 = I_{11}^* \otimes (x_1,\Delta_1)^{r_1} I_{12}^* \quad (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I_{11}^* \oplus (x_1,\Delta_1)^{r_1} I_{12}^*, P_1) \\ I_2 = I_{21}^* \otimes (x_2,\Delta_2)^{r_2} I_{22}^* \quad (\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'') = \mathfrak{S}_2 \boxplus_s^1 (I_{21}^* \oplus (x_2,\Delta_2)^{r_2} I_{22}^*, P_2) \\ \mathfrak{C}_1 = \mathfrak{C} \boxminus_c (\{(l_1,i_1),(l_2,i_2)\} \cup V, \{\{\Delta_1 l_1, \Delta_2 l_2\}\} \cup E) \quad (E/\sim_c) = \emptyset\end{array}}{\mathfrak{M} \mapsto_r (\mathfrak{S}_3, \mathfrak{C}' = \mathfrak{C}_1, \mathfrak{R} \cup \mathfrak{R}' \cup \mathfrak{R}'' \cup (\bigcup \mathfrak{S}_3 \bowtie^{\mathfrak{C}'} (\mathfrak{S}' \cup \mathfrak{S}')))} \text{ (m11)}$$

Now we introduce the rule that describes the execution of a complex-communication $(i_1,i_2,r,\Delta,\Gamma,k,\mathit{complex})$.

$$\frac{\begin{array}{c}(i_1,i_2,r,\Delta_1,\Delta_2,k,\mathit{complex}) \in \mathfrak{R} \quad (i_1,j_1,I_1,P_1) \in \mathfrak{S} \quad (i_1,j_2,I_2,P_2) \in \mathfrak{S} \\ P_1 \equiv_p \langle C_1\rangle x_1!z.P_1' + M_1 \mid Q_1 \quad P_2 \equiv_p \langle C_2\rangle x_2?y.P_2' + M_2 \mid Q_2 \quad \wr C_1 \int_{I_1} = \wr C_2 \int_{I_2} = \mathit{true} \\ \oplus(x_1,\Delta_1)^{r_1} \in I_1 \quad \oplus(x_2,\Delta_2)^{r_2} \in I_2 \quad z \notin \mathsf{sub}(I_1) \quad \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I_1,P_1) \boxminus_s (I_2,P_2) \\ (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}') = \mathfrak{S}_1 \boxplus_s^1 (I_1,P_1' \mid Q_1) \quad (\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'') = \mathfrak{S}_2 \boxplus_s^1 (I_2,P_2'\{^z/_y\} \mid Q_2) \\ \mathfrak{C}_1 = \mathfrak{C} \boxminus_c (\{(l_1,i_1),(l_2,i_2)\} \cup N, V = \{\{\Delta_1 l_1, \Delta_2 l_2\}\} \cup V')\end{array}}{\mathfrak{M} \mapsto_r (\mathfrak{S}_3, \mathfrak{C}' = \mathfrak{C}_1 \boxplus_c (\{(l_1,k_1),(l_2,k_2)\} \cup N, V), \mathfrak{R} \cup \mathfrak{R}' \cup \mathfrak{R}'' \cup (\bigcup \mathfrak{S}_3 \bowtie^{\mathfrak{C}'} (\mathfrak{S}' \cup \mathfrak{S}')))} \text{ (m12)}$$

Given the species (i_1,j_1,I_1,P_1) and (i_2,j_2,I_2,P_2) in $\mathfrak{S}$, the rule states that for a valid output on x_1 in P_1, expressed as $P_1 \equiv_p \langle C_1\rangle x_1!z.P_1' + M_1 \mid Q_1$ with C_1 evaluating to *true* and side condition $z \notin \mathsf{sub}(I_1)$, and each valid input on x_2 in P_2, expressed as $P_2 \equiv_p \langle C_2\rangle x_2?y.P_2' + M_2 \mid Q_2$ with C_2 evaluating to *true*, we first decrease by one the population of the species (I_1,P_1) and (I_2,P_2) in $\mathfrak{S}$, obtaining $\mathfrak{S}_1$, and then we add to $\mathfrak{S}_1$ the species $(I_1,P_1' \mid Q_1)$ and $(I_2,P_2'\{^z/_y\} \mid Q_2)$, generated by executing the inter-communication, obtaining $\mathfrak{S}_2$. Names x_1 and x_2 are the subjects of the sorts involved in the complex-communication. Note that we assume $z \notin I_2$; this is guaranteed by the fact the systems from which encodings are obtained are in safe normal form. Then, we have to decrease the population of a machine complex $(\{(l_1,i_1),(l_2,i_2)\} \cup N, V = \{\{\Delta_1 l_1, \Delta_2 l_2\}\} \cup V')$. Note that the existence of the machine complex is a prerequisite

for the execution of the action, because a complex-communication can be executed only through existing bindings. If a machine complex of this form exists, then its population is decreased by one and a new machine complex, where a proper substitution of species identifiers on the involved nodes is made, is added to the set of machine complexes.

The set of rules describing the execution of inter-communications $(i_1, i_2, r, \Delta, \Gamma, k, inter)$ is given in the following:

$$\frac{\begin{array}{c}(i_1,i_2,g,\Delta_1,\Delta_2,k,inter)\in\mathfrak{R}\quad(i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad(i_1,j_2,I_2,P_2)\in\mathfrak{S}\\ P_1\equiv_p\langle C_1\rangle x_1!z.P_1'+M_1\mid Q_1\quad P_2\equiv_p\langle C_2\rangle x_2?y.P_2'+M_2\mid Q_2\quad \wr C_1\int_{I_1}=\wr C_2\int_{I_2}=true\\ \oplus(x_1,\Delta_1)^{r_1}\in I_1\quad\oplus(x_2,\Delta_2)^{r_2}\in I_2\quad z\notin \mathsf{sub}(I_1)\quad \mathfrak{S}_1=\mathfrak{S}\boxminus_s(I_1,P_1)\boxminus_s(I_2,P_2)\\ (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1(I_1,P_1'\mid Q_1)\quad(\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1(I_2,P_2'\{z/y\}\mid Q_2)\\ \mathfrak{C}_1=\mathfrak{C}\boxminus_c(\{(l_1,i_1),(l_2,i_2)\}\cup N,V)\end{array}}{\mathfrak{M}\mapsto_g(\mathfrak{S}_3,\mathfrak{C}'=\mathfrak{C}_1\boxplus_c(\{(l_1,k_1),(l_2,k_2)\}\cup N,V),\mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}'}(\mathfrak{S}'\cup\mathfrak{S}'')))}\quad\text{(m13)}$$

$$\frac{\begin{array}{c}(i_1,i_2,g,\Delta_1,\Delta_2,k,inter)\in\mathfrak{R}\quad(i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad(i_1,j_2,I_2,P_2)\in\mathfrak{S}\\ P_1\equiv_p\langle C_1\rangle x_1!z.P_1'+M_1\mid Q_1\quad P_2\equiv_p\langle C_2\rangle x_2?y.P_2'+M_2\mid Q_2\quad \wr C_1\int_{I_1}=\wr C_2\int_{I_2}=true\\ \oplus(x_1,\Delta_1)^{r_1}\in I_1\quad\oplus(x_2,\Delta_2)^{r_2}\in I_2\quad z\notin \mathsf{sub}(I_1)\quad \mathfrak{S}_1=\mathfrak{S}\boxminus_s(I_1,P_1)\boxminus_s(I_2,P_2)\\ (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1(I_1,P_1'\mid Q_1)\quad(\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1(I_2,P_2'\{z/y\}\mid Q_2)\\ \mathfrak{C}_1=\mathfrak{C}\boxminus_c(\{(l_1,i_1)\}\cup N_1,V_1)\quad\mathfrak{C}_2=\mathfrak{C}_1\boxminus_c(\{(l_2,i_2)\}\cup N_2,V_2)\end{array}}{\begin{array}{l}\mathfrak{M}\mapsto_g(\mathfrak{S}_3,\mathfrak{C}'=\mathfrak{C}_2\boxplus_c(\{(l_1,k_1)\}\cup N_1,V_1)\boxplus_c(\{(l_2,k_2)\}\cup N_2,V_2),\\ \qquad\mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}'}(\mathfrak{S}'\cup\mathfrak{S}'')))\end{array}}\quad\text{(m14)}$$

$$\frac{\begin{array}{c}(i_1,i_2,g,\Delta_1,\Delta_2,k,inter)\in\mathfrak{R}\quad(i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad(i_1,j_2,I_2,P_2)\in\mathfrak{S}\\ P_1\equiv_p\langle C_1\rangle x_1!z.P_1'+M_1\mid Q_1\quad P_2\equiv_p\langle C_2\rangle x_2?y.P_2'+M_2\mid Q_2\quad \wr C_1\int_{I_1}=\wr C_2\int_{I_2}=true\\ \oplus(x_1,\Delta_1)^{r_1}\in I_1\quad\oplus(x_2,\Delta_2)^{r_2}\in I_2\quad z\notin \mathsf{sub}(I_1)\quad \mathfrak{S}_1=\mathfrak{S}\boxminus_s(I_1,P_1)\boxminus_s(I_2,P_2)\\ (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1(I_1,P_1'\mid Q_1)\quad(\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1(I_2,P_2'\{z/y\}\mid Q_2)\\ \mathfrak{C}_1=\mathfrak{C}\boxminus_c(\{(l_1,i_1)\}\cup N_1,V_1)\quad\nexists(\{(l,i_2)\}\cup N,V)\in\mathfrak{C}\end{array}}{\mathfrak{M}\mapsto_g(\mathfrak{S}_3,\mathfrak{C}'=\mathfrak{C}_1\boxplus_c(\{(l_1,k_1)\}\cup N_1,V_1),\mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}'}(\mathfrak{S}'\cup\mathfrak{S}'')))}\quad\text{(m15)}$$

$$\frac{\begin{array}{c}(i_1,i_2,g,\Delta_1,\Delta_2,k,inter)\in\mathfrak{R}\quad(i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad(i_1,j_2,I_2,P_2)\in\mathfrak{S}\\ P_1\equiv_p\langle C_1\rangle x_1!z.P_1'+M_1\mid Q_1\quad P_2\equiv_p\langle C_2\rangle x_2?y.P_2'+M_2\mid Q_2\quad \wr C_1\int_{I_1}=\wr C_2\int_{I_2}=true\\ \oplus(x_1,\Delta_1)^{r_1}\in I_1\quad\oplus(x_2,\Delta_2)^{r_2}\in I_2\quad z\notin \mathsf{sub}(I_1)\quad \mathfrak{S}_1=\mathfrak{S}\boxminus_s(I_1,P_1)\boxminus_s(I_2,P_2)\\ (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1(I_1,P_1'\mid Q_1)\quad(\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1(I_2,P_2'\{z/y\}\mid Q_2)\\ \nexists(\{(l,i_1)\}\cup N,V)\in\mathfrak{C}\quad\mathfrak{C}_1=\mathfrak{C}\boxminus_c(\{(l_2,i_2)\}\cup N_2,V_2)\end{array}}{\mathfrak{M}\mapsto_g(\mathfrak{S}_3,\mathfrak{C}'=\mathfrak{C}_1\boxplus_c(\{(l_2,k_2)\}\cup N_2,V_2),\mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}'}(\mathfrak{S}'\cup\mathfrak{S}'')))}\quad\text{(m16)}$$

$$\frac{\begin{array}{c}(i_1,i_2,g,\Delta_1,\Delta_2,k,inter)\in\mathfrak{R}\quad (i_1,j_1,I_1,P_1)\in\mathfrak{S}\quad (i_1,j_2,I_2,P_2)\in\mathfrak{S}\\ P_1\equiv_p \langle C_1\rangle x_1!z.P_1'+M_1\mid Q_1\quad P_2\equiv_p \langle C_2\rangle x_2?y.P_2'+M_2\mid Q_2\quad \wr C_1\int_{I_1}=\wr C_2\int_{I_2}=true\\ \oplus(x_1,\Delta_1)^{r_1}\in I_1\quad \oplus(x_2,\Delta_2)^{r_2}\in I_2\quad z\notin \mathsf{sub}(I_1)\quad \mathfrak{S}_1=\mathfrak{S}\boxminus_s(I_1,P_1)\boxminus_s(I_2,P_2)\\ (\mathfrak{S}_2,\mathfrak{S}',k_1,\mathfrak{R}')=\mathfrak{S}_1\boxplus_s^1(I_1,P_1'\mid Q_1)\quad (\mathfrak{S}_3,\mathfrak{S}'',k_2,\mathfrak{R}'')=\mathfrak{S}_2\boxplus_s^1(I_2,P_2'\{z/y\}\mid Q_2)\\ \nexists(\{(l,i_1)\}\cup N,V)\in\mathfrak{C}\quad \nexists(\{(l,i_2)\}\cup N,V)\in\mathfrak{C}\end{array}}{\mathfrak{M}\mapsto_g(\mathfrak{S}_3,\mathfrak{C},\mathfrak{R}\cup\mathfrak{R}'\cup\mathfrak{R}''\cup(\bigcup\mathfrak{S}_3\bowtie^{\mathfrak{C}}(\mathfrak{S}'\cup\mathfrak{S}')))}\ (\text{m17})$$

Given the species (i_1,j_1,I_1,P_1) and (i_2,j_2,I_2,P_2) in $\mathfrak{S}$, all the rules state that for a valid output on x_1 in P_1, expressed as $P_1\equiv_p\langle C_1\rangle x_1!z.P_1'+M_1\mid Q_1$ with C_1 evaluating to *true* and side condition $z\notin\mathsf{sub}(I_1)$, and each input on x_2 in P_2, expressed as $P_2\equiv_p\langle C_2\rangle x_2?y.P_2'+M_2\mid Q_2$ with C_2 evaluating to *true*, we first decrease by one the population of the species (I_1,P_1) and (I_2,P_2) in $\mathfrak{S}$, obtaining $\mathfrak{S}_1$, and then we add to $\mathfrak{S}_1$ the species $(I_1,P_1'\mid Q_1)$ and $(I_2,P_2'\{z/y\}\mid Q_2)$ generated by executing the inter-communication, obtaining $\mathfrak{S}_2$. Note that x_1 and x_2 are the subjects of the sorts involved in the inter-communication. Also in this case we assume $z\notin I_2$; this is guaranteed by the fact the the systems from which encodings are obtained, are in safe normal form. Moreover, as for bind reactions, all the given rules distinguish between 5 different cases: the communication happens in a single machine complex (m13); the communication happens between two different machine complexes (m14); the communication happens between a single species and a complex (m15-16); the communication happens between two single species (m17).

The final rule we present describes the dynamics of events. Given an event of the form (k,N,h,N'), indeed, it has the form:

$$\frac{\begin{array}{c}(k,\{\!|\, i_1,\cdots,i_k\,|\!\},h,\{\!|\, i_{k+1},\cdots,i_m\,|\!\})\in\mathfrak{R}\quad \{(i_1,j_1,I_1,P_1)\}\cup\cdots\cup\{(i_m,j_m,I_m,P_m)\}\subseteq\mathfrak{S}\\ \mathfrak{S}_1=\mathfrak{S}\boxminus_s(I_1,P_1)\boxminus_s\cdots\boxminus_s(I_k,P_k)\quad \mathfrak{S}_2=\mathfrak{S}_1\overline{\boxplus}_s\{\!|\,(I_{k+1},P_{k+1}),\cdots,(I_m,P_m)\,|\!\}\end{array}}{\mathfrak{M}\mapsto_h(\mathfrak{S}_1,\mathfrak{C},\mathfrak{R})}\ (\text{m18})$$

The populations of all the species with identifiers in N are decreased by one, and the populations of all the species with identifiers in N' are increased by one. Note that the event is defined only if the populations of the species in N is such that the event is enabled, i.e., the population of the species identified by elements in N is enough to allow all the $\boxminus_s$ operations. Moreover, since N and N' are multisets, the population of a species can decrease or increase by values greater than 1, because more instances of the same species identifier can appear both in N and N'. Since event reactions are derived from sBlenX events, then no complexes are involved and hence no modifications on the set of the machine complexes are taken. Moreover, set $\mathfrak{R}$ remains unchanged because we

are sure that no new species are added by executing an event.

6.2.1 Correctness

To prove the correctness of our abstract machine, we follow the approach used in [85]. The correctness, indeed, is expressed in terms of the following properties: soundness, completeness, termination and duration. Soundness ensures that the machine can only perform valid execution steps. Completeness is a much stronger property, which ensures that the machine can execute all possible behaviours of the language. Termination ensures that the machine does not loop forever unnecessarily, and duration ensures that each reduction in the machine takes the same amount of time as the corresponding reduction in the language, and vice-versa.

We start presenting some auxiliary results.

Proposition 6.2.1. *Let $S = (B, E, \xi) \in \overline{\mathcal{S}}$, $[\![\hat{S}]\!] = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$, $I \in \mathcal{I}$ and $P \in \mathcal{P}$. Then $B \equiv_b I_1[P_1]_{n_1} \parallel \cdots \parallel I_k[P_k]_{n_k} \parallel B'$ with $I_i[P_i] \sim_s I[P]$ for all $i = 1, \cdots, k$ and such that $\nexists I_j[P_j]_{n_j} \in B'$ with $I_j[P_j] \sim_s I[P] \iff (i, k, I', P') \in \mathfrak{S}$ with $I[P] \sim_s I'[P']$.*

Proof. By encoding $[\![\cdot]\!]$ definition. □

Definition 6.2.2. *Let $\mathfrak{M}$ and $\mathfrak{M}'$ be machine terms. Then $\mathfrak{M} \doteq \mathfrak{M}'$ iff $\mathfrak{M}' = \mathfrak{M}$ up to renaming of identifiers and substitution of box species.*

When we say substitution of box species we mean that given a species (i, j, I, P), we can substitute I and P with corresponding I' and P' such that $I'[P'] \sim_s I[P]$. Moreover if subjects of interfaces with same sorts in I and I' are different, we have to rename these names also in all the monomolecular reactions referring to them.

Lemma 6.2.3. *Let $S, S' \in \overline{\mathcal{S}}$. Then $S \equiv S' \Rightarrow [\![\hat{S}]\!] \doteq [\![\hat{S}']\!]$.*

Proof Sketch. By definition of $\equiv$ it results that S and S', and hence also $\hat{S}$ and $\hat{S}'$, are rearrangements of the same number of structurally congruent boxes (up-to box identifiers) where links are preserved. This means that for each species (i, j, I, P) in $[\![\hat{S}]\!]$, there exists a species (i', j', I', P') in $[\![\hat{S}']\!]$ such that $I[P] \sim_s I'[P']$, and vice-versa. Moreover, for each machine complex in $[\![\hat{S}]\!]$, there exists a corresponding machine complex in $[\![\hat{S}']\!]$ such that the corresponding BlenX complexes are structurally congruent. Since there exists a correspondence between the set of species and machine complexes of the two encoded system, then the set of bimolecular reactions generated by the operators $\bowtie_b$, $\bowtie_u^{\mathfrak{C}}$, $\bowtie_c^{\mathfrak{C}}$ and $\bowtie_i$ are equal up-to species identifiers; species identifiers, although referring to the same species can be different in the two sets. Moreover, for each species (i, j, I, P) in $[\![\hat{S}]\!]$ and its corresponding species in (i', j', I', P') in $[\![\hat{S}']\!]$, the sets of monomolecular reactions differ

only by identifiers and names referring to interface subjects. Indeed, since by structural congruence definition we can have that corresponding interfaces in I and I' can differ by their subjects, then intra-communications on interface subjects and change actions can differ with respect to the name they refer to.

By Def. 6.2.2 this means $[\![\hat{S}]\!] \doteq [\![\hat{S}']\!]$. □

Lemma 6.2.4. *$\mathfrak{M} \doteq \mathfrak{M}'$ and $\mathfrak{M} \mapsto_g \mathfrak{M}_1 \Rightarrow \exists \mathfrak{M}'_1$ such that $\mathfrak{M}' \mapsto_g \mathfrak{M}'_1$ and $\mathfrak{M}_1 \doteq \mathfrak{M}'_1$.*

Proof Sketch. Since $\mathfrak{M} \doteq \mathfrak{M}'$, for each reaction in $\mathfrak{M}$ that can generate a derivation $\mathfrak{M} \mapsto_g \mathfrak{M}_1$, there exists a corresponding reaction in $\mathfrak{M}'$ that can generate a reaction $\mathfrak{M}' \mapsto_g \mathfrak{M}'_1$, for some $\mathfrak{M}'_1$. Since the identifiers of the reactions in $\mathfrak{M}$ and $\mathfrak{M}'$ refer respectively to identifiers of species (i, j, I, P) and (i', j', I', P') in $\mathfrak{M}$ and $\mathfrak{M}'$ such that $I[P] \sim_s I'[P']$, then each new pairs (I_1, P_1) and (I'_1, P'_1) obtained from (I, P) and (I', P'), by executing $\mathfrak{M} \mapsto_g \mathfrak{M}_1$ and $\mathfrak{M}' \mapsto_g \mathfrak{M}'_1$, are such that $I_1[P_1] \sim_s I'_1[P'_1]$. The same holds for the complexes generated by the reactions. Hence the species, complexes and reactions generated by the two reactions are equal up-to renaming of identifiers and substitution of box species. Since $\mathfrak{M} \doteq \mathfrak{M}'$ and the elements added respect the $\doteq$ definition, then it holds $\mathfrak{M}_1 \doteq \mathfrak{M}'_1$. □

To prove soundness and completeness of the encoding we first introduce a relation $\leq$ between machine terms. Indeed, given the encoding of a sBlenX system, its dynamics continues to add species, machine complexes and reactions, without deleting anything. However, during the computation, many of these species and complexes have population equal to 0 and hence also many reactions are not active. Relation $\leq$ relates machine terms that differ only in the presence of reactions that are not active and cannot be executed.

Definition 6.2.5. *Let $\mathfrak{M} = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$ and $\mathfrak{M}' = (\mathfrak{S} \cup \mathfrak{S}', \mathfrak{C} \cup \mathfrak{C}', \mathfrak{R} \cup \mathfrak{R}')$ be two machine terms. We say that $\mathfrak{M} \leq \mathfrak{M}'$ if and only if it holds that: (a) each species in $\mathfrak{S}'$ has population 0; (b) each complex in $\mathfrak{C}'$ has population 0 and at least one node associated with a species in $\mathfrak{S}'$; (c) for each monomolecular reaction in $\mathfrak{R}'$ the corresponding identifier is in $\mathfrak{S}'$ and each bimolecular reaction or event reaction in $\mathfrak{R}'$ has at least one species identifiers in $\mathfrak{S}'$.*

Lemma 6.2.6. *$\mathfrak{M} \leq \mathfrak{M}_1$ and $\mathfrak{M} \mapsto_g \mathfrak{M}' \Rightarrow \exists \mathfrak{M}'_1$ s.t. $\mathfrak{M}_1 \mapsto_g \mathfrak{M}'_1$ with $\mathfrak{M}' \leq \mathfrak{M}'_1$*

Proof Sketch. It is enough to observe that all the reactions that are enabled in $\mathfrak{M}$ are also enabled in $\mathfrak{M}_1$. □

Lemma 6.2.7. *$\mathfrak{M} \leq \mathfrak{M}_1$ and $\mathfrak{M}_1 \mapsto_g \mathfrak{M}'_1 \Rightarrow \exists \mathfrak{M}'_1$ s.t. $\mathfrak{M} \mapsto_g \mathfrak{M}'$ with $\mathfrak{M}' \leq \mathfrak{M}'_1$*

Proof Sketch. It is enough to observe that all the reactions in $\mathfrak{M}'$ refer at least to a species that has population 0 and hence no rule can be applied. □

Theorem 6.2.8 (Soundness)**.** *$S \to_g S'$ with $g \neq 0 \Rightarrow \exists \mathfrak{M}$ s.t. $[\![\hat{S}]\!] \mapsto_g \mathfrak{M}$ with $[\![\hat{S'}]\!] \leq \doteq \mathfrak{M}$.*

Proof Sketch. By Lemma 4.4.2 we can distinguish among 6 cases, depending on the axiom driving the normalized derivation corresponding to $S \to_g S'$.

(case 1)

We have $S \equiv (K(x,\Delta)^r\, I^*[\,\langle C\rangle \mathsf{ch}(x,\Gamma,r).P + G \,|\, P_1\,]_n \parallel B, E, \xi)$ with $\wr C \int_I = true$. By encoding definition and using Lemma 6.2.3 we have that $[\![\hat{S}]\!]$ contains a species (i, j, I', P') such that $P' \equiv \langle C''\rangle \mathsf{ch}(y,\Gamma,r).P'' + G'' \,|\, P_1''$ with $K(y,\Delta)^r \in I'$. We have two subcases.

If the box with identifier n is not part of any link in ξ then we can apply rule (m2) on $[\![\hat{S}]\!]$, generating a machine $\mathfrak{M}$ where the population of the previous species is decreased by 1 and the pair $(I', P'' \,|\, P_1'')$ is added to the set of species. By encoding definition and Lemma 6.2.3 we have that $[\![\hat{S'}]\!]$ contains a species (i', j', I_2, P_2) such that $I_2[P_2] \sim_s I'[P'' \,|\, P_1'']$. In particular, since $S' \equiv (K(x,\Gamma)^r\, I^*[\,P \,|\, P_1\,]_n \parallel B, E, \xi)$, then the species corresponding to $(I', P'' \,|\, P_1'')$ has the same population both in $\mathfrak{M}'$ and in $[\![\hat{S'}]\!]$. Now, if (i, j, I', P') was such that $j > 1$ then $[\![\hat{S'}]\!]$ has a corresponding species with cardinality $j - 1$, which by encoding definition corresponds to the population of (i, j, I', P') in $\mathfrak{M}'$. In this case we have $\mathfrak{M} \doteq [\![\hat{S'}]\!]$. If instead before the execution $j = 1$ then in $\mathfrak{M}'$ the species (i, j, I', P') has population 0, while in $[\![\hat{S'}]\!]$ this species is not present. This means that from $\mathfrak{M}'$ we can obtain a machine term $\mathfrak{M}''$ such that $[\![\hat{S'}]\!] \leq \mathfrak{M}'' \doteq \mathfrak{M}$, which means $[\![\hat{S'}]\!] \leq \doteq \mathfrak{M}$.

If the box with identifier n is part of a link in ξ then we can apply rule (m1) on $[\![\hat{S}]\!]$, generating as before a machine $\mathfrak{M}'$ where the population of the species (i, j, I', P') is decreased by 1 and the pair $(I', P'' \,|\, P_1'')$ is added to the set of species. Moreover, $[\![\hat{S}]\!]$ contains the machine complex $(\{(l, i)\} \cup V, E)$ corresponding to the sBlenX complex containing the box executing the change action. Its population is decreased by one and a new complex $(\{(l, k)\} \cup V, E')$ is added to the set of complexes, where k is the identifier of the species corresponding to the pair $(I', P'' \,|\, P_1'')$ in $\mathfrak{M}'$. If the population of both the species and the complex before the execution is greater than 1 it results $\mathfrak{M}' \doteq [\![\hat{S'}]\!]$. If instead the population of the species or the complex before the execution is equal to 1, then in $\mathfrak{M}'$ they have population 0 and corresponding species and complexes are not present in $[\![\hat{S'}]\!]$. Hence it results $[\![\hat{S'}]\!] \leq \doteq \mathfrak{M}'$.

All the other cases are similar. □

Corollary 6.2.9. *$S \to_{g_1} S_1 \to_{g_2} S_2 \to_{g_3} \cdots \to_{g_n} S_n$ with $g_i \neq 0$ for all $i = 1, \cdots, n$ $\Rightarrow \exists \mathfrak{M}_1, \cdots, \mathfrak{M}_n$ s.t. $[\![\hat{S}]\!] \mapsto_{g_1} \mathfrak{M}_1 \mapsto_{g_2} \mathfrak{M}_2 \mapsto_{g_3} \cdots \mapsto_{g_n} \mathfrak{M}_n$ and $[\![\hat{S}_i]\!] \leq \doteq \mathfrak{M}_i$ for all $i = 1, \cdots, n$.*

Proof Sketch. It can be proved by using Lemma 6.2.6 and Theorem 6.2.8. □

Theorem 6.2.10 (Completeness)**.** *Let* $S \in \overline{\mathcal{S}}$. $[\![\hat{S}]\!] \mapsto_g \mathfrak{M} \Rightarrow \exists S'$ *such that* $S \rightarrow_g S'$ *and* $[\![\hat{S'}]\!] \leq\doteq \mathfrak{M}$.

Proof Sketch. By cases on the derivation of $[\![\hat{S}]\!] \mapsto_g \mathfrak{M}$.

(case m2)

By hypothesis, the machine term $[\![\hat{S}]\!]$ contains a species (i, j, I, P) such that $P \equiv_p \langle C_1\rangle x?z.Q_1 + M_1 \mid \langle C_2\rangle x!y.Q_2 + M_2 \mid Q_3$ with $\wr C_1 \int_I = \wr C_2 \int_I = true$. Moreover, we know that the population of (i, j, I, P) in $\mathfrak{M}'$ is decreased by one and a pair $(I, Q_1 \mid Q_2\{z/y\} \mid Q_3)$ is added to the set of species. By Prop. 6.2.1 we have that $\hat{S}$ has a bio-process structurally congruent to $I[\,P\,]_n \parallel B$. By applying rules (s2), (s7) and (s8) of Tab. 6.2 we can hence derive $\hat{S} \rightarrow_g S'$ such that the bio-process of S' is equal to $I[\,Q_1 \mid Q_2\{z/y\} \mid Q_3\,]_n \parallel B$. Hence we can also derive $S \rightarrow_g S'$. Note that by Prop. 6.2.1, $[\![\hat{S'}]\!]$ contains all the species defined in B and a species corresponding to the box $I[\,Q_1 \mid Q_2\{z/y\} \mid Q_3\,]_n$, as $[\![\hat{M'}]\!]$. Now, if in B there are other boxes belonging to the same species of $I[\,P\,]_n$, then the species (i, j, I, P) has population equal or greater than 0 in $\mathfrak{M}'$ and a species corresponding to (i, j, I, P) is also present in $[\![\hat{S'}]\!]$, with the same population. Hence $[\![\hat{S'}]\!] \doteq \mathfrak{M}'$. Instead, if in B there are no other boxes belonging to the same species of $I[\,P\,]_n$, then the species (i, j, I, P) has population equal to 0 in $\mathfrak{M}'$ and a species corresponding to (i, j, I, P) is not present in $[\![\hat{S'}]\!]$. Hence $[\![\hat{S'}]\!] \leq\doteq \mathfrak{M}'$.

(case m5)

We know that $[\![\hat{S}]\!]$ contains species (i_1, j_1, I_1, P_1) and (i_2, j_2, I_2, P_2) and, by Prop. 6.2.1, S is such that its bio-process is structurally congruent to $I_1[\,P_1\,]_{n_1} \parallel I_2[\,P_2\,]_{n_2} \parallel B$. Moreover, by encoding definition, we know that the two species in $\hat{S}$ are part of two different machine complexes $(m_1, k_1, \{(l_1, i_1)\} \cup V_1, E_1)$ and $(m_2, k_2, \{(l_2, i_2)\} \cup V_2, E_2)$ and hence we have that n_1 and n_2 belong to links l_1 and l_2 in the environments of S such that $l_1 \not\sim_c l_2$. By hypothesis we know that $I_1 = I_{11}^* \oplus(x_1, \Delta_1)^{r_1} I_{12}^*$ and $I_2 = I_{21}^* \oplus(x_2, \Delta_2)^{r_2} I_{22}^*$, and since $(i_1, i_2, r, \Delta_1, \Delta_2, bind)$ is in $[\![\hat{S}]\!]$, then by encoding definition it is $\alpha_b^s(\Delta_1, \Delta_2) \neq 0$. Hence, by using rules (s3), (s7) and (s8) we can derive $\hat{S} \rightarrow_r S'$ where the bio-process of S' system is structurally congruent to $I_{11}^* \otimes(x_1, \Delta_1)^{r_1} I_{12}^*[\,P_1\,]_{n_1} \parallel I_{21}^* \oplus(x_2, \Delta_2)^{r_2} I_{22}^*[\,P_2\,]_{n_2} \parallel B$ and its environment is updated by adding the new link $\{\Delta_1 n_1, \Delta_2 n_2\}$. Note that we can hence derive also $S \rightarrow_r S'$. By encoding definition we know that $[\![\hat{S'}]\!]$ contains all the species defined in B and species corresponding to boxes $I_{11}^* \otimes(x_1, \Delta_1)^{r_1} I_{12}^*[\,P_1\,]_{n_1}$ and $I_{21}^* \oplus(x_2, \Delta_2)^{r_2} I_{22}^*[\,P_2\,]_{n_2}$, as $\mathfrak{M}'$. In the machine term $\mathfrak{M}'$, moreover, the population of the complexes $(m_1, k_1, \{(l_1, i_1)\} \cup V_1, E_1)$ and $(m_2, k_2, \{(l_2, i_2)\} \cup V_2, E_2)$ is decreased by one and a new machine complex representing the binding of the two machine complexes is added. By encoding definition, the new machine complex corresponds exactly to the one that can be constructed from the set of links containing l_1 and l_2 that can now be obtained by partitioning the environment of S' by means of $\sim_c$. In S', indeed, it results

$l_1 \sim_c l_2$. In the partition, moreover, the number of set of links corresponding to machine complexes $(m_1, k_1, \{(l_1, i_1)\} \cup V_1, E_1)$ and $(m_2, k_2, \{(l_2, i_2)\} \cup V_2, E_2)$ is decreased by one with respect to $\hat{S}$. As for the previous case, if none of the species and complexes involved in the derivation assumes population 0, by encoding definition it results that the overall set of complexes and species in $\mathfrak{M}'$ coincide with the one in $[\![\hat{S}']\!]$ and hence we have $[\![\hat{S}']\!] \doteq \mathfrak{M}'$. Instead, if there are species and complexes that assume population 0 in $\mathfrak{M}'$, then by encoding definition the only difference between $\mathfrak{M}'$ and $[\![\hat{S}']\!]$ is that in $[\![\hat{S}']\!]$ these species and complexes are not present. Hence it results $[\![\hat{S}']\!] \leq \doteq \mathfrak{M}'$.

All the other cases are similar. □

Corollary 6.2.11. *Let $S \in \overline{\mathcal{S}}$. $[\![\hat{S}]\!] \mapsto_{g_1} \mathfrak{M}_1 \mapsto_{g_2} \mathfrak{M}_2 \mapsto_{g_3} \cdots \mapsto_{g_n} \mathfrak{M}_n \Rightarrow \exists S_1, S_2, \cdots, S_n$ such that $S \to_{g_1} S_1 \to_{g_2} S_2 \to_{g_3} \cdots \to_{g_n} S_n$ and $[\![\hat{S}]\!]_i \leq \doteq \mathfrak{M}_i$ for all $i = 1, \cdots, n$.*

Proof Sketch. It can be proved by using Lemma 6.2.7 and Theorem 6.2.10. □

Theorem 6.2.12 (Termination)**.** *Let $S \in \overline{\mathcal{S}}$. $S \not\to_g$ with $g \neq 0 \iff [\![\hat{S}]\!] \not\mapsto_g$.*

Proof Sketch. ($\Leftarrow$) Let $S = (B, E, \xi)$. Since by definition we have that if $S \not\to_g$ with $g \neq 0$, then the structure of B and ξ are such that operators $\boxplus_c$, $\bowtie_b$, $\bowtie_u^{\mathfrak{C}}$, $\bowtie_c^{\mathfrak{C}}$, $\bowtie_i$ generate a set of reactions where all the reactions refer to species with population equal to zero; Thus, no rule can be applied on $[\![\hat{S}]\!]$, resulting $[\![\hat{S}]\!] \not\mapsto_g$.
($\Rightarrow$) By encoding definition $[\![\hat{S}]\!] \not\mapsto_g$ implies that the only reactions present in the set of $[\![\hat{S}]\!]$ reactions, are reactions referring to species with population 0 or to edges that are still not present in any complex. If operators $\boxplus_c$, $\bowtie_b$, $\bowtie_u^{\mathfrak{C}}$, $\bowtie_c^{\mathfrak{C}}$, $\bowtie_i$ do not generate reactions on species that have population greater than zero this means, by operators definition, that in $\hat{S}$, and hence in S, the environment and all the boxes are such that they cannot generate any derivation $S \to_g S'$. Hence $S \not\to_g$. □

To guarantee that our encoding preserves duration of reactions, we define also in this setting a measure to calculate the probability to perform a certain reaction $\mathfrak{M} \to_g \mathfrak{M}'$. Also in this case we distinguish between two cases. When we have an *immediate reaction* $\mathfrak{M} \to_\infty \mathfrak{M}'$, we say that the probability of performing the reaction is given by $1/R^0(\mathfrak{M})$, where $R^0(\mathfrak{M})$ denotes the *immediate apparent probability* of $\mathfrak{M}$, that is the total number of immediate reactions that $\mathfrak{M}$ can perform. When instead we have a *stochastic reaction* $\mathfrak{M} \to_g \mathfrak{M}'$, with $g \in \mathbb{R}_{\geq 0} \cup \mathcal{F}$, we say that the probability of performing the reaction is given by $s/R^1(\mathfrak{M})$, where s is equal to g if $g \in \mathbb{R}_{\geq 0}$ or is equal to $\wr g \int_{\mathfrak{M}}$ if $g \in \mathcal{F}$ and where $R^1(\mathfrak{M})$ denotes the *stochastic apparent rate* of $\mathfrak{M}$, i.e., the sum of the rates of all the stochastic reactions in $\mathfrak{M}$. Tab. 6.7 shows how to compute $R^0(\mathfrak{M})$ and $R^1(\mathfrak{M})$ by considering separately each reaction type.

$$\mathfrak{M} = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$$

$$R^0_{intra}(\mathfrak{M}) = \sum_{(i,x,\infty,in,out,mix)\in\mathfrak{R}} \mathsf{P}_{\mathfrak{S}}(i) \times (in \times out - mix)$$

$$R^1_{intra}(\mathfrak{M}) = \sum_{(i,x,s,in,out,mix)\in\mathfrak{R}} \mathsf{P}_{\mathfrak{S}}(i) \times s \times (in \times out - mix)$$

$$R^0_{change}(\mathfrak{M}) = \sum_{(i,x,\infty,change)\in\mathfrak{R}} \mathsf{P}_{\mathfrak{S}}(i) \times change$$

$$R^1_{change}(\mathfrak{M}) = \sum_{(i,x,s,change)\in\mathfrak{R}} \mathsf{P}_{\mathfrak{S}}(i) \times s \times change$$

$$R^0_{bind}(\mathfrak{M}) = \sum_{(i_1,i_2,\infty,\Delta_1,\Delta_2,1,bind)\in\mathfrak{R}} \begin{cases} \mathsf{P}_{\mathfrak{S}}(i_1) \times \mathsf{P}_{\mathfrak{S}}(i_2) & \text{if } i_1 \neq i_2 \\ \mathsf{P}_{\mathfrak{S}}(i_1) \times (\mathsf{P}_{\mathfrak{S}}(i_2) - 1) & \text{if } i_1 = i_2 \wedge \Delta_1 \neq \Delta_2 \\ \dfrac{\mathsf{P}_{\mathfrak{S}}(i_1) \times (\mathsf{P}_{\mathfrak{S}}(i_2) - 1)}{2} & \text{if } i_1 = i_2 \wedge \Delta_1 = \Delta_2 \end{cases}$$

$$R^1_{bind}(\mathfrak{M}) = \sum_{(i_1,i_2,s,\Delta_1,\Delta_2,1,bind)\in\mathfrak{R}} \begin{cases} \mathsf{P}_{\mathfrak{S}}(i_1) \times \mathsf{P}_{\mathfrak{S}}(i_2) \times s & \text{if } i_1 \neq i_2 \\ \mathsf{P}_{\mathfrak{S}}(i_1) \times (\mathsf{P}_{\mathfrak{S}}(i_2) - 1) \times s & \text{if } i_1 = i_2 \wedge \Delta_1 \neq \Delta_2 \\ \dfrac{\mathsf{P}_{\mathfrak{S}}(i_1) \times (\mathsf{P}_{\mathfrak{S}}(i_2) - 1)}{2} \times s & \text{if } i_1 = i_2 \wedge \Delta_1 = \Delta_2 \end{cases}$$

$$R^0_{unbind}(\mathfrak{M}) = \sum_{(i_1,i_2,\infty,\Delta,\Gamma,1,unbind)\in\mathfrak{R}} \mathsf{E}_{\mathfrak{C}}(i_1, i_2, \Delta, \Gamma)$$

$$R^1_{unbind}(\mathfrak{M}) = \sum_{(i_1,i_2,s,\Delta,\Gamma,1,unbind)\in\mathfrak{R}} s \times \mathsf{E}_{\mathfrak{C}}(i_1, i_2, \Delta, \Gamma)$$

$$R^0_{complex}(\mathfrak{M}) = \sum_{(i_1,i_2,\infty,\Delta,\Gamma,k,complex)\in\mathfrak{R}} \mathsf{E}_{\mathfrak{C}}(i_1, i_2, \Delta, \Gamma)$$

$$R^1_{complex}(\mathfrak{M}) = \sum_{(i_1,i_2,s,\Delta,\Gamma,k,complex)\in\mathfrak{R}} s \times k \times \mathsf{E}_{\mathfrak{C}}(i_1, i_2, \Delta, \Gamma)$$

$$R^0_{inter}(\mathfrak{M}) = \sum_{(i_1,i_2,\infty,\Delta_1,\Delta_2,k,inter)\in\mathfrak{R}} \begin{cases} \mathsf{P}_{\mathfrak{S}}(i_1) \times \mathsf{P}_{\mathfrak{S}}(i_2) & \text{if } i_1 \neq i_2 \\ \mathsf{P}_{\mathfrak{S}}(i_1) \times (\mathsf{P}_{\mathfrak{S}}(i_2) - 1) & \text{if } i_1 = i_2 \wedge \Delta_1 \neq \Delta_2 \\ \dfrac{\mathsf{P}_{\mathfrak{S}}(i_1) \times (\mathsf{P}_{\mathfrak{S}}(i_2) - 1)}{2} & \text{if } i_1 = i_2 \wedge \Delta_1 = \Delta_2 \end{cases}$$

$$R^1_{inter}(\mathfrak{M}) = \sum_{(i_1,i_2,s,\Delta_1,\Delta_2,k,inter)\in\mathfrak{R}} \begin{cases} \mathsf{P}_{\mathfrak{S}}(i_1) \times \mathsf{P}_{\mathfrak{S}}(i_2) \times s \times k & \text{if } i_1 \neq i_2 \\ \mathsf{P}_{\mathfrak{S}}(i_1) \times (\mathsf{P}_{\mathfrak{S}}(i_2) - 1) \times s \times k & \text{if } i_1 = i_2 \wedge \Delta_1 \neq \Delta_2 \\ \dfrac{\mathsf{P}_{\mathfrak{S}}(i_1) \times (\mathsf{P}_{\mathfrak{S}}(i_2) - 1)}{2} \times s \times k & \text{if } i_1 = i_2 \wedge \Delta_1 = \Delta_2 \end{cases}$$

$$R^1_{inter}(\mathfrak{M}) = \sum_{(i_1,i_2,f,\Delta,\Gamma,k,inter)\in\mathfrak{R}} \begin{cases} \wr f \int_{\mathfrak{S}} & \text{if } i_1 \neq i_2 \text{ and } \mathsf{P}_{\mathfrak{S}}(i_1) > 0 \text{ and } \mathsf{P}_{\mathfrak{S}}(i_2) > 0 \\ \wr f \int_{\mathfrak{S}} & \text{if } i_1 = i_2 \text{ and } \mathsf{P}_{\mathfrak{S}}(i_1) > 1 \\ 0 & \text{otw} \end{cases}$$

$$R^1_{event}(\mathfrak{M}) = \sum_{(k,N,f,N')\in\mathfrak{R}} \begin{cases} \wr f \int_{\mathfrak{S}} & \text{if } \forall i \in N \text{ it holds } \mathsf{P}_{\mathfrak{S}}(i) \geq N(i) \\ 0 & \text{otw} \end{cases}$$

$$R^0_{event}(\mathfrak{M}) = \sum_{(k,N,\infty,N')\in\mathfrak{R}} \begin{cases} 1 & \text{if } \forall i \in N \text{ it holds } \mathsf{P}_{\mathfrak{S}}(i) \geq N(i) \\ 0 & \text{otw} \end{cases}$$

Table 6.7: Immediate apparent probability and stochastic apparent rate.

In Tab. 6.7, there are two functions that are heavily used. The first one is $\mathsf{P}_{\mathfrak{S}}(i)$, that given a set of species $\mathfrak{S}$ and an identifier i returns the population associated with i in $\mathfrak{S}$; if there is no species with identifier i, the function returns value 0. The other function is $\mathsf{E}_{\mathfrak{C}}(i_1, i_2, \Delta_1, \Delta_2)$, that given a set of complexes $\mathfrak{C}$ takes two species identifiers and two sorts and returns the total number of edges of machine complexes in $\mathfrak{C}$ that connect nodes associated with species i_1 and i_2 through sorts Δ_1 and Δ_2, respectively.

Note that the information saved in all the reactions are essential for a correct computation of immediate apparent probabilities and stochastic apparent rates.

Lemma 6.2.13. *Let $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}$. Then we have $R^0_{change}(B) = R^0_{change}([\![\hat{S}]\!])$ and $R^1_{change}(B) = R^1_{change}([\![\hat{S}]\!])$.*

Proof Sketch. By function definition we have that given $\hat{S} = (B, E, \xi)$ with $B = I_1[P_1]_{n_1} \parallel I_2[P_2]_{n_2} \parallel \cdots \parallel I_m[P_m]_{n_m} \parallel \mathsf{Nil}$ it results:

$$R^0_{change}(B) = Ch^0_x(I_1, P_1) + \cdots + Ch^0_x(I_m, P_m)$$

Moreover, we have that:

$$R^0_{change}([\![\hat{S}]\!]) = \sum_{(i,x,m,chinf)\in\mathfrak{R}} \mathsf{P}_{\mathfrak{S}}(i) \times m$$

where the value m is computed for each species (i, j, I, P) appearing in the reactions considered by the sum, with function $Ch^0_x(I, P)$.

Note that by Prop. 6.2.1, for each $Ch^0_x(I, P)$ in $R^0_{change}([\![\hat{S}]\!])$, there are $\mathsf{P}_{\mathfrak{S}}(i)$ elements $Ch^0_x(I_k, P_k)$ in $R^0_{change}(B)$, and vice-versa, such that $I_k[P_k] \sim_s I[P]$. Moreover, by Prop. 6.1.8, $Ch^0_x(I_k, P_k) = Ch^0_x(I, P)$. Hence, the overall sum coincide and we have $R^0_{change}(B) = R^0_{change}([\![\hat{S}]\!])$.

$R^1_{change}(B) = R^1_{change}([\![\hat{S}]\!])$ can be proved similarly. □

Lemma 6.2.14. *Let $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}$. Then $R^0_{intra}(B) = R^0_{intra}([\![\hat{S}]\!])$ and $R^1_{intra}(B) = R^1_{intra}([\![\hat{S}]\!])$.*

Proof Sketch. Similar to Lemma 6.2.13. □

Lemma 6.2.15. *Let $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}$ and $[\![\hat{S}]\!] = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$. Then $R^0_{bind}(B, \wp_2(\mathcal{T})) = R^0_{bind}([\![\hat{S}]\!])$ and $R^1_{bind}(B, \wp_2(\mathcal{T})) = R^1_{bind}([\![\hat{S}]\!])$.*

Proof Sketch. By Prop. 6.2.1 and by observing that boxes belonging to the same species preserve the number of interfaces and their sorts, it results that the number of combinations of interfaces with sorts Δ and Γ, computed in B by considering all the combinations

of the two sorts on different boxes, and computed in $\mathfrak{S}$ by considering pairs of species and their populations, coincide. Since both $R^0_{bind}(B, \wp_2(\mathcal{T}))$ and $R^0_{bind}(\mathfrak{R})$ are computed by counting the number combinations with $\alpha(\Delta, \Gamma) = \infty$, then $R^0_{bind}(B, \wp_2(\mathcal{T})) = R^0_{bind}(\mathfrak{R})$. Moreover, since both $R^1_{bind}(B, \wp_2(\mathcal{T}))$ and $R^1_{bind}([\![\hat{S}]\!])$ are computed by summing the values $\alpha^s_b(\Delta, \Gamma)$ corresponding to all the combinations with $\alpha(\Delta, \Gamma)^s_b > 0$, then also in this case we have $R^1_{bind}(B, \wp_2(\mathcal{T})) = R^1_{bind}([\![\hat{S}]\!])$. □

Lemma 6.2.16. *Let $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}$ and $[\![\hat{S}]\!] = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$. Then $R^0_{unbind}(B, \xi) = R^0_{unbind}([\![\hat{S}]\!])$ and $R^1_{unbind}(B, \xi) = R^1_{unbind}([\![\hat{S}]\!])$.*

Proof Sketch. By encoding definition we have that the number of links in ξ connecting sorts Δ and Γ and the number of edges in machine complexes in $\mathfrak{C}$ connecting sorts Δ and Γ, coincide. The value $R^0_{unbind}(B, \xi)$ is computed by simply summing the number of links in ξ with unbind affinity equal to ∞. On the other hand, a reaction $(i_1, i_2, \infty, \Delta_1, \Delta_2, 1, \mathit{unbind})$ is in $\mathfrak{R}$ only if $\alpha^s_b(\Delta_1, \Delta_2) = \infty$. The value $R^0_{unbind}([\![\hat{S}]\!])$ is computed by summing, for each entry $(i_1, i_2, \infty, \Delta_1, \Delta_2, 1, \mathit{unbind})$ in $\mathfrak{R}$, the number of edges $\{\Delta_1 l_1, \Delta_2 l_2\}$ connecting nodes with species identifiers i_1 and i_2, respectively. By encoding definition, this calculation counts all the edges of all the machine complexes with immediate unbind. Hence, since the total number of links and edges coincide, we have $R^0_{unbind}(B, \xi) = R^0_{unbind}([\![\hat{S}]\!])$.

Similarly, the value $R^1_{unbind}(B, \xi)$ computes the sum of all the unbind compatibility values of links in ξ with unbind affinity in $\mathbb{R}_{>0}$. A reaction $(i_1, i_2, s, \Delta_1, \Delta_2, 1, \mathit{unbind})$ is in $\mathfrak{R}$ only if $\alpha^s_b(\Delta_1, \Delta_2) = s \in \mathbb{R}_{>0}$. The value $R^0_{unbind}([\![\hat{S}]\!])$ is computed by summing, for each entry $(i_1, i_2, s, \Delta_1, \Delta_2, 1, \mathit{unbind})$ in $\mathfrak{R}$, the unbind compatibility values s of edges $\{\Delta_1 l_1, \Delta_2 l_2\}$ connecting nodes with species identifiers i_1 and i_2, respectively. By encoding definition, this calculation counts all the edges of all the machine complexes with unbind compatibility in $\mathbb{R}_{>0}$. Hence, since the total number of links and edges coincide, we have $R^1_{unbind}(B, \xi) = R^1_{unbind}([\![\hat{S}]\!])$. □

Lemma 6.2.17. *Let $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}$ and $[\![\hat{S}]\!] = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$. Then $R^0_{complex}(B, \xi) = R^0_{complex}([\![\hat{S}]\!])$ and $R^1_{complex}(B, \xi) = R^1_{complex}([\![\hat{S}]\!])$.*

Proof Sketch. Similar to Lemma 6.2.16. □

Lemma 6.2.18. *Let $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}$ and $[\![\hat{S}]\!] = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$. Then $R^0_{inter}(B, \mathcal{T}^2) = R^0_{inter}([\![\hat{S}]\!])$ and $R^1_{inter}(B, \mathcal{T}^2) = R^1_{inter}([\![\hat{S}]\!])$.*

Proof Sketch. Similar to Lemma 6.2.15. □

Lemma 6.2.19. *Let $\hat{S} = (B, E, \xi) \in \overline{\mathcal{S}}$ and $[\![\hat{S}]\!] = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$. Then $R^0_{event}(\hat{S}, E, \xi) = R^0_{event}([\![\hat{S}]\!])$ and $R^1_{event}(\hat{S}, E, \xi) = R^1_{event}([\![\hat{S}]\!])$.*

Proof Sketch. By encoding definition there is a one-to-one correspondence between events in $\hat{S}$ and events in $[\![\hat{S}]\!]$. Both $R^0_{event}(\hat{S}, E, \xi)$ and $R^0_{event}([\![\hat{S}]\!])$ count all the events having rate ∞. By soundness and completeness of the encoding we know that if an event is enabled in $\hat{S}$ then it is also enabled in $[\![\hat{S}]\!]$. Hence $R^0_{event}(\hat{S}, E, \xi) = R^0_{event}([\![\hat{S}]\!])$.

Similarly $R^1_{event}(\hat{S}, E, \xi)$ and $R^1_{event}([\![\hat{S}]\!])$ sum all the evaluations of the function of all the events having rate expressed by a function. By Prop. 6.2.1 and the way in which functions are evaluated, we know that the evaluation of a function f in $\hat{S}$ or in $[\![\hat{S}]\!]$ returns the same value and hence also in this case $R^1_{event}(\hat{S}, E, \xi) = R^1_{event}([\![\hat{S}]\!])$. □

Definition 6.2.20. *Given a system $\hat{S} \in \overline{\mathcal{S}}_s$, the immediate apparent rate of $[\![\hat{S}]\!] = (\mathfrak{R}, \mathfrak{C}, \mathfrak{R})$, denoted with $R^0([\![\hat{S}]\!])$, is computed by the following formula:*

$$\begin{aligned} R^0([\![\hat{S}]\!]) = \; & R^0_{change}([\![\hat{S}]\!]) + R^0_{intra}([\![\hat{S}]\!]) + R^0_{bind}([\![\hat{S}]\!]) + R^0_{unbind}([\![\hat{S}]\!]) + \\ & R^0_{complex}([\![\hat{S}]\!]) + R^0_{inter}([\![\hat{S}]\!]) + R^0_{event}([\![\hat{S}]\!]) \end{aligned}$$

Moreover, the stochastic apparent rate of $\hat{S}$, denoted with $R^1([\![\hat{S}]\!])$, is computed by the following formula:

$$\begin{aligned} R^1([\![\hat{S}]\!]) = \; & R^1_{change}([\![\hat{S}]\!]) + R^1_{intra}([\![\hat{S}]\!]) + R^1_{bind}([\![\hat{S}]\!]) + R^1_{unbind}([\![\hat{S}]\!]) + \\ & R^1_{complex}([\![\hat{S}]\!]) + R^1_{inter}([\![\hat{S}]\!]) + R^1_{event}([\![\hat{S}]\!]) \end{aligned}$$

Theorem 6.2.21 (Duration)**.** *Let $S \in \overline{\mathcal{S}}$. Then $R^0(\hat{S}) = R^0([\![\hat{S}]\!])$ and $R^1(\hat{S}) = R^1([\![\hat{S}]\!])$.*

Proof Sketch. Follows directly from the previous lemmas. □

6.3 Stochastic simulation

In this section we show how an extended version of the Gillespie algorithm can be implemented on the BlenX abstract machine. The algorithm is performed on a BlenX machine term by choosing, for each simulation step, the fastest reaction in the machine and making the machine term to evolve accordingly. Obviously, since immediate reactions have the precedence with respect to stochastic reactions, the Gillespie algorithm is extended to deal with this aspect. Our algorithm is similar to the efficient variant of the Gillespie algorithm proposed by Gibson and Bruck (see Chapt. 2).

Tab. 6.8 introduces the pseudo-code of the sBlenX stochastic simulation algorithm. It takes a well-formed sBlenX system S and a limit time *limitTime*. The procedure generates the machine term $[\![\hat{S}]\!] = (\mathfrak{S}, \mathfrak{C}, \mathfrak{R})$ and then invokes the procedure InitEnv(), which initializes the rest of simulation environment. Indeed, apart from containing the

```
SIMULATION (S,limitTime):
  let 𝔐 = [[Ŝ]];
  INITENV();
  actualTime : = 0;
  print initial configuration;
  while true do

    if actualTime > limitTime then
      quit the loop;

    if R^0(Ireact) > 0 then
      choose react in Ireact with probability R^0_type({react}) / R^0(Ireact);
      newTime := actualTime;
    else
      (react, newTime) = STOCHASTICSELECTION();
      if newTime := ∞ then
        quit the loop;

    AffectedSpecies = EXECUTE(react);
    if react not immediate then
      UPDATETIME(AffectedSpecies);

    print actual configuration;
    actualTime := newTime;


COMPUTETIME (react):
  if R^1_type({react}) = 0 then
    return ∞;
  rn := RANDOM[0,...,1];
  return 1 / R^1_type({react}) × log(1/rn);
```

Table 6.8: Pseudo-code of the stochastic simulation slgorithm.

sets characterizing the machine term, the simulation environment is made up also of a collection of data-structures that allow for a more efficient implementation.

First of all the environment is initialized by creating a set of immediate reactions *Ireact* and a set of stochastic reactions *Sreact*. In particular, the set $\mathfrak{R}$ is partitioned in two disjoint sets $\mathfrak{R}_1 \cup \mathfrak{R}_2 = \mathfrak{R}$ such that $\mathfrak{R}_1$ contains all the immediate reactions and $\mathfrak{R}_2$ contains all the stochastic reactions (we use *react*, *react*$'$, $react_1$, ... to range over elements in $\mathfrak{R}$). *Ireact* and *Sreact* are initialized with $\mathfrak{R}_1$ and $\mathfrak{R}_2$, respectively.

Following the Gibson and Bruck implementation, also in our case we have an indexed priority queue PQ. It is an indexed priority queue implemented as an heap. Elements in PQ have associate a value *time*, that can be a real number of the value ∞, which is assumed to be greater than all the real numbers. The queue holds elements ordered by the time value.

Instead of having a dependency graph (as the Gibson and Bruck implementation) we use a hash map *HMap*. Given the identifier i of a species, *HMap*(i) returns the set of reactions in *Sreact* involving the species i. Each reaction in *Sreact*, moreover, is associated univocally to an element in the indexed priority queue PQ and, on the other end, each element in PQ is associated univocally with a reaction in *Sreact*. We assume to have functions *GetPQ* and *GetReact*, that given a priority queue element *pqElem* or a reaction in *Sreact*, return the corresponding reaction or the PQ element, respectively. Fig. 6.4 shows how *HMap*, *Sreact* and PQ are related.

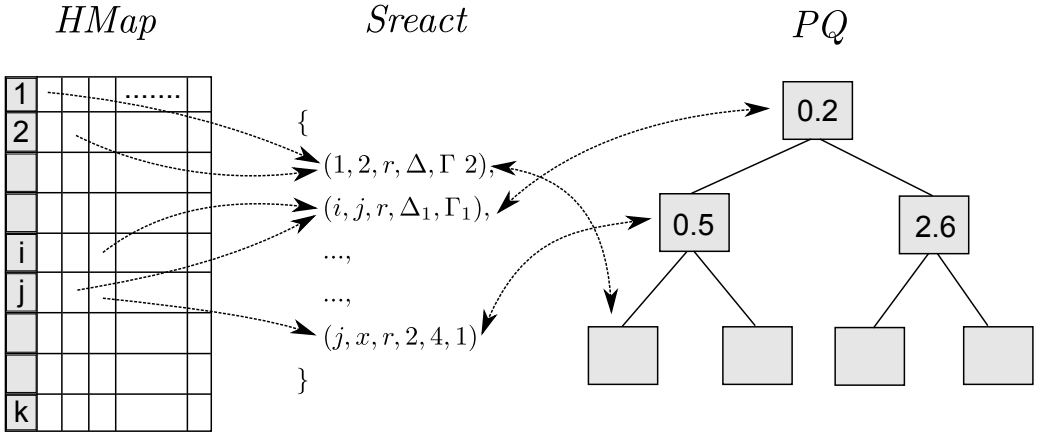


Figure 6.4: The data structures used by the simulation environment.

Procedure InitEnv() initializes all these data structures. The *time* value of each element *pqElem* in PQ is initialized by invoking the function ComputeTime on the reaction *GetReact*(*pqElem*). When a time value in the priority queue is modified, the PQ is automatically reordered.

Then, the value *actualTime* is initialized to 0 and the simulation loop starts. If the *actualTime* is greater than *limitTime*, then the simulation loop is terminated. If this is not the case, the algorithm controls the presence of immediate actions. If $R^0(Ireact)$ is greater than zero, then in the machine term there are immediate reactions that can be executed. Since they have the precedence with respect to stochastic reactions, one of these reactions in *Ireact* is chosen with a probability proportional to $R^0(Ireact)$; each immediate reaction can be chosen with probability $R^0_{type}(\{react\})/R^0(Ireact)$ (*type* indicates the type of reaction *react*, in accordance with Tab. 6.7). In this case *newTime*, that is the simulation

time after the execution of the reaction, is equal to *actualTime*; immediate actions take time 0 to be executed.

If no immediate reactions are present, then a stochastic reaction is selected using the function StochasticSelection(). This function simply takes the top element *pqElem* in *PQ*, invokes *GetReaction(pqElem)*, obtaining a reaction *react* in *Sreact*, and returns the pair $(react, time)$, where *time* is the time value associated with the selected *pqElem*; this value is assigned to *newTime*. Note that the top element of *PQ* always refers to the fastest reaction.

If the *newTime*, resulting from the stochastic selection, is equal to ∞, this means that all the elements in *PQ* have time value equal to ∞, which consequently means that all the reactions in *Sreact* are not enabled. Hence, the simulation loop is terminated.

If we are still in the simulation loop, the selected reaction *react*, both if immediate or stochastic, is executed by invoking the function Execute(*react*); it returns a set of species identifiers *affectedSpecies*. Function Execute implements the machine dynamics rules of the abstract machine presented in the previous section and returns the identifiers of the species that are involved in the reaction. For example, consider $react = (i, x, change, ch)$ and suppose that species i is in at least one complex. Then rule:

$$\frac{\begin{array}{c}(i, x, change, type) \in \mathfrak{R} \quad (i, j, I, P) \in \mathfrak{S} \\ P \equiv_p \langle C\rangle \mathsf{ch}(x, \Gamma, r).Q_1 + M_1 \mid Q_2 \quad \wr\, C \,\int_I = true \quad I = I_1^* \oplus (x, \Delta)^{r'} I_2^* \quad \Gamma \notin \mathsf{sorts}(I_1^* I_2^*) \\ \mathfrak{S}_1 = \mathfrak{S} \boxminus_s (I, P) \quad (\mathfrak{S}_2, \mathfrak{S}_3, k, \mathfrak{R}') = \mathfrak{S}_1 \boxplus_s (I_1^* \oplus (x, \Gamma)^{r'} I_2^*, Q_1 \mid Q_2) \\ \mathfrak{C}_1 = (\mathfrak{C} \boxminus_c (\{(l, i)\} \cup V, E)) \boxplus_c (\{(l, k)\} \cup V, E)\end{array}}{\mathfrak{M} \mapsto_r (\mathfrak{S}_2, \mathfrak{C}_1, \mathfrak{R} \cup \mathfrak{R}' \cup (\bigcup \mathfrak{S}_2 \bowtie^{\mathfrak{C}_1} \mathfrak{S}_3))} \quad \text{(m3)}$$

is implemented by first selecting in P one of the possible enabled actions $\langle C\rangle \mathsf{ch}(x, \Gamma, r).Q_1 + M_1 \mid Q_2$ with uniform probability (among all the enabled change actions on x) and in $\mathfrak{C}$ a complex $(\{(l, i)\} \cup V, E)$ with uniform probability (among all the complexes, with population greater thatn zero, having nodes referring to species i). The rule is applied on the selected elements and the set $affectedSpecies = \{i, k\}$ is returned. In implementing the execution, moreover, when the new reactions are added to the machine reactions set, also sets *Ireact* and *Sreact* are updated consequently adding to them the new immediate and stochastic reactions, respectively.

If the executed reaction is not an immediate reaction, the set *affectedSpecies* is then used as parameter in the invocation of UpdateTimes. For each species identifier i contained in *affectedSpecies*, the procedure uses $HMap(i)$ to obtain the set of reactions involving species i and substitute the time values of the corresponding elements in *PQ* with a value obtained from the sum of the *actualTime* and the value obtained with

ComputeTime applied on the corresponding reaction. Note that since in *affectedSpecies* we can have reactions referring to more than one species identifier, to avoid multiple calculations we can restrict the substitution only for the time values lower or equal than the *actualTime*; when is bigger, indeed, this means that its value has been already recomputed. The times corresponding to the reactions with identifiers not in *affectedSpecies* are not recomputed. Indeed, since their apparent stochastic rates do not change, then there is no need to recompute the corresponding reaction time; we can keep the one already in the priority queue element.

After the update, the *actualTime* is substitued with the *newTime*, the actual machine configuration is printed and the loop restarts. Note that the only case in which the simulation loop does not terminate is when an infinite sequence of immediate reactions is executed.

The main difference between our simulation algorithm and the various Gillespie's implementations, is that we have to deal with immediate reactions and with a number of species, machine complexes and reactions that is dynamic, i.e., can change during the simulation. This is why we preferred an hashmap instead of a dependency graph; modifying dynamically a dependency graph can be computationally expensive. However, the dynamical nature of our systems causes our algorithm to have an execution step complexity which is a bit more expensive with respect to the Gibson and Bruck implementation. The complexity of their algorithm, indeed, is $O(log\ r)$, where r is the number of reactions in the system.

Although a precise description of the complexity of our algorithm is outside the scope of this thesis, we can say that the complexity of our simulation algorithm is $O(f_1(r_{imm}) + f_2(s) + f_3(c) + log\ r_{stoch})$, where r_{imm} is the number of immediate reactions, s is the number of species, c is the number of complexes and r_{stoch} is the number of stochastic reactions. With f_1, f_2 and f_3, moreover, we indicate functions such that $O(f_1(r_i)) \leq O(r_i)$, $O(f_2(s)) \leq O(s)$ and $O(f_3(c)) \leq O(c)$. Functions f_1, f_2 and f_3 depend on how operators on set of species, set of complexes and reactions are implemented and on which data-structures we use to manage immediate reactions. The main goal regarding the simulation algorithm was only to show that by compressing structural congruent boxes and complexes we can simulate complex systems with dynamic number of species, complex and reactions, using the same principles on which the NRM is based, hence obtaining exact and efficient simulations.

Chapter 7

The Beta Workbench

The Beta Workbench (BWB for short) is a set of tools to design, simulate and analyse models written in sBlenX; it is freely available for non commercial purposes at www.cosbi.eu. The core of BWB is a command-line application that hosts three tools: the BWB simulator, the BWB CTMC generator and the BWB reactions generator. This application, implemented in C++, takes as input three text files that represent a sBlenX system, passes them to the compiler that translates these files into a runtime representation that is then stored into a runtime environment. The BWB simulator is a stochastic simulation engine and its runtime environment is the implementation of the stochastic abstract machine and the simulation algorithm we introduced in the previous chapter. It is important to underline that the last version of the BWB simulator implements also some extensions of sBlenX that are not presented in this thesis.

The BWB reactions generator identifies all the complexes and species that could be generated by the execution of a sBlenX system as a result of interactions and state changes, without actually executing or simulating the model [65]. The output of the tool is a description of the system as a list of box species and a list of reactions in which these species are involved. These lists are abstracted as a digraph in which nodes represent species and edges represent reactions. This graph can be reduced to avoid, whenever possible, the presence of immediate reactions. The final result is an SBML description of the original sBlenX system.

Around the core BWB, several tools have been developed to ease model writing and interpretation of results. Examples are the BWB Designer and the BWB Plotter. The BWB designer is a tool that allows to write sBlenX systems both in a textual and in a graphical way. The two representations are interchangeable: the tool can parse and generate the graphical representation from any valid sBlenX system description, and generate the textual representation from the graphical form. In particular, it is possible to draw boxes, processes, interactions, events and to form complexes using graphs. The textual

representation can then be used as input to the core BWB. The BWB Plotter, instead, is a tool that can be used to perform some initial analysis of the simulation results. It is able to display the reactions as both a graph (with some different layouts) and a reaction list; it plots the variation of population of all or selected species and it can be used to see how species and complexes are structured.

The logical arrangement of the computational blocks of BWB is depicted in Fig. 7.1.

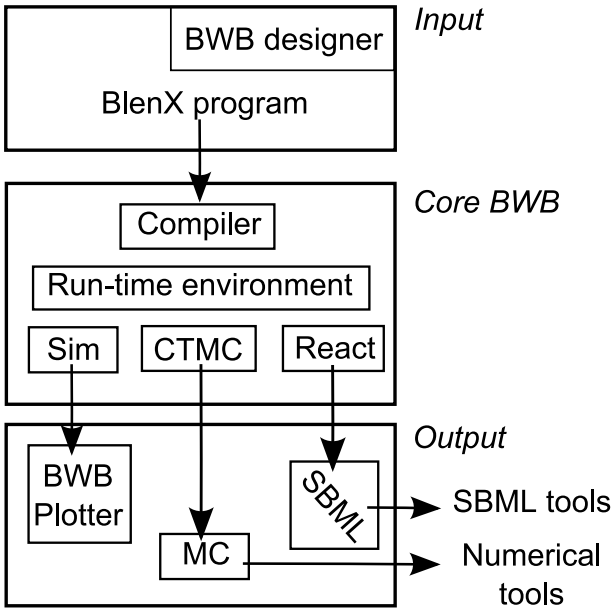


Figure 7.1: The logical strucure of BWB.

My main thesis contribution with respect to the BWB concerns the implementation of the stochastic simulation engine for sBlenX, its runtime environment and the CTMC generator.

7.1 The concrete syntax

To simplify the writing of sBlenX systems, a concrete syntax for the language has been defined. A BlenX program, i.e., a sBlenX system written in the concrete syntax, is made of an optional *declarations* file for the declaration of user-defined constants and functions, a *sorts definition* file that associates unique sorts to interfaces of boxes used by the program and a *program* file, that contains the specification of the bio-processes, events and complexes.

Before starting a more detailed presentation of the concrete syntax we show how the enzyme activation examples we used throughout the previous chapters to introduce BlenX and sBlenX are implemented in the concrete syntax. The complete code of the three

examples is reported in Tab. 7.1. It gives an initial overview of how a BlenX program is structured; each row represent a different program.

Program file	**Sorts file**	**Declarations file**

`[steps = 1000]`
`<< BASERATE:inf >>`
 	
`let E : bproc = #(x,DE)`
`  [ rep x!().nil ];`
`let S : bproc = #(y,DS)`
`  [ y?().ch(y,DP).nil ];`
 	
`run 1 E || 100 S`
	`{ DE, DS, DP }`
	`%%`
	`{ (DE,DS,`
	`  rate(ka),`
	`  rate(kb),`
	`  rate(kc)`
	`  ),`
	`  (DE,DP,0,inf,0) }`
		`let ka : const = 1.0;`
		`let kb : const = 1.0;`
		`let kc : const = 10.0;`

`[steps = 1000]`
 	
`let E : bproc = #(x,DE)[nil];`
`let S : bproc = #(y,DS)[nil];`
`let P : bproc =  #(y,DP)[nil];`
`let ES : bproc = #(y,DES)[nil];`
 	
`when(E,S::rate(ka)) join(ES);`
`when(ES::rate(kb)) split(E,S);`
`when(ES::rate(kc)) split(E,P);`
 	
`run 1 E || 100 S`
	`{ DE, DS, DP, DES }`	`let ka : const = 1.0;`
		`let kb : const = 1.0;`
		`let kc : const = 10.0;`

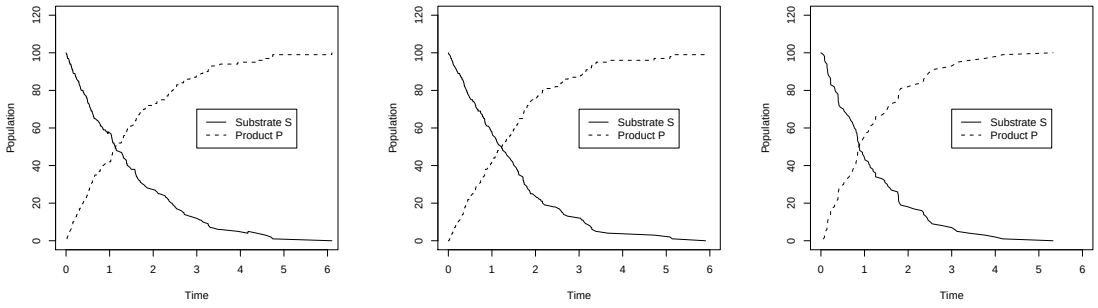
`[steps = 1000]`
`<< BASERATE:inf >>`
 	
`let E : bproc = #(x,DE)`
`  [ rep x!().nil ];`
`let S : bproc = #(y,DS)`
`  [ y?().ch(y,DP).nil ];`
 	
`run 1 E || 100 S`
	`{ DE, DS, DP }`
	`%%`
	`{ (DE,DS,f1) }`
		`let ka : const = 1.0;`
		`let kb : const = 1.0;`
		`let kc : const = 10.0;`
		`let Km : const =`
		`  (kb + kc)/ka;`
		`let VMax : const = 100;`
		`let f1: function =`
		`  VMax*(|S|/(Km+|S|));`

Table 7.1: Different programs implementing enzyme catalysis. Rows represent different programs while columns represent different files.

Note how the program files contain the specification of the boxes and the events specifying the enzyme activation example. Although following a programming language

style, note how the concrete syntax is very close to the abstract one. One of the main differences is that instead of specifying a bio-process, a parallel composition of events and an environment, in the concrete syntax we first define a list of boxes and events (and complexes as we will see later) and then in the end we use a specific keyword to initialize the runtime environment with different populations of the defined boxes species (and complexes). By looking at the second program, it is clear that we implemented only specific classes of events and not the general classes of events characterizing sBIenX; this has been done by considering the results we obtained in Chapt. 5 for the B^{e}_{core} subset. Moreover, note how at the beginning of the file we specify the information about the duration of the simulation (in this case 1000 simulation steps) and some global quantitative information that will be explained later. The sorts files contain the list of all the sorts used in the corresponding program files, along with the implementation of the α function; it is implemented by listing all the sorts compatibilities. The declarations files contain the definitions of constants, variables and functions that are used in the corresponding program and sorts files to define rates and specify the number of initial boxes species. The last thing we anticipate is that complexes are specified with an explicit construct that allows directly the construction of graphs of boxes; this avoid the need to specify the environment at the beginning, simplifying the usability of the language.

We conclude this brief introduction by showing in Fig.7.2 the simulation results we obtained by running in the BWB simulator the three examples of Tab.7.1. Each graph in Fig. 7.2 shows the variation of population in time of the box species representing the substrate and the product. The figures show how the three programs implement coherently the biological scenario of enzyme catalysis.



(a) Example of simulation result of the first program in Tab.7.1.

(b) Example of simulation result of the second program in Tab.7.1.

(c) Example of simulation result of the third program in Tab.7.1.

Figure 7.2: Simulation results of the enzyme catalysis programs.

7.1.1 Rates, variables and functions

All the BlenX files share the syntax definition of identifiers, numbers and rates as reported in Tab.7.2. Note that in the following sections, during the description of the programming constructs, we prefix some qualifying words to Id in order to clarify the kind of identifier that can occur in a given position. We will write *boxId*, *sortId*, *funcId* and *varId* to specify the identifiers referring to boxes, sorts, functions and variables, respectively. Syntactically, they are all equal to Id; the disambiguation is done by the BWB compiler, using a symbol table. For examples, if an identifier Id is used in a function declaration, it will be stored as a *funcId* in the symbol table.

$Letter$	::=	$[a-zA-Z]$
$Digit$	::=	$[0-9]$
Exp	::=	$[Ee][+\backslash-]?\{Digit\}$
$real1$	::=	$\{Digit\}^+\{Exp\}$
$real2$	::=	$\{Digit\}^*.\{Digit\}^+(\{Exp\})?$
$real3$	::=	$\{Digit\}^+.\{Digit\}^*(\{Exp\})?$
$Real$	::=	$real1 \mid real2 \mid real3$
$Decimal$	::=	$\{Digit\}^+$
Id	::=	$(\{Letter\}\|\_)(\{Letter\}\|\{Digit\}\|\_)^*$
$number$	:=	$Real \mid Decimal$
$rate$	:=	$number \mid$ **rate** (Id) $\mid$ **inf**

Table 7.2: Definition of identifiers, numbers and rates. For the regular expressions we use the FLEX syntax.

7.1.2 The declarations file

A declarations file is a file with *.decl* extension that contains the definition of variables, constants and functions (see Tab.7.3). Since these constructs are optional, it is possible to skip the definition of the whole file. The declaration file has the syntax reported in Tab. 7.3.

An *expression* is made up of operators and operands. The syntax for the expression *exp* and the possible algebraic operators that can be used is given in the previous table. An operand can be a *number*, an Id that refers to a constant or a variable (see below) or a value $|Id|$ that refers to the actual population of the box species identified by Id. Operator precedence follows the common rules found in every programming language. Operators + and − have the precedence when used as unary operators, while × and /

have the precedence with respect to + and − when used as binary operators.

declarations	::=	*decList*
decList	::=	*dec*
	|	*dec decList*
dec	::=	**let** *Id* : **function** = *exp* ;
	|	**let** *Id* : **var** = *exp* ;
	|	**let** *Id* : **const** = *exp* ;
exp	::=	*number*
	|	*Id*
	|	| *Id* |
	|	**log** (*exp*)
	|	**sqrt** (*exp*)
	|	**exp** (*exp*)
	|	**pow** (*exp* , *exp*)
	|	*exp* + *exp*
	|	*exp* − *exp*
	|	*exp* ∗ *exp*
	|	*exp* / *exp*
	|	−*exp*
	|	+*exp*
	|	(*exp*)

Table 7.3: Declaration file BNF grammar.

A *state variable* or simply *variable* is an identifier that can assume real modifiable values (Real value). The content of a variable is automatically updated when the defining expression *exp* changes. After the variable identifier and the **var** keyword, the user has to specify the expression used to control the value of the variable. Some examples of variable declarations follows:

```
let v1 : var = 10 * |A|;
let mCycB : var = 2 * |X| * log(v1) init 0.1;
```

A *constant* is an identifier that assumes a value that cannot be changed at run-time and specified through a constant expression (an expression that does not rely on any variable or species population $|Id|$ to be evaluated). As an extension, BlenX allows the use of *constant expressions*. Examples of constant declarations and of constant expressions follow:

```
let c1 : const = 1.0;
let pi : const = 3.14;
let c3 : const = (4.0/3.0) * pi * pow(c1, 3);
```

In the current version of BWB, *functions* are parameterless and always return a Real value. As is, a function is only a named expression that can be used to evaluate a rate or to update the content of a state variable. An example of function definition follows:

```
let f1 : function = (k5s / alpha) / (pow((J5 / (m * alpha * |X|)), 4));
```

When a function is used to describe inter-communications, then we can use $|1B|$ and $|2B|$ to identify the population of the boxes performing the inter-communication. In particular, *1B* identifies the box corresponding to the first sort in the compatibility description, while *2B* corresponds to the second.

7.1.3 The sorts definition file

The sorts definition file is a file with *.sorts* extension that stores all the interface sorts that can be used in the program file and in the definition of compatibilities. The BNF grammar of the sorts file is given in Tab. 7.4.

affinities	::=	{ *sortIdList* }
	|	{ *sortbinderIdList* }%%{ *affinityList* }
sortIdList	::=	*sortId*
	|	*bsortId*, *sortIdList*
affinity	::=	(*sortId*, *sortId*, *rate*)
	|	(*sortId*, *sortId*, *funcId*)
	|	(*sortId*, *sortId*, *rate*, *rate*, *rate*)
affinityList	:=	*affinity*
	|	*affinity*, *affinityList*

Table 7.4: Sorts file BNF grammar.

As shown in the previous chapters, *compatibilities* are a peculiar feature of BlenX and are specified by the function α. Thus the sorts file allows the implementation of the α function. By specifying any compatibility in a separate file we put any information about boxes interaction information in a place that can be modified without altering the program file. The utility of a separate description will be showed Chapt. 9.

A *compatibility* is a tuple of three or five elements. The first two elements are interface sorts declared in the *sortIdList*, while the other elements can either be rate values or a single function identifier. If the compatibility tuple contains a single rate value, then the value is interpreted as the rate of *inter-communications* between interfaces with sorts equal to the first and second $sortId$, respectively. If the compatibility tuple, instead, contains three rate values, these values are interpreted as the rates for *binding*, *unbinding* and *complex-communication*, respectively, between interfaces with sorts equal to the first and second $sortId$, respectively. When the element after the two $sortId$ values is a function identifier, the expression associated with the function will be evaluated to yield a value, then interpreted as the rate of the corresponding *inter-communication*. Note finally that

for all the pairs of sorts that are not specified in the list, no compatibility is assumed.

7.1.4 The program file

The central part of a BlenX program is the program file. The program file has a *.prog* extension and is generated by the BNF grammar in Tab. 7.5.

program	::=	*info* ⟨⟨ *rateDec* ⟩⟩ *decList* **run** *bp*
	|	*info decList* **run** *bp*
info	::=	[**steps** = *decimal*]
	|	[**steps** = *decimal*, **delta** = *number*]
	|	[**time** = *number*]
rateDec	::=	*Id* : *rate*
	|	**CHANGE** : *rate*
	|	**BASERATE** : *rate*
	|	*rateDec*, *rateDec*
decList	::=	*dec*
	|	*dec decList*
dec	::=	**let** *Id* : **pproc** = *process* ;
	|	**let** *Id* : **bproc** = *box* ;
	|	**let** *Id* : **complex** = *complex* ;
	|	**when** (*cond*) *verb* ;
bp	::=	*Decimal Id*
	|	*bp* || *bp*

Table 7.5: Program file BNF grammar.

A *prog* file is made up of a header *info*, an optional list of rate declarations (*rateDec*), a list of declarations *decList*, the keyword **run** and a list of starting entities *bp*.

The *info* header contains any information used by the BWB simulator that will execute the program. A stochastic simulation can be considered as a succession of timestamped steps that are executed sequentially, in a non-decreasing time order. Thus, the duration of a simulation can be specified as a **time**, intended as the maximum timestamp value that the simulation clock will reach, or as a number of **steps** that the simulator will schedule and execute. The **delta** parameter can be optionally specified to instruct the simulator to record events only at a certain frequency.

The *rateDec* specifies the global rate associations for individual channel names or for a particular *class* of actions that a program can perform. In addition, a special class **BASERATE** can be used to set a common basic rate for all the actions that do not have

an explicit rate set. The explicit declaration of a rate in the definition of an *action* has the precedence on this global association. Note how global rate specifications for names are used to recover the specification of the function δ.

The list of declarations *decList* follows. Each declaration is a small, self-contained piece of code ended by a ';'. A declaration can be named, e.g., it can have an *Id* that designates uniquely the declaration unit in the program, or it can be nameless. Declarations of boxes, processes and complexes must be named[1], while events are *nameless*.

7.1.5 Processes and boxes

Boxes and processes are generated by the BNF grammar reported in Tab. 7.6. Like for the abstract syntax, the definition of a box specifies the interfaces and the internal process. Each interface can be defined by specifying the rate associated with the subject or by avoiding it; in the latter case it is assumed equal to zero. The absence of boxes identifiers and of the environment (that specifies how boxes are bound together) is due to the fact that in the concrete syntax we simplify the specification of the complexes by introducing a proper language construct (see below). Once having the initial complexes, all the bindings, unbindings and creation/deletion of complexes are managed automatically by the runtime environment.

The concrete syntax of processes is very close to the abstract one. The main difference is in the definition of the conditions guarding actions. First of all, the concrete syntax of conditions relies on the well-know "if" statement. Moreover, we simplified the notation by allowing the specification of conditions also on choices and not only on single actions. Indeed, a condition on choices can be easily translated into a form in which the condition is replicated on all the single elements composing the choice, hence recovering the abstract syntax of conditions. Moreover, the *true* condition is simply omitted. Finally, atoms of conditions can check the sort of the interface associated with a subject, the state of the associated interface but also both the sort and the state together.

The replication operator $*$ is denoted with the keyword **rep** and in the specification of inputs and outputs we can omit the subject if we are interested in pure synchronization. Moreover, change actions can be defined by specifying a characteristic rate or by avoiding it. If the rate of the change is not specified, then the action is associated with the rate specified in the global declaration **CHANGE**, or if also this is not specified, with the **BASERATE**. If no one of these rates is specified then an error occur.

Note that the separate processes definitions allow to simplify the writing and description of boxes and other processes. Each process definition can indeed be used to define

[1]Note that some language constructs, i.e., processes and sequences, can appear without a name throughout a program; they must be named only when they appear as a declaration

box	::=	*interfaces* [*process*]
interfaces	::=	# (*Id* : *rate*, *sortId*)
	|	# (*Id*, *sortId*)
	|	*interfaces*, *interfaces*
process	::=	*par*
	|	*choice*
par	::=	*parElem*
	|	*choice* | *choice*
	|	*choice* | *par*
	|	*par* | *choice*
	|	*par* | *par*
	|	(*par*)
choice	::=	*choice*
	|	*choice* + *choice*
	|	(*choice*)
choice	::=	**nil**
	|	*seq*
	|	**if** *condexp* **then** *sum* **endif**
parElem	::=	*Id*
	|	**rep** *action* . *process*
seq	::=	*action*
	|	*action* . *process*
action	::=	*Id* ! (*Id*)
	|	*Id* ! ()
	|	*Id* ? (*Id*)
	|	*Id* ? ()
	|	**ch** (*Id*, *sortId*)
	|	**ch** (*rate*, *Id*, *sortId*)
condexp	::=	*atom*
	|	*condexp* **and** *condexp*
	|	*condexp* **or** *condexp*
	|	**not** *condexp*
	|	(*condexp*)
atom	::=	(*Id*, *sortId*)
	|	(*Id*, **bound**)
	|	(*Id*, *sortId*, **bound**)

Table 7.6: Program file BNF grammar.

other processes or the internal behaviour of a box. The scoping we assume is static and we do not allow recursive definition. Recursive definitions can be recovered by using the replication operator [99]. An example of processes and boxes definitions is reported below:

```
let KKKs1 : pproc = recv?().ch(p,kaseKKK).ch(recv,baseKKK).s0!();
let KKKs0 : pproc = recv?().ch(p,outKKK).ch(recv,activeKKK).s1!() ;

let KKK: bproc = #(p, kaseKKK), #(recv, baseKKK)
  [ rep s0?().KKKs0 | rep s1?(). KKKs1 | KKKs0 | rep p!(plus).nil ];
```

This example implements a sort of state machine and is part of a program we will introduce in the next modelling part of the thesis.

7.1.6 Complexes

In order to simplify the specification of complexes, the concrete syntax is provided with a proper construct for the definition of complexes. A complex can be defined using the BNF grammar reported in Tab. 7.7.

complex	::=	{ (*edgeList*) ; *nodeList* }
edgeList	::=	*edge*
	|	*edge*, *edgeList*
edge	::=	(*Id*, *Id*, *Id*, *Id*)
nodeList	::=	*node*
	|	*node nodeList*
node	::=	*Id* : *Id* = (*complInterfaceList*) ;
	|	*Id* = *Id* ;
complInterfaceList	::=	*Id*
	|	*Id*, *complInterfaceList*

Table 7.7: Definition of complexes

A complex is created by specifying directly the graph of boxes, i.e., the list of edges (*edgeList*) and the list of nodes (*nodeList*). Each *edge* is a composition of 4 *Ids*. The first and the third identifiers represent node names, while the others represent subject names. Each *node* in the *nodeList* associates to a node name the corresponding box name and specifies the subjects of the bound interfaces. As an example, let us consider the following piece of code:

```
let B1 : bproc = #(x,A0),#(y,A1)
  [ x!(sig) ];
let B2 : bproc = #(x,A0),#(y,A1)
  [ y!(sig) ];
let C : complex =
{
  (
    (Node0,y,Node1,x),(Node1,y,Node2,x),
    (Node2,y,Node3,x),(Node3,y,Node0,x)
  );
  Node0:B1=(x,y);
  Node1:B2=(x,y);
  Node2=Node0;
  Node3=Node1;
};
```

The complex C defines a complex with a structure equivalent to the one reported in Figure 7.3. Note that by specifying all the initial complexes in this way there is no need to specify the initial environment. It is generated and managed by the runtime environment following the mechanisms described by the stochastic abstract machine of sBlenX.

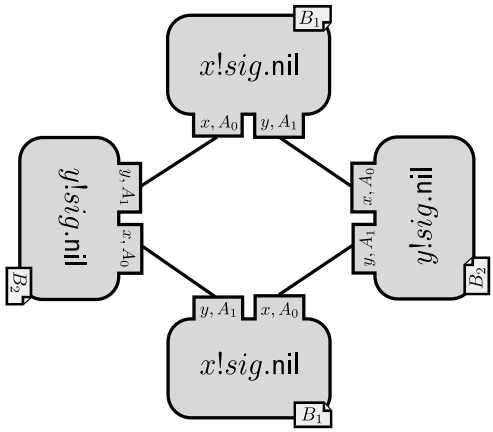


Figure 7.3: Complex generated by the BlenX program.

7.1.7 Events

In the concrete syntax, an events specifies a statement, or a *verb*, to be executed with a specified rate and/or when some conditions are satisfied. The current implementation of events contains some extensions of sBlenX that are not part of this thesis. We introduce here only the syntax of the class of events that are derived from the sBlenX events. The complete concrete syntax of events is reported in Tab. 7.8.

dec	::=	...
	|	**when** (*cond*) *verb* ;
	|	...
cond	::=	*entityList* : *simStatus* : **inf**
	|	*entityList* : *simStatus* : *funcId*
	|	*entityList* : *simStatus* :
	|	*entityList* :: **inf**
	|	*entityList* :: *funcId*
	|	: *EvExpr* :
entityList	::=	*boxId*
	|	*boxId*, *entityList*
simStatus	::=	**time** = *Real*
	|	**steps** = *Decimal*
verb	::=	**split** (*boxId*, *boxId*)
	|	**join** (*boxId*)
	|	**new** (*Decimal*)
	|	**delete** (*Decimal*)
	|	**new**
	|	**delete**

Table 7.8: Events file BNF grammar.

In the concrete syntax, a single event is the composition of a condition *cond* and an action *verb*. Events are used to express reactions that are enabled by global conditions about the simulation status, expressed by *cond*. Conditions are used to trigger the execution of an event when some elements are present in the system, with a given rate, or at a precise simulation time or simulation step. More precisely, a condition *cond* consists of three parts: *entityList*, a list of boxes present in the system; an expression used to enable or disable the event; a *rate* or rate function.

As an example, let us consider the following event:

```
when(A, B :: inf) join (C);
```

The boxes species involved in the event are A and B, as they appear in the *entityList*. If in the current configuration of the system an instance of A and an instance B are present, then the event can fire immediately.

The element *simStatus* specifies: a condition on *simulation time*, that will be satisfied as soon as the simulation clock is greater or equal to the specified time; a conditions on *simulation steps*, that will be satisfied as soon as the step count will exceed the number specified in the *simStatus*. In both cases, the condition will remain *true* until the event is

fired. So, events for which the condition specified is the number of steps or the execution time are guaranteed to fire exactly once. For example, the event:

```
when(A : time = 3.0 : inf) delete;
```

will fire as soon as the simulation clock reaches 3.0, removing an instance of the box species A form the system. Simulation time and simulation step conditions, although not present in the abstract syntax of sBlenX, have been introduced because in a simulation framework are essential to program perturbation of the systems and to observe how the overall behaviour is affected (e.g., the knock-out of a gene at a given time).

Note that the number of *Id*s specified in the *entityList* depends on the event verb that is used for the current event (see below).

An event can be immediate (form 1 and 4 in the BNF specification of *cond*) or its rate can be described with a rate function (form 2 and 5 in the BNF specification of *cond*). Note how rate functions allows to express complex expressions where the stochastic behaviour associated with an event can depend on the population of other boxes species. For the case in which rates are specified as functions (form 2 and 5 in the BNF specification of *cond*), the function is evaluated and the resulting value is used directly to compute the reaction rate. Form 3 and 6 assume that the event is immediate.

Events can split a box into two boxes, join two boxes into a single one, inject or remove boxes into/from the system. These four classes of events correspond in the abstract syntax to the following classes of events:

$$
\begin{array}{ll}
\textbf{(split)} & I[\,P\,]_n \rhd_h I'[\,P'\,]_{n'} \parallel I''[\,P''\,]_{n''} \text{ or} \\
& I[\,P\,]_n \rhd_h I'[\,P'\,]_{n'} \parallel \mathsf{Nil} \\
\textbf{(join)} & I'[\,P'\,]_{n'} \parallel I''[\,P''\,]_{n''} \rhd_h I[\,P\,]_n \\
\textbf{(new)} & I[\,P\,]_n \rhd_h I[\,P\,]_{n_1} \parallel \cdots \parallel I[\,P\,]_{n_m} \\
\textbf{(delete)} & I[\,P\,]_{n_1} \parallel \cdots \parallel I[\,P\,]_{n_m} \rhd_h \mathsf{Nil}
\end{array}
$$

Verbs and conditions have some dependencies: not all verbs can apply to all conditions. The *entityList* in *cond* is used by the event to understand which species the event will modify; at the same time, the *verb* dictates which action will take place. Indeed, a *verb* specify how many entities will be present in the *entityList*: the **split** verb requires exactly one entity to be specified in the condition list; the **join** verb requires exactly two entities to be specified in the condition list; the **new** and **delete** verbs require exactly one entities to be specified in the condition list.

7.1.8 Templates

Templates, often referred to as *generics* or *parametric processes*, are a feature of many programming languages that allows to use an extended grammar, whose corresponding code can contain variable parts that are then instantiated later by the compiler with respect to the base grammar.

In BlenX template code is *specialized* and *instantiated* at compile time using binder identifiers that are passed as template arguments. Therefore, BlenX provides a grammar for defining templates and code to instantiate and use them.

Template declaration

It is possible to define templates for processes, boxes and sequences. The BNF for template declaration is reported in Tab. 7.9.

dec	::=	...
	|	**template** Id : **pproc** $\langle\langle\ formList\ \rangle\rangle = process$;
	|	**template** Id : **bproc** $\langle\langle\ formList\ \rangle\rangle = box$;
	|	**let** Id : **bproc** $= Id\ \langle\langle\ invTempList\ \rangle\rangle$;
form	::=	**name** Id
	|	**rate** Id
	|	**pproc** Id
	|	**sort** Id
formList	::=	*form*
	|	*form*, *formList*
invTempElem	::=	Id
	|	$Id\ \langle\langle\ invTempList\ \rangle\rangle$
invTempList	::=	*invTempElem*
	|	*invTempElem*, *invTempList*
parElem	::=	...
	|	$Id\ \langle\langle\ invTempList\ \rangle\rangle$
bp	::=	...
	|	$Decimal\ Id\langle\langle\ invTempList\ \rangle\rangle$

Table 7.9: Definition of templates.

The declaration of a template **bproc** or **pproc** follows closely the declaration of their standard counterparts, with the **let** keyword substituted by **template**, and an additional list of template formal parameters enclosed by double angular parenthesis.

The template parameter *formList* is a comma-separated list of *form*s; each *form* de-

clares a template argument made up of a keyword along name, pproc, sort followed by an Id. The Id will be added to the environment of the object being defined, acting as a place-holder for the object that will be used during parameter instantiation. For example, in the following code:

```
template P : pproc << pproc P1, name N1, name N2, sort T1 >> =
   x?().N1!().ch(N2, T1).P1;
```

There is no need to define the pproc P_1, nor to insert the sort identifier T_1 into the sorts file: this piece of code will compile without errors, as the process P_1 and the sort identifier T_1 are inserted into P's environment as template arguments. P will be treated by the compiler as a pproc with four template arguments: a process, two names and an interface sort.

Template instantiation

A declared template (pproc or bproc) is held in its symbol-table by the compiler in order to satisfy any following *invocations* or *instantiations* of that template. A template instantiation is a compile time procedure that substitutes the template formal parameters with the actual parameter with which the template object will be used. For example, the following code is a possible instantiation of the previous pproc template:

```
let NilProc : pproc = nil;
let B : bbproc  = #(z, Z)
   [ P<<NilProc, y, z, Z2>> | y?().nil ];
```

The code generate by the compiler, as the result of this instantiation, is equivalent to the following hand-written code:

```
let NilProc : pproc = nil;
let B : bbproc  = #(z, Z)
   [ x?().y!().ch(z, Z2).NilProc | y?().nil ];
```

More precisely, a template is instantiated by using the Id of the template (pproc or bproc) and providing it with a list *invTempList* of comma-separated template invocations *invTempElem*s, whose kind has to match the kind of the template formal parameters.

Note that templates do not increase the expressive power of the language, they only make easier the writing of generic and reusable code.

Part II

Modelling with BlenX

Chapter 8

Signalling networks

In this chapter we investigate the modelling of signalling networks.

We concentrate on a unique underlying example, the MAPK cascade (see Sect. 2.1.3), and we use it to explain different modelling strategies for signalling networks. The modelling session starts with the description of a design pattern that allows to model multiple protein conformational states in a standard way. We then present a simplified model of the MAPK cascade and then improve it in two different ways to obtain the detailed MAPK cascade. All the models are implemented following some standard design patterns, so that they can be reused and adapted to model other similar scenarios.

The correctness of our models is shown by comparing the simulation results of our models against the ones presented in [86], where models of simplified and detailed MAPK cascade have been implemented in π-calculus and simulated with SPiM.

We will refer to the model in [51] as the *detailed* model. However, for simplicity, we start our presentation with a *simplified* version of the model where all the enzymatic reactions:

$$E + S \underset{d_1}{\overset{a_1}{\rightleftharpoons}} ES \xrightarrow{k_1} E + P$$

are substituted with simpler reactions of the form:

$$E + S \xrightarrow{r_1} E + P$$

For the simplified models we follow [86] and set all the reaction rates r_i to a nominal value of $0.05s^{-1}$ and initialize the system with two instances of *E1*, *E2*, *KKPase* and *KPase*, 20 of *KKK* and 200 of *KK* and *K*. Detailed models are implemented with reaction rates derived from [51], where all the a_i reaction rates are equal to $1s^{-1}$ and all the d_i and k_i reaction rates are equal to $150s^{-1}$. Moreover, we initialize the system with one instance of *E1*, *E2* and *KKPase*, 120 of *KPase*, 3 of *KKK* and 1200 of *KK* and *K*.

8.1 Protein conformational states

Here we give a design pattern to model conformational states of proteins.

Proteins are represented by boxes. Internal processes implement kinds of state machines where state changes are driven by the reception of external signals and messages. A state of a box (and hence of the represented protein) is the combination of a certain configuration of interfaces and internal process. Each state is implemented following a common pattern. There is first an input action, followed by a sequence of immediate change actions, followed by an output over the channel representing the destination state:

```
let P_state_i =
  recv?().ch(inf,...). ... .state_j!();
```

It is clear that the sequence of immediate change actions updates the sorts of the interfaces so that the resulting box coincides with the one representing the destination state. Processes representing states can then be combined to define a box as follows:

```
let P_state_0 = x?(). ... .state_1!();
let P_state_1 = y?(). ... .state_2!();
let P_state_2 = x?(). ... .state_0!();

let B : bproc = #(x,state_0_sort1),#(y,state_0_sort2)
  [
    rep state_0?().P_state_0 |
    rep state_1?().P_state_1 |
    rep state_2?().P_state_2 |
    P_state_0 |
    ...
  ];
```

After initializing the box species with the sorts corresponding to the one of the initial state (in this case state 0), the internal process is created by guarding each process representing a state with an input on the corresponding channel and then by composing all these processes in parallel. Moreover, the process representing the initial state is also composed in parallel. Note that other parallel processes, as we will see later, can be present.

The process just described implements a kind of deterministic state machine because, given a state, after receiving a signal, we can go only in one other state. Sometimes, this is not enough and non-deterministic state machines are needed. With non-deterministic we mean that after receiving an input, more and different states can be reached. A first choice can be to condition the state change depending on the reception of a signal from different channels:

```
let P_state_i =
  recv1?().ch(inf,...). ... .state_j1!() +
  recv2?().ch(inf,...). ... .state_j2!() +
  ...                                    +
  recvM?().ch(inf,...). ... .state_jM!() +
;
```

Instead, if we assume that the inputs that guard state changes receive a name, then the state change can depend on the received name, leading to the following description.

```
let P_state_i =
  recv?(what).(
    what!() |
    x1?(). ... .state_j1!() +
    x2?(). ... .state_j2!() +
    ...                     +
    xM?(). ... .state_jM!()
  );
```

Note that instead of consuming only an input, we consume an input and then, depending on the content of the message, we consume an intra-communication that selects one of many possible different state changes.

A visual representation of a box implemented as a state machine is given in Fig. 8.1. The first implements a state machine with two states, where changes happen with the reception of signals over a specified interface. The second implements a state machine with three states where changes depend on the received name. The third implements a more complicated state machine with two intermediate states.

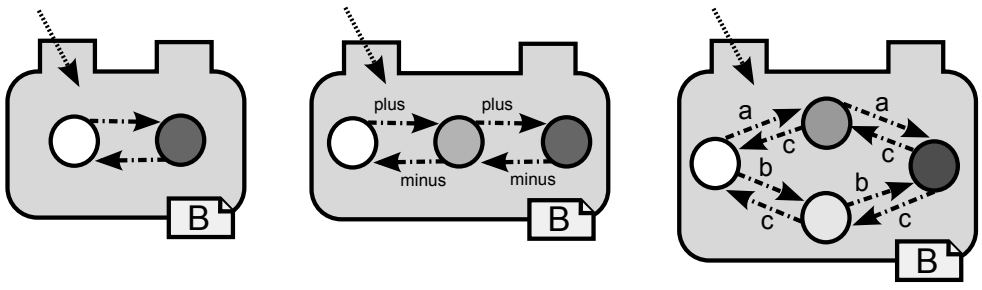


Figure 8.1: Visual representation of three different state machine implementations. The upper arrow means that a signal is received over that interface. Circles represent states and colours identify different states. Arrows represent state changes and labels on arrows (if any) represent the name that drives the state change.

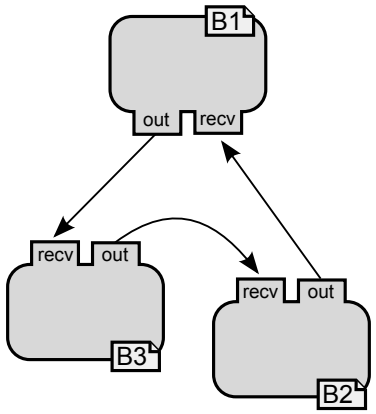


Figure 8.2: Schematic representation of proteins sending and receiving signals.

8.2 A simplified model

Each protein composing the MAPK cascade is represented as a box. Protein domains are divided into a set of *sensing domains* and a set of *effecting domains*. Sensing domains are the places where a protein receives signals. Effecting domains are the places that a protein uses to propagate signals. The internal structure codifies for the mechanism that transforms an input signal into a protein conformational change, which can result in the activation or deactivation of other domains. This description is inspired by the available knowledge of protein structure and function (see for example [103]).

In this first model, a *sensing domain* is represented by an interface, and the mechanism of message-passing (or simply synchronization in some cases) is used to implement the reception of *activation* (e.g., phosphorylation) and *deactivation* (e.g., dephosphorylation) signals sent to the protein. The *effecting domain* is instead used to communicate, and hence to activate or inhibit other proteins. In this setting, a box receives signals from sensing domains and send signals through effecting domains; a schematic representation is given in Fig.8.2.

Following this schema and the state machine pattern introduced previously, *KKK* has one sensing domain and one effecting domain and is activated after receiving a single signal; it is implemented by the following pattern:

```
let KKKs1 : pproc = recv?().ch(p,kaseKKK).ch(recv,baseKKK).s0!();
let KKKs0 : pproc = recv?().ch(p,outKKK).ch(recv,activeKKK).s1!() ;

let KKK: bproc = #(p, kaseKKK), #(recv, baseKKK)
  [ rep s0?().KKKs0 | rep s1?(). KKKs1 | KKKs0 | rep p!(plus).nil ];
let KKKp: bproc = #(p, outKKK), #(recv, activeKKK)
  [ rep s0?().KKKs0 | rep s1?(). KKKs1 | KKKs1 | rep p!(plus).nil ];
```

When *KKK* executes an inter-communication through the interface *#(recv,baseKKK)*, the action *recv*?() in *KKK* is consumed and immediately the actions *#ch(p,outKKK)* and *#ch(recv,activeKKK)* are also consumed (we assume the global rate for the change action to be set to *inf*). The obtained box is structurally congruent to *KKKp* and hence the protein has reached its active form, where the interfaces have associated the right sorts. Now, if the box *KKKp* executes an inter-communication through the interface *#(recv,activeKKK)*, the reverse mechanism is executed and the protein returns back in its inactive form *KKK*.

Note that, although not needed, we define also the active form *KKKp*. In this way we will have the right name associated in the simulation output files. If we do not define the active form, then in the output file the corresponding box species will be present with a random name of the form S_n. In this way we can give a name to all the protein conformational states we are interested in and render hence easier the output interpretation.

KK requires two signals to be activated and hence the implementation pattern is slightly different.

```
let KKs2 : pproc = recv?().ch(p,kaseKK).ch(recv,intKK).s1!();
let KKs1 : pproc = recv?(what).what!() |
                   ( plus?().ch(p,outKK).ch(recv,activeKK).s2!() +
                     minus?().ch(recv,baseKK).s0!() );
let KKs0 : pproc = recv?().ch(recv,intKK).s1!().nil;

let KK: bproc = #(p, kaseKK), #(recv, baseKK)
  [ rep s0?().KKs0 | rep s1?().KKs1 | rep s2?().KKs2 | KKs0 |
  rep p!(plus).nil ];
let KKpp: bproc = #(p, outKK), #(recv, activeKK)
  [ rep s0?().KKs0 | rep s1?().KKs1 | rep s2?().KKs2 | KKs2 |
  rep p!(plus).nil ];
```

These three processes encode the state machine that allows to switch from the inactive to the active state and back. After receiving the first signal (with an activation mechanism similar to the one described for single signal activation), the process *KKs1*, representing an intermediate configuration, is activated. This process presents a choice behaviour: when a name *minus* is received, the process for the inactive state is enabled again; otherwise, if a name *plus* is received, the active process *KKs2* is enabled, leading the protein to reach the conformational state represented by process *KKpp*, the active form. Protein *K* is implemented in the same way.

Finally, signals *E1* and *E2* and phosphatases *KKpase* and *Kpase* are implemented in the following ways:

```
let E1: bproc = #(x, signalE1) [ rep x!().nil ];
let E2: bproc = #(x, signalE2)[ rep x!().nil ];
let KKpase: bproc = #(x, paseKK)[ rep x!(minus).nil ];
let Kpase: bproc = #(x, paseK) [ rep x!(minus).nil ];
```

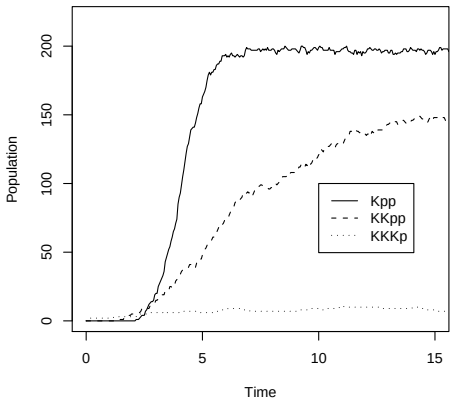
They have no sensing domains and simply send signals continuously. Note that since KKK can be only in active or inactive state, then activation and deactivation (through E_1 and E_2, respectively) can be done by using pure synchronization.

At this point, we have to specify which active protein activates or inhibits each other protein. This is done simply by specifying the compatibilities in the sorts file between the right sorts in the following way:

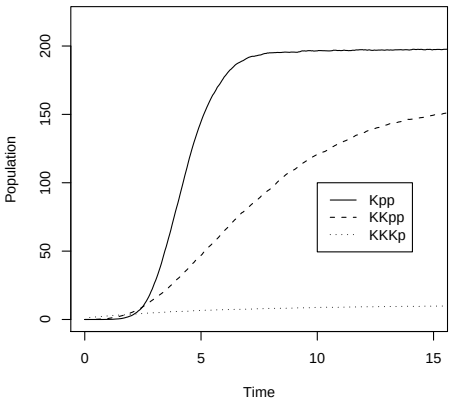
```
{
 baseK, intK, activeK, kaseK, outK,
 baseKK, intKK, activeKK, kaseKK, outKK,
 baseKKK, activeKKK, kaseKKK, outKKK,
 signalE1, signalE2, paseP1, paseP2
}
%%
{
 (signalE1,baseKKK,0.05),
 (signalE2,activeKKK,0.05),
 (outKKK,baseKK,0.05),
 (outKKK,intKK,0.05),
 (paseP2,intKK,0.05),
 (paseP2,activeKK,0.05),
 (outKK,baseK,0.05),
 (outKK,intK,0.05),
 (paseP1,activeK,0.05),
 (paseP1,intK,0.05)
}
```

For example, note how *signalE1* acts on the sort *baseKKK*, representing the sensing domain of the inactive form KKK, while *signalE2* acts on the sort *activeKKK*, representing the sensing domain of the active form $KKKp$.

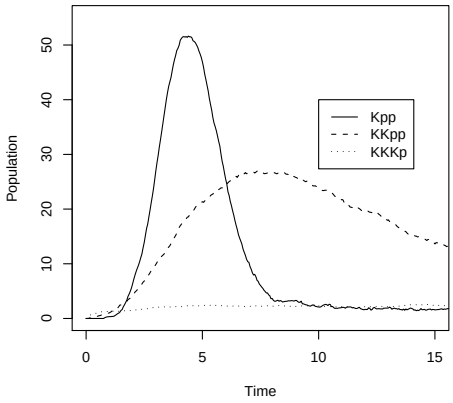
Finally, Fig. 8.3 reports simulation results for this model. Similar response profiles are observed for the output of Kpp with respect to the detailed model presented in [51], despite the differences in the simulation parameters, the system still behaves as an ultrasensitive switch. As we will see later, however, average behaviour and standard deviation are slightly different with respect to the detailed model. This is due to the fact that in the simplified model the switch is more effective and presents less stochasticity.



(a) Example of BWB simulation.



(b) Average over 100 simulations.



(c) Standard deviation over 100 simulations.

Figure 8.3: Simulation results of the MAPK cascade simplified model.

8.3 The detailed model

The detailed model is obtained by improving the previous simplified model. The first modification we have to bring in the model is related to the compatibility values. A detailed implementation of the kinetics:

$$E + S \underset{d_1}{\overset{a_1}{\rightleftharpoons}} ES \xrightarrow{k_1} E + P$$

requires, indeed, the specification of binding, unbinding and complex-communication rates. Following the implementation of the enzymatic reactions we saw in the previous

chapters, we actually implement the kinetics:

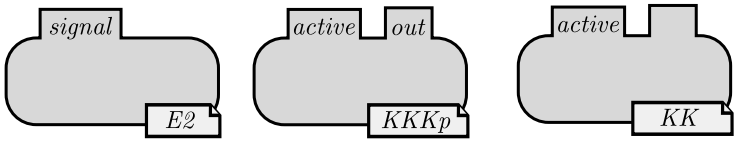
$$E + S \underset{d_1}{\overset{a_1}{\rightleftharpoons}} ES \xrightarrow{k_3} EP \xrightarrow{\infty} E + P$$

which corresponds to the previous one, but that adds an immediate step in the end. Thus, the new sorts file is:

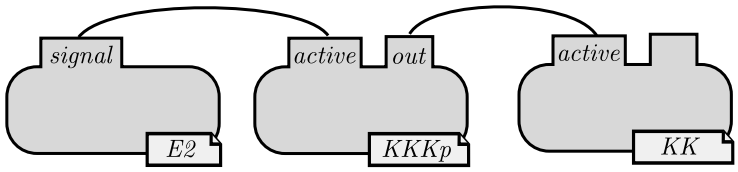
```
{
 baseK, intK, activeK, hiddenK, kaseK, outK,
 baseKK, intKK, activeKK, hiddenKK, kaseKK, outKK,
 baseKKK, activeKKK, hiddenKKK, kaseKKK, outKKK,
 signalE1, signalE2, paseP1, paseP2, UNBIND, INERT
}%%
{
 (signalE1,baseKKK,1,150,150),
 (UNBIND,baseKKK,0,inf,0),
 (signalE2,activeKKK,1,150,150),
 (UNBIND,activeKKK,0,inf,0),
 (outKKK,baseKK,1,150,150),
 (UNBIND,baseKK,0,inf,0),
 (outKKK,intKK,1,150,150),
 (paseP2,intKK,1,150,150),
 (UNBIND,intKK,0,inf,0),
 (paseP2,activeKK,1,150,150),
 (UNBIND,activeKK,0,inf,0),
 (outKK,baseK,1,150,150),
 (UNBIND,baseK,0,inf,0),
 (outKK,intK,1,150,150),
 (paseP1,intK,1,150,150),
 (UNBIND,intK,0,inf,0),
 (paseP1,activeK,1,150,150),
 (UNBIND,activeK,0,inf,0)
}
```

The sort *UNBIND* is used to implement all the last immediate steps, while new sorts *hiddenKKK*, *hiddenKK* and *hiddenK* are used to implement a mechanism that avoids the creation of complexes that are not present in the original MAPK model.

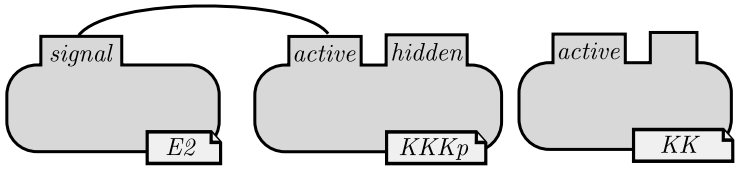
Consider indeed a scenario in which we have *KKKp*, *E2* and *KK*:



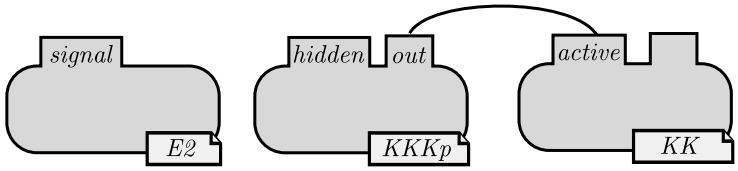
where we omit interface subjects and where sorts are simplified by omitting the protein name. Looking at the compatibilities specification, it results that the system admits a configuration where the following complex can be formed:



This complex, however, is not contemplated in the MAPK model of [51] and hence, to be coherent with the original model, we have to implement boxes avoiding the formation of these intermediates. We implement this mechanism by using the "if" statement and simplify the model by using templates. Consider again the protein *KKKp*; it can act both as an enzyme and a substrate. What we want is that when *E2* binds to *KKKp*, then the other interface is not available. We want hence the formation of a complex:



where *hiddenKKK* and *activeKK* are not compatible. Moreover, we want the same behaviour when *KKKp* binds to *KK*:



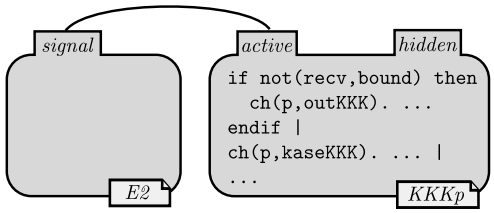
where *hiddenKKK* and *signalE2* are not compatible. This procedure is implemented by the following template:

```
template cInter : pproc
<< name x, name y, name z, sort S, sort T >> =
  if (x,bound) and (y,S) then
    ch(y,T). if not(x,bound) then
               ch(y,S).z!()
             endif
  endif;
```

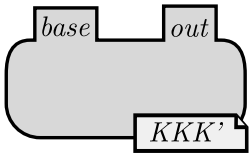
and is used, for example, to implement *KKKp* in the following way:

```
let KKKp: bproc = #(p, kaseKKK), #(recv, baseKKK)
  [
    KKKs1 | rep s0?().KKKs0 | rep s1?().KKKs1 |

    rep c1?().cInter<< p, recv, c1, baseKKK, hiddenKKK >> |
    cInter<< p, recv, c1, baseKKK, hiddenKKK >> |
    rep c2?().cInter<< p, recv, c2, activeKKK, hiddenKKK >> |
    cInter<< p, recv, c2, activeKKK, hiddenKKK >> |
    rep c3?().cInter<< recv, p, c3, kaseKKK, hiddenKKK >> |
    cInter<< recv, p, c3, kaseKKK, hiddenKKK >> |
    rep c4?().cInter<< recv, p, c4, outKKK, hiddenKKK >> |
    cInter<< recv, p, c4, outKKK, hiddenKKK >> |

    ...
  ];
```

The template *cInter* is instantiated for all the interfaces and for each sort they can assume. Now, suppose *E2* and *KKKp* to be bound and consider the previous definition of *KKKs1*. We have that if the two boxes perform an inter-communication on the bound interfaces, then they reach the following configuration:



This configuration leads to different possible executions. The correct one is when the two boxes unbind, the sort of the hidden interface is set again to *outKKK* and finally the state change *ch(recv,baseKKK)* is executed. An incorrect execution, instead, occurs when first the action *ch(recv,baseKKK)* is executed, then the two boxes unbind and finally then the sort of the hidden interface is updated with *outKKK*; in the wrong execution we end up with a box:



that does not represent any valid *KKK* state. For this reason we have to make the sequence of actions deterministic, in a way that the process can proceed only with the correct execution. We do this by rewriting the *KKK* states in the following way:

```
let KKKs1 : pproc =
  recv?().if (p,outKKK) then
            ch(p,kaseKKK).ch(recv,baseKKK).s0!()
          endif ;
let KKKs0 : pproc =
  recv?().if (p,kaseKKK) then
            ch(p,outKKK).ch(recv,activeKKK).s1!()
          endif ;
```

Consider process *KKKs1*. After receiving the signal we are sure that the sort associated with p is *hiddenKKK* and that it will be changed again to *outKKK* immediately after the unbinding. Hence, to keep the correct order it is enough to guarantee that the state change happens only once the sort of interface p is again *outKKK*. All the described sequence is made up of immediate actions.

From the description, it is clear that the unbind reaction after the inter-communication, i.e., the last immediate reaction in the enzymatic kinetics schema, is driven by the protein that acts as an enzyme. All the proteins that act as enzyme, hence, are provided with a parallel internal process that implements this template:

```
template signal : pproc
<<  name x, name y, binder S>> =
  rep init?().x!(y).ch(x,UNBIND).
    if not(x,bound) then
      init!().ch(x,S)
    endif |
  x!(y).ch(x,UNBIND).
  if not(x,bound) then
    init!().ch(x,S)
  endif
;
```

It implements a recursive behaviour where the enzyme, after sending the signal, changes the sort of the output interface with *UNBIND*, waits for the unbind (that by the compatibility value is immediate) and then sets again the right sort. This template is instantiated in all the proteins that act, or can act, as enzymes. Hence, it is instantiated in all the proteins but K.

The complete code that uses templates is reported in Tab. 8.1 and Tab. 8.2. Note that in the complete model we define a template for the implementation of KKK and KK and K and all their intermediate and active states. This example shows the utility of templates.

Moreover, simulation results are reported in Tab. 8.4. Note how the dynamics is in accordance with the one reported in [51].

```
[steps = 10000, delta = 0.001]
<<BASERATE: inf>>

template signal: pproc <<  name x, name y, binder S >> =
  rep init?().x!(y).ch(x,UNBIND).if not(x,bound) then init!().ch(x,S) endif |
  x!(y).ch(x,UNBIND).if not(x,bound) then init!().ch(x,S) endif ;

template cInter : pproc
<< name x, name y, name z, sort S, sort T >> =
  if (x,bound) and (y,S) then
    ch(y,T). if not(x,bound) then ch(y,S).z!() endif endif;

template Kn: bproc
<< sort S0, sort S1, sort S2, sort S3, sort S4, sort S5, sort S6, sort S7,
  pproc P1, pproc P2, pproc P3, pproc P4 >> =
#(p, S6), #(recv, S7)
[ rep s0?().P1 | rep s1?().P2 | rep s2?().P3 | P4 |
  rep c1?().cInter<< p, recv, c1, S0, S5 >> | cInter<< p, recv, c1, S0, S5 >> |
  rep c2?().cInter<< p, recv, c2, S1, S5 >> | cInter<< p, recv, c2, S1, S5 >> |
  rep c3?().cInter<< p, recv, c3, S2, S5 >> | cInter<< p, recv, c3, S2, S5 >> |
  rep c4?().cInter<< recv, p, c4, S3, S5 >> | cInter<< recv, p, c4, S3, S5 >> |
  rep c5?().cInter<< recv, p, c5, S4, S5 >> | cInter<< recv, p, c5, S4, S5 >> |
  signal<< p, plus, S4 >> ];

let pNil : pproc = nil;

// Definition of signals and phosphatases
let E1: bproc = #(x, signalE1) [ signal<< x, plus, signalE1 >> ];
let E2: bproc = #(x, signalE2) [ signal<< x, plus, signalE2 >> ];
let P1: bproc = #(x, paseP1) [ signal<< x, minus, paseP1 >>  ];
let P2: bproc = #(x, paseP2) [ signal<< x, minus, paseP2 >> ];

let KKKs1: pproc =
  recv?().if not(recv,bound) and (p,outKKK) then
            ch(p,kaseKKK).ch(recv,baseKKK).s0!() endif ;
let KKKs0: pproc =
  recv?().if not(recv,bound) and (p,kaseKKK) then
            ch(p,outKKK).ch(recv,activeKKK).s1!() endif ;

let KKK: bproc =
  Kn<<baseKKK,INERT,activeKKK,kaseKKK,outKKK,hiddenKKK,kaseKKK,
      baseKKK,KKKs0,KKKs1,pNil,KKKs0>>;
let KKKp: bproc =
  Kn<<baseKKK,INERT,activeKKK,kaseKKK,outKKK,hiddenKKK,outKKK,
      activeKKK,KKKs0,KKKs1,pNil,KKKs1>>;
```

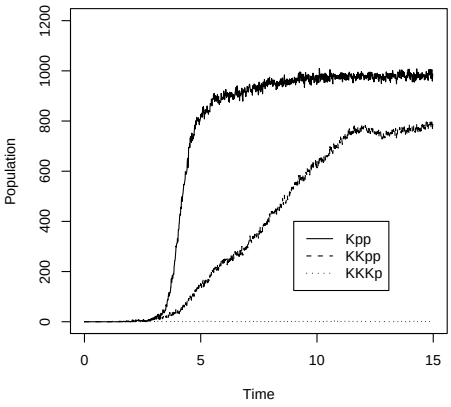
Table 8.1: Complete code of the detailed MAPK cascade model (part 1).

```
let KKs2 : pproc =
  recv?().if (p,outKK) then ch(p,kaseKK).ch(recv, intKK).s1!() endif;
let KKs1 : pproc =
  recv?(what).what!().nil |
    (
      plus?().if (p,kaseKK) then ch(p,outKK).ch(recv, activeKK).s2!() endif +
      minus?().if (p,kaseKK) then ch(recv, baseKK).s0!() endif
     );
let KKs0 : pproc =
  recv?().if (p,kaseKK) then ch(recv,intKK).s1!() endif ;

let KK: bproc =
  Kn<<baseKK,intKK,activeKK,kaseKK,outKK,hiddenKK,kaseKK,baseKK,KKs0,KKs1,
      KKs2,KKs0>>;
let KKp: bproc =
  Kn<<baseKK,intKK,activeKK,kaseKK,outKK,hiddenKK,kaseKK,intKK,KKs0,KKs1,
      KKs2,KKs1>>;
let KKpp: bproc =
  Kn<<baseKK,intKK,activeKK,kaseKK,outKK,hiddenKK,outKK,activeKK,KKs0,KKs1,
      KKs2,KKs2>>;

let Ks2 : pproc =
  recv?().if (p,outK) then ch(p,kaseK).ch(recv, intK).s1!() endif;
let Ks1 : pproc =
  recv?(what).what!().nil |
    (
      plus?().if (p,kaseK) then ch(p,outK).ch(recv, activeK).s2!() endif +
      minus?().if (p,kaseK) then ch(recv, baseK).s0!() endif
    );
let Ks0 : pproc =
  recv?().if (p,kaseK) then ch(recv,intK).s1!() endif ;

let K: bproc =
  Kn<<baseK,intK,activeK,kaseK,outK,hiddenK,kaseK,baseK,Ks0,Ks1,Ks2,Ks0>>;
let Kp: bproc =
  Kn<<baseK,intK,activeK,kaseK,outK,hiddenK,kaseK,intK,Ks0,Ks1,Ks2,Ks1>>;
let Kpp: bproc =
  Kn<<baseK,intK,activeK,kaseK,outK,hiddenK,outK,activeK,Ks0,Ks1,Ks2,Ks2>>;

run 1 E1 || 1 E2 || 1200 K || 1200 KK || 3 KKK || 120 P1 || 1 P2
```

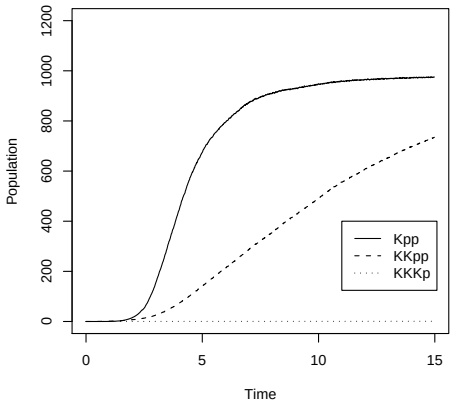
Table 8.2: Complete code of the detailed MAPK cascade model (part 2).

8.4 Using functions

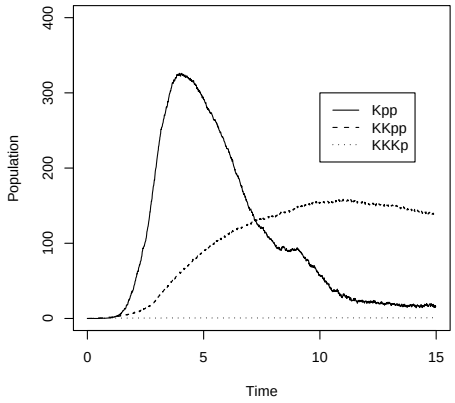
The detailed MAPK cascade model can be obtained also by using functions with the simplified model. By the Michelis-Menten formulation, indeed, we can represent an enzymatic



(a) Example of BWB simulation.



(b) Average over 100 simulations.



(c) Standard deviation over 100 simulations.

Figure 8.4: Simulation results of the MAPK cascade detailed model.

reaction:

$$E + S \underset{d_1}{\overset{a_1}{\rightleftharpoons}} ES \xrightarrow{k_1} E + P$$

with a simplified reaction:

$$E + S \xrightarrow{r} E + P$$

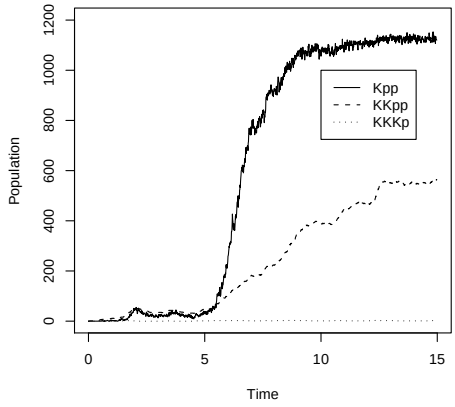
where the rate k is determined by:

$$r = \frac{k_1 \times |E| \times |S|}{\frac{a_1 + k_1}{d_1} + |S|}$$

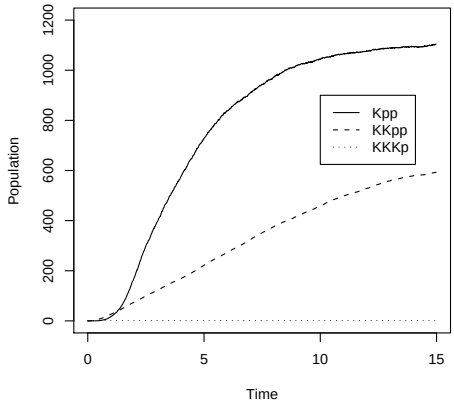
Since all the enzymatic reactions in the detailed model have the same rates, then we can instantiate a declarations file containing only the definition:

```
let f : function = (150 * |1B| * |2B|) / (300 + |2B|);
```

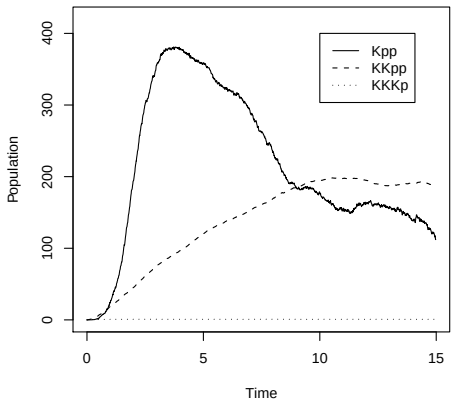
and substitute all the values 0.05 in the sort file of the simplified model with f. In this way, the kinetics governing the communications are described by functions that implement Michaelis-Menten kinetics. Note that we have to be sure that in the sorts file the order of the pair of sorts reflects the fact that the first sorts refers to the box that performs the output, while the second to the box that performs the input. In Tab. 8.5 we can see that the dynamics of the detailed model is preserved.



(a) Example of BWB simulation.



(b) Average over 100 simulations.



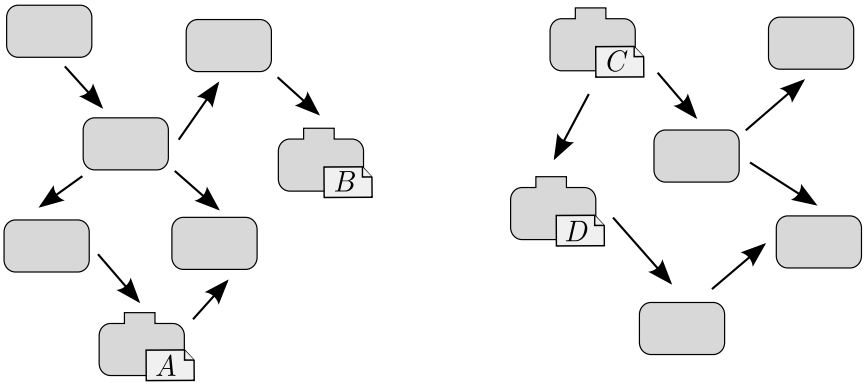
(c) Standard deviation over 100 simulations.

Figure 8.5: Simulation results of the MAPK cascade detailed model with functions.

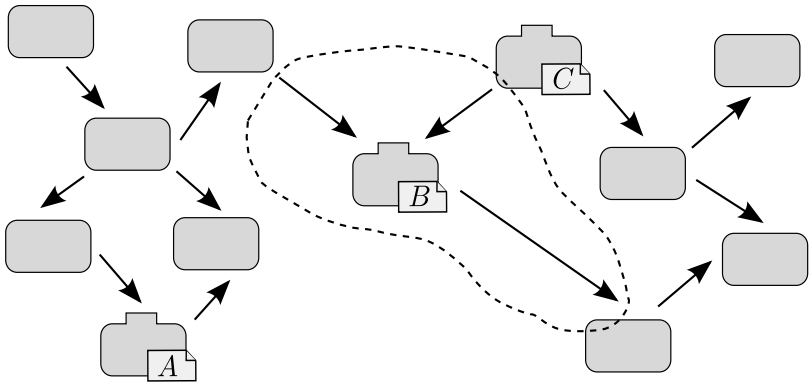
8.5 Composition of networks

Cellular functions, such as signal transductions, are carried out by *modules* made up of many species of interacting proteins. Modules can be linked by *crosstalks*, i.e., protein species in signal transduction are shared between different signal networks and responses to a signal inducing condition can activate multiple responses, or can be linked by an exchange of signals between their components.

Consider a program in which there are two independent signalling networks:

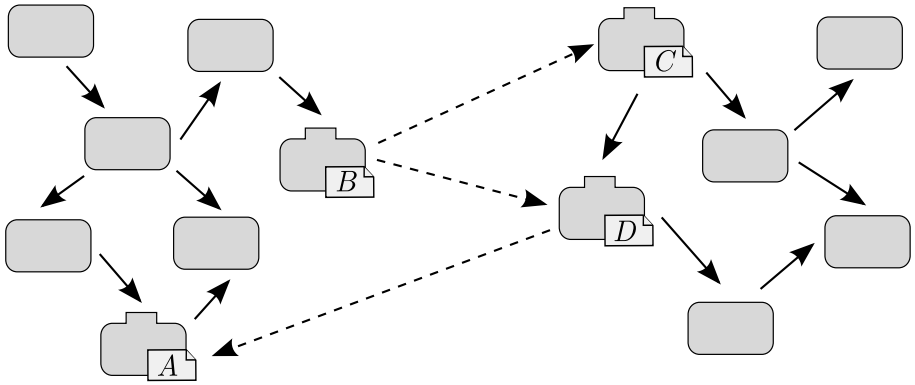


Suppose there exists a crosstalk between the two modules. Being independent, the two networks contain one (or more) protein that belong to the same species but have different names. Suppose for example that B and D represent the same protein species and assume that in both models they are activated by a single signal. Having in mind our design pattern for signalling networks, it is clear that to obtain the crosstalk effect we simply have to delete one of the two boxes (e.g., D), and substitute in the sorts file all the sorts regarding D with the corresponding sorts regarding B. In such a way we obtain a new signalling network in which the crosstalk is present:

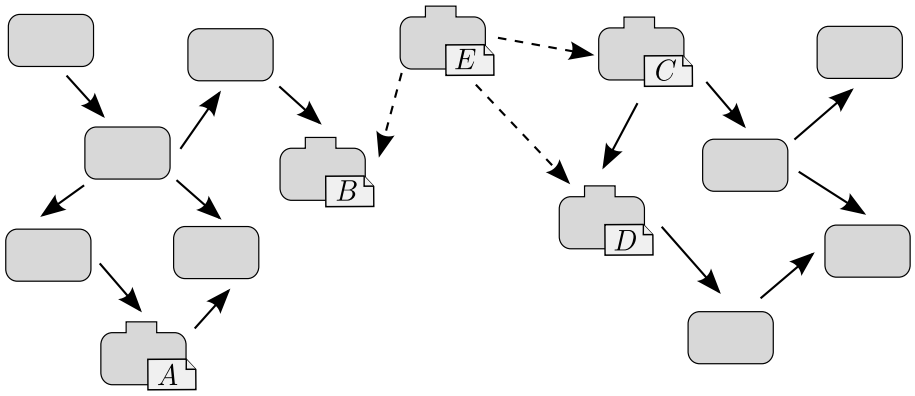


In a different scenario, suppose that all the protein species composing the two independent signalling networks are different. Now, suppose we want to compose the two

networks by adding some interactions between their components. Imagine that B sends an activation signal to C and D and that D sends an activation signal to A:



This can be reproduced by updating the sorts file with new connections. Note that no modifications are needed in the program file. Note also that if we use the design pattern of the detailed model, it is enough to update the sorts file, because all the intermediate complexes are automatically generated during the execution. A similar procedure is needed if we want to add a new protein species E that acts on one or both the networks:



In this case indeed it is enough to define the new protein species and to add the corresponding compatibilities in the sorts file. All these simple examples show how the interaction framework based on compatibility provides a certain compositionality in the description of networks. It is indeed possible to develop different pieces of a model separately and then put all together by working on the compatibilities definition.

It is important to observe that modularity and compositionality can be used only by fixing right from the start a series of policies and design patterns. Obviously, they can differ depending on the scenario and for that it is quite difficult to identify general implementation principles. However, general policies and design patterns are a good practice

in all the programming languages; they indeed give a reusable solution to commonly occurring problem design by providing a description of how to solve a problem that can be used in many different situations.

8.6 Encoding space

In signalling networks the activity of proteins is heavily influenced by their location inside the cell. From a topological perspective a cell can be seen as an architecture of physical membrane-bounded locations, called also compartments. The spatial architecture of compartments, therefore, is an essential aspect of cellular biology which calls for an adequate modelling support.

Here we show how space, and particularly compartments, can be handled in our signalling network BlenX design pattern. We concentrate here on the pattern of the simplified model.

Compartments are usually arranged in a tree-like structure and only proteins belonging to the same compartment can interact. In our compartment representation we rely on a more general model of space in which we simply associate locations to proteins. This association is obtained by stipulating a standard approach in representing protein domains. Following the previously described design pattern, proteins are modelled in BlenX as boxes and their domains as interfaces. Interaction capabilities of proteins are determined by interfaces sorts and their compatibilities. By refining the interpretation of sorts, we can limit the interaction capabilities of boxes and hence, indirectly, represent a particular topological space arrangement. As an example, consider the space arrangement in Fig. 8.6. There are two compartments A and B, one enclosed in the other, and the presence of an enzyme E and a substrate S in both the compartments. Our aim is to avoid

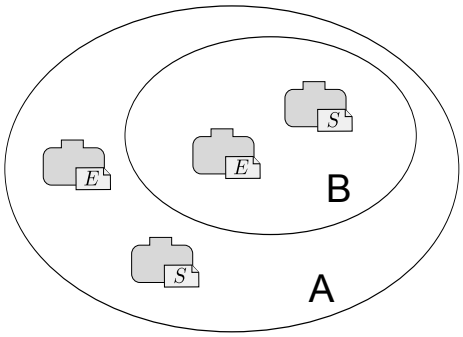


Figure 8.6: Example of compartimetalized space.

any interaction between the enzyme in the inner compartment and the substrate in the outer compartment, and vice-versa. In BlenX this can be achieved by simply duplicating

the descriptions of the enzyme and the substrate in the following way:

```
let E_A : bproc = #(x,siteE_A) [ rep x!().nil ];
let E_B : bproc = #(x,siteE_B) [ rep x!().nil ];
let S_A : bproc = #(x,siteS_A) [ x?().ch(x,siteP_A).P ];
let S_B : bproc = #(x,siteS_B) [ x?().ch(x,siteP_B).P ];
```

and setting compatibilities:

```
{ siteE_A, siteE_B, siteS_A, siteS_B }
%%
{
  (siteE_A, siteS_A, 0.05),
  (siteE_B, siteS_B, 0.05)
}
```

We establish that each sort of each box in a program contains the information of the location in which the boxes are placed in. In the example, indeed, box species of enzymes and substrates initially located in different compartments, differ in the specification of the location information contained in their sorts. Moreover, if the specification of the internal process contains change actions, we have to guarantee that, depending on the location of the protein, the sort is updated in the correct way, referring to the correct location. Referring to the design pattern, it is evident how this requires, at a first instance, the duplication of all the definitions of states and sorts in order to represent all the different protein conformational states in the different locations.

In this setting, moving a protein from a location A to a location B entails intuitively the update of all the sorts in the interfaces and in the internal process with the new location information. Obviously, we cannot use classical process calculi name substitution for this name, because sorts are not affected by it. Hence we extend the design pattern of signalling networks by adding a mechanism for moving proteins from one location to another. In doing this we always assume that the movement is generated by a sequence of immediate actions enabled after the consumption of an inter-communication over a special interface, used only for movement purposes.

Consider the space arrangement of Fig. 8.6 and imagine to initialize KKK (the version used in the simplified MAPK cascade model) in A. To guarantee that KKK can move across compartments A and B, we update the definition of the protein in the following way:

```
let KKKs1_A : pproc = recv?().ch(p,kaseKKK_A).ch(recv,baseKKK_A).s0_A!() +
                      Move<<s0_A,s0_B,plus>>;
let KKKs0_A : pproc = recv?().ch(p,outKKK_A).ch(recv,activeKKK_A).s1_A!() +
```

```
                      Move<<s1_A,s1_B,plus>>;
let KKKs1_B : pproc = recv?().ch(p,kaseKKK_B).ch(recv,baseKKK_B).s0_B!() +
                      Move<<s0_A,s0_B,plus>>;
let KKKs0_B : pproc = recv?().ch(p,outKKK_B).ch(recv,activeKKK_B).s1_B!() +
                      Move<<s1_A,s1_B,plus>>;

let KKK_A: bproc = #(p, kaseKKK_A), #(recv : inf, baseKKK_A), #(mov : inf, KKK_A)
  [
    rep s0_A?().KKKs0_A | rep s1_A?().KKKs1_A  |
    rep s0_B?().KKKs0_B | rep s1_B?().KKKs1_B  |
    KKKs0_A | rep p!(plus).nil |
  ];
```

where *#(mov,KKK_A)* is the new interface used only for movements and the template *Move* is defined in the following way:

```
template Move : pproc << name a, name b, name val >> =
  mov?(what).( what!() | A?().a!().recv!(val) + B?().b!().recv!(val) );
```

With the consumption of the inter-communication through the interface with subject *mov*, the box *KKK_A* receives a name indicating the destination location. Being *KKK_A* placed in the location *A*, then the template *Move* first activates the state *KKKs1_B*, the generator of the actual state in the destination location, and secondly executes the state change performing an output on *recv*. This leads to *KKK_B*.

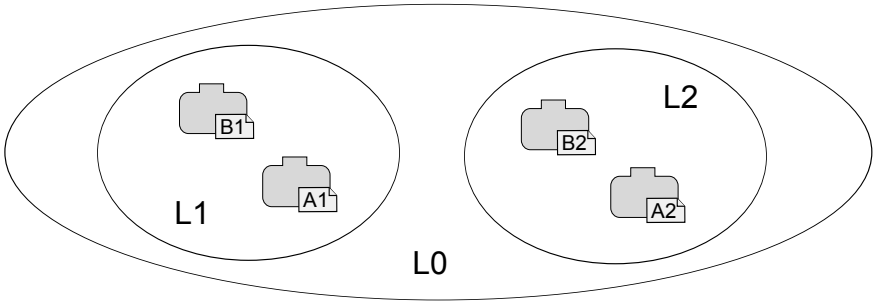
For completeness, the definition of boxes *KK* and *K* (not shown) are obtained similarly:

```
let KKs2_A : pproc = recv?().ch(p,kaseKK_A).ch(recv,intKK_A).s1_A!()+
                     Move<<s1_A,s1_B,plus>>;
let KKs1_A : pproc = recv?(what).what!() |
                     ( plus?().ch(p,outKK_A).ch(recv,activeKK_A).s2_A!() +
                     minus?().ch(recv,baseKK_A).s0_A!() ) +
                     Move<<s0_A,s0_B,plus>>;
let KKs0_A : pproc = recv?().ch(recv,intKK_A).s1_A!() +
                     Move<<s1_A,s1_B,minus>>;
let KKs2_B : pproc = recv?().ch(p,kaseKK_B).ch(recv,intKK_B).s1_B!()+
                     Move<<s1_A,s1_B,plus>>;
let KKs1_B : pproc = recv?(what).what!() |
                     ( plus?().ch(p,outKK_B).ch(recv,activeKK_B).s2_B!() +
                     minus?().ch(recv,baseKK_B).s0_B!() ) +
                     Move<<s0_A,s0_B,plus>>;
let KKs0_B : pproc = recv?().ch(recv,intKK_B).s1_B!() +
                     Move<<s1_A,s1_B,minus>>;
```

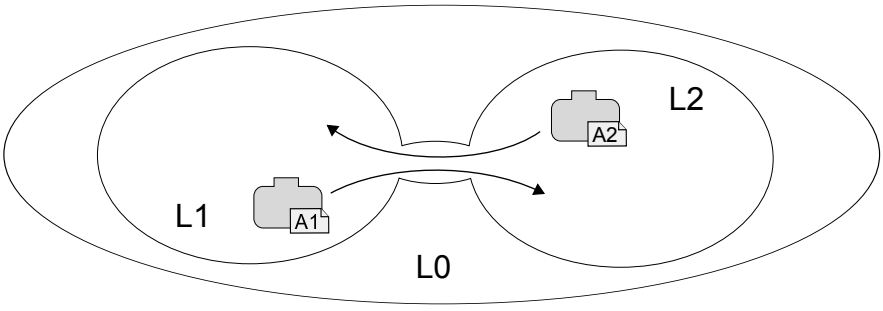
```
let KK_A: bproc = #(p, kaseKK_A), #(recv, baseKK_A), #(mov, sort_A)
  [
    rep s0_A?().KKs0_A | rep s1_A?().KKs1_A | rep s2_A?().KKs2_A |
    rep s0_B?().KKs0_B | rep s1_B?().KKs1_B | rep s2_B?().KKs2_B |
    KKs0_A | rep p!(plus).nil
  ];
```

Note that the sort in charge of the movement has been set to *sort_A*. This is due to the fact that, depending on the sort, the movement of box species (see below) can be more or less specific. Moreover, if we are interested in moving only a particular state of the protein, then the implementation of the specific state, in the state machine pattern, has to consider also the possibility to generate the movement.

The process of moving proteins around is usually called *protein targeting* or *protein sorting*. Protein sorting can mainly happen through secretory pathways or through non-secretory pathways following several different translocation phenomena. Following [93] we classify all these phenomena into a single abstract translocation process. The abstract translocation phenomenon can be seen as a door that can be opened by unlocking it simultaneously with two different keys. Note how this mechanism allows protein sorting to happen between any pair of locations. For example, consider the space arrangement:



and assume that proteins *B1* and *B2* are the key proteins in inactive conformational state that, once activated, allow the proteins *A1* and *A2* to translocate. When the two proteins are active, they create a kind of tunnel:



that allows the translocations of *A1* and *A2*. Note that this tunnel remains open until one of the two proteins is deactivated; when open, it continuously translocates proteins *A1* and *A2* at a specific rate.

In BlenX this mechanism is implemented by combining communication primitives and events. Always referring to our previous definitions, assume that *B1* and *B2* correspond both to *KKK*. What we want is that each time an instance of *KKK* is active both in *L1* and in *L2*, then the tunnel is activated. This mechanism is implemented with the event:

```
when (KKKp_L1,KKKp_L2::inf) join(Tunnel_A1_L0_A2_L1);
```

where the box *Tunnel_A1_L0_A2_L1* is defined as follows:

```
let Tunnel_A1_L0_A2_L1 : pproc =
  #(x1,T_L1),#(x2,T_L2),#(y1,activeKKK_L1),#(y2,activeKKK_L2)
  [
    rep x1!(L2).nil | rep x2!(L1).nil |
    y?1().nil + y2?().nil
  ];
```

The sorts file contains the following compatibilities:

```
( T_L1, A1_L1, 1.0 ),
( T_L2, A2_L2, 1.0 )
```

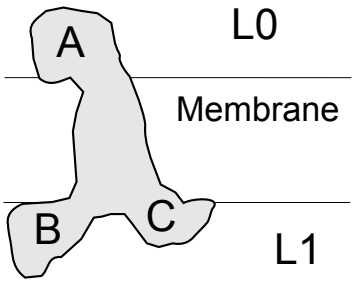
where we assume a basal translocation rate of 1 and movement sorts for proteins *A1* and *A2* equal to *A1_L1* and *A2_L2*, respectively. The tunnel is closed when the box *Tunnel_A1_L0_A2_L1* receives a deactivation signal, representing the deactivation of one of the two proteins. In particular, this is implemented through the following piece of code:

```
let Tunnel_A1_L0_A2_L1_close : pproc =
  #(x1,T_L1),#(x2,T_L2),#(y1,activeKKK_L1),#(y2,activeKKK_L2)
  [ rep x1!(L1).nil | rep x2!(L2).nil ];

when (Tunnel_A1_L0_A2_L1_close::inf) split (KKK_L1,KKK_L2);
```

Although translocations phenomena provide a great level of abstraction in modelling, the movement and rearrangement of proteins across compartments, a simpler mechanism can also be imagined and used in combination with it.

We can indeed imagine to represent transmembrane proteins and let them in charge of moving substances across compartments. Transmembrane proteins are bound to membranes spanning over both surfaces (integral proteins) or just residing on one side of the

membrane (peripheral proteins). Also without explicitly representing membranes, transmembrane proteins can be modelled as special proteins that cannot move, but can send activation and movement signals to the other proteins. The single critical case to determine the location of a site (whether it is exposed either towards the outside of the membrane or towards the inside of the membrane) is relative to integral protein. Indeed the exposition of the sites of integral proteins determines the set of components that can come in touch with the site due to the physical boundaries originated by the spatial hierarchical structure of cells. An example is the integral transmembrane protein in the following picture:



that has three domains: one directed towards the location *L0* and two directed towards the location *L1*. Following our design pattern, this protein can be implemented by the following box:

```
let Integral_protein : bproc =
  #(a,A_L0),#(b,B_L1),#(c,C_L1)
  [ ... ];
```

The location information in the sort respects our design pattern and represents exactly the situation of the previous picture. Note that the integral protein can still be designed following our design pattern, but we have to set compatibilities and output messages in such a way that the protein can send not only activation signals, but also movement signals (e.g., domain *A_L0*, once activated, can send to proteins in *L0* movement messages containing the message *L1*).

We want to conclude by discussing about the use of the presented space representation also in combination with the detailed description of signalling networks. Indeed, the detailed design pattern guarantees that while a box is complexed with some other box, all the interfaces of the boxes that are not involved in the binding are changed to inactive sorts. In this way, complexes cannot be involved in movements because they are only intermediate entities emerging from the dynamics. Hence, also the detailed design pattern can be easily updated to take into account spatial modelling aspects.

Chapter 9

Evolutionary framework

Cross fertilization between biology and computer science dates back to the '60s. Algorithms and tools that mimic evolution are known from the pioneering work of Fogel in 1966 [38] on *evolutionary algorithms*. Evolutionary algorithms are inspired by evolutionary biology concepts, like inheritance, mutation, selection and crossover. These algorithms have been applied to areas such as machine learning, optimization, search problems; however they soon lost their strict connection with biology and therefore brought advantages only to computer science.

Recently there is an interest in using *evolutionary approaches* to study networks. Understanding how networks emerged during evolution can help us to understand their basic properties, such as the role of complexity and the importance of topology and feedback loops. Historically approaches to study evolution are commonly based on comparative genomics or proteomics and on phylogenetic analysis [100, 4]. These studies compare networks from different organisms to infer how evolution affects the internal structure of the network of interactions. In the last years alternative approaches emerged that simulate evolution *in silico*, but differently from *evolutionary algorithms*, they mimic evolution in a very close and precise way [102, 83, 84, 39]. Up to now, these approaches have used ad-hoc tools and representations of network dynamics, usually based on mathematical models, without exploiting the capability of the new computational and conceptual tools of *systems biology*.

Here, we aim at blending evolution *in silico* with computational model based on systems biology oriented languages, rejuvenating the mutual enrichment between biology and computer science. We develop a framework to allow the study of network evolution based on the BWB. The great flexibility of BlenX in the definition of the structure of proteins allows us to introduce primitives for mutations used to build domain-based interaction and mutation models. Starting from the study of mutations at a biological level, we end up with some interesting program modifications that permit us to mutate the BlenX rep-

resentation of biological entities in a meaningful and automatic way. Moreover, network dynamics can be easily modelled, and the interactions of emerged networks analysed. In this chapter we describe the approach and we show the feasibility of the proposed solution by implementing it. We discuss a case study for the application of the resulting tool.

9.1 The framework

We propose a framework for simulating the evolution of networks *in silico*. It is based on the simplified design pattern for signalling networks presented in the previous chapter.

In order to simulate evolution by natural selection, we must be able to express populations of individuals, variability and fitness. A BlenX program represents an individual. A population consists of a set of different BlenX programs, each representing an individual composing the population. Evolution proceeds through selection acting on the variance generated by random mutation events. Individuals replicate in proportion to their performance, referred to as *fitness*.

This process can be modelled as shown in Tab. 9.1.

```
EvolutionAlgorithm ():
  Population := GenerateInitialPopulation();
  for i = 0 to generations do
    for each Individual in Population do
      output := Simulate(Individual);
      fitnesses[Individual] := ComputeFitness(output);
    NewPopulation := ReplicateAndMutate(fitnesses, Population);
    Population := NewPopulation;
```

Table 9.1: Generic EvolutionAlgorithm.

This algorithm differs slightly from the generic evolutionary algorithms used in computer science, being closer to real biological observations made for the asexual reproduction of organisms. Each individual in the population is codified using a BlenX program, and the boxes in each program are the abstraction of all the entities present in that individual. The interaction among these entities result in the behaviour of the network we want to study.

There are four main procedures in the algorithm:

- **GenerateInitialPopulation:** the initial population can be generated randomly, from a predefined network configuration to be used as a starting point, or it can be

a network with no interactions. All the individuals in the initial population can be equal at the beginning, as they will be differentiated later by the mutation phase.

- **Simulate:** each individual in the population is simulated separately using the BetaWB stochastic simulator.

- **ComputeFitness:** the output of the simulation is used to compute the fitness value of the current individual. Note that the fitness value is problem-dependent; for an example, refer to Sec. 9.3.

- **ReplicateAndMutate:** this is the most important part of the algorithm. Like in a real environment, individuals with the highest fitness values are more likely to survive, replicate and produce a progeny that resembles them, being not, however, completely equal to them.

```
ReplicateAndMutate (fitnesses, Population):
  for i = 0 to i < Population.Size do
    Individual := ChooseOneIndividual(Population, fitnesses);
    for each Protein in Individual.Proteins do
      if Random() < DuplicationProbabily then
        Protein2 := Protein.Duplicate();
        Individual.Proteins.Add(Protein2);
      for each Domain in Protein.Domains do
        if Random() < MutationProbability then
          MutationType := GetRandomMutation();
          if IsMutationFeasible(Domain, MutationType) then
            Domain2 := Individual.CompDomain(Domain, MutationType);
            Individual.Mutate(Domain, Domain2, MutationType);
      NewPopulation.Add(Individual);
    return NewPopulation;
```

Table 9.2: The ReplicateAndMutate algorithm.

The ReplicateAndMutate algorithm (Tab. 9.2) creates a new population with the same number of individuals of the current generation, using as a base the current individuals. At each step it chooses one individual, with probability proportional to its fitness (ChooseOneIndividual in the code above). This is achieved by constructing a cumulative probability array a from the *fitness* array, generating a random number in the

range $0...a[Population.Size]$, and then finding the index into which the random number falls.

The selected individual will replicate and pass to the next generation. During the replication, to each protein in the "genome" of the individual is given the chance to mutate, according to a probability.

A mutation is selected among all the possible types by the GetRandomMutation function, and this mutation is applied. Finally the individual, which can be either equal to its predecessor or mutated, is added to the new population. We now define in more detail the mutations that we consider in our framework.

9.1.1 Mutations

Here we consider the end-effects of point mutations occurring at the DNA level. These mutations ultimately affect network dynamics. For example, mutations in a DNA sequence can change the protein amino-acid sequence, leading to changes in its tertiary structure with implications on the affinity of this protein with other proteins or substrates. Similarly, events at DNA level as gene duplication or domain shuffling can alter network structure and dynamics.

A computer program which is used to mimic evolution of a species must implement random mutations in individuals during the replication as well. Here, we can easily implement these molecular processes using the domain and network model we discussed in Sect. 8.2.

We will take as an example the three-proteins network represented in Fig. 9.1(a) and we will illustrate how different mutations can be modelled in BlenX.

Duplication and deletion of proteins

Gene duplication at DNA level is implemented with a duplication of the box associated with the protein the gene codifies for. The new box will have a similar internal process and the same interface subjects, while interface sorts will be new but will have the same compatibilities. This is achieved by copying the affinities of the original binder identifiers. Duplication of sorts is needed because subsequent mutations on one of the interfaces of the duplicated protein must not affect the original one. Furthermore, since the new protein is a new distinct entity, it must not be structural equivalent to the original one. The same considerations hold for the internal processes: duplication and deletion of domains may lead to a modification of the internal structure (see next section); the internal processes must be duplicated so that each box has its own, distinct internal behaviour. Following the simplified design pattern, the boxes for the protein C:

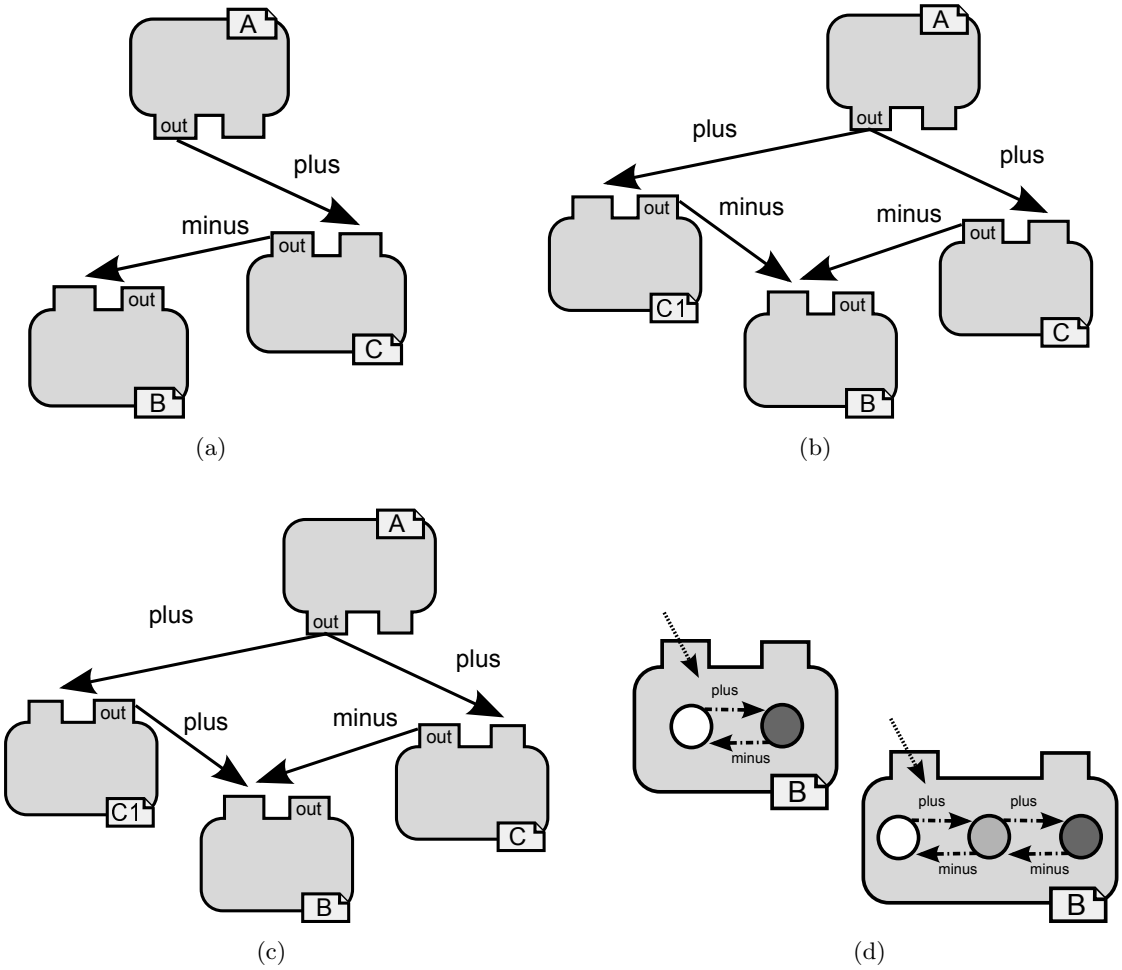


Figure 9.1: Different kinds of mutations: in (a) the initial configuration; in (b) duplication of protein *C* followed by mutation of domain *outC1* in (c). Finally, (d) displays how the internal structure could change to accommodate the duplication of a domain.

```
let Cs0 : pproc = recv?(what). ... ;
let Cs1 : pproc = recv?(what). ... ;

let C : bproc = #(p,kaseC),#(recv,baseC)
  [ Cs0 | rep s0?().Cs0 | rep s1?().Cs1 | rep p!(plus).nil ];
```

will be duplicated to:

```
let Cs0 : pproc = recv?(what). ... ;
let Cs1 : pproc = recv?(what). ... ;

let C : bproc = #(p,kaseC),#(recv,baseC)
  [ Cs0 | rep s0?().Cs0 | rep s1?().Cs1 | rep p!(plus).nil ];
```

```
let C1s0 : pproc = recv?(what). ... ;
let C1s1 : pproc = recv?(what). ... ;

let C1 : bproc = #(p,kaseC),#(recv,baseC)
  [ C1s0 | rep s0?().C1s0 | rep s1?().C1s1 | rep p!(plus).nil ];
```

Deletion of a protein is accomplished by deleting the associated box, the internal process it refers to and the appropriated entries in the sorts file.

Mutation of domains

Point mutations in DNA can change the protein amino-acid sequence, and consequently lead to the mutation of a domain and to changes in the interaction capabilities of the protein to which it belongs. In our formalism, this is achieved by changing the α function on the two domains that take part in the interaction. More specifically, the mutation on a domain can be a *change of interaction*, for which we modify the affinity adding a number sampled from a normal distribution, an *addition of an interaction* between two domains $d1$ and $d2$, modelled as the addition of a compatibility $(d1, d2, x)$, with $x > 0$, and finally a *removal of an interaction* between two domains $d1$ and $d2$ setting $(d1, d2, 0)$. For example, the mutation on domain *outC1* that can be observed in Fig.9.1(c) is obtained by changing the sorts file compatibility by changing $(outC1, activeB, x)$, with $x > 0$, to $(outC1, activeB, 0)$ and adding $(outC1, baseB, y)$, with $y > 0$; in this way the internal process of C is now allowed to send a $plus$ message when the B process is in an inactive state, represented by the interface sort $baseB$.

Duplication and deletion of domains

Domain duplication or deletion is more complex as it involves not only interfaces or rates, but requires also modification of the internal behaviour in response to stimuli.

Duplicating or removing domains can be easily done acting on the interfaces and on the sorts file; however, for these domains to act as sensing or effecting domains in cooperation or in antagonism with the existing ones, the internal process must also be changed. We devised several possible modifications of the behaviour when a domain is added.

As an example, consider the case of a sensing domain: when a signal arrives -by means of a ligand binding, or by phosphorylation of a residue- the internal process of the protein changes bringing it to a different *state*. If that domain is duplicated, the internal behaviour must be changed accordingly. The second domain may act *concurrently* with the old domain, with the result that the activation of this second domain will bring the protein in the same state as the old one, acting in parallel. This is the case, for example,

of a receptor that can bind to two different signal molecules. Alternatively, the duplicated domain can affect the capability of the protein to reach that state, and so must act *in coordination* with the original one; this is the case of kinases that must be phosphorylated twice to activate (double phosphorylation, as in Fig. 9.1(d)).

These mutations are obtained by manipulating the structure of the internal process to *transform* their behaviour. In both cases, we assume that the internal process have a standard structure, as described in Sec. 8.2. This process is built through parallel composition of different processes, each representing a different *state*. The set of processes in parallel is a set of *mutually exclusive* ones: at any given time only one of the processes can be active (e.g. not blocked waiting for a communication). Moreover, each process in the set *enables* another one by issuing a communication immediately before blocking itself.

```
let Bs1 : pproc =
   recv?().ch(p,kaseB).ch(recv,baseB).s0!();
let Bs0 : pproc =
   recv?().ch(recv,activeB).s1!().nil;
```

```
let Bs2 : pproc =
   recv?().ch(p,kaseB).ch(recv,intB).s1!();
let Bs1 : pproc =
   recv?(what).what!() |
      plus?().ch(p,outB).ch(recv,activeB).s2!() +
      minus?().ch(recv,baseB).s0!() ;
let Bs0 : pproc =
   recv?().ch(recv,intB).s1!().nil;
```

Figure 9.2: Transformation for the modification of a sensing domain, introducing a new *state*. In grey we highlight the modifications.

In the case of "cooperative" domains, where a signal on both is required to reach the desired internal configuration, the transformation can be accomplished by substituting the process codifying for the current active state with a new process, adjusting the channel names used for the intra-communications and binder identifiers accordingly. In Fig. 9.2, for example, it is shown how it is possible to manipulate an internal process to transform a protein activated (or deactivated) by a single phosphorylation into a protein that is activated (or deactivated) by a double phosphorylation, encoding an intermediate step of "half-activation".

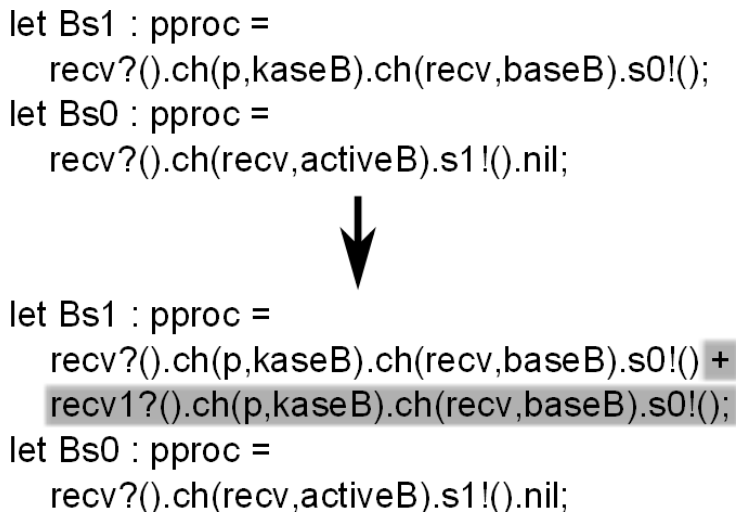


Figure 9.3: Transformation for the modification of a sensing domain. In grey we highlight the introduced actions.

The case of *concurrent*, or *competitive* domains, where each of the signals can lead to the desired internal configuration, can be handled in a similar way; in this case however the process representing the state is substituted with a different process (see Fig. 9.3).

Deletion of a domain requires to undo the steps done while duplicating it. This task is accomplished again by transformation of the internal process, restoring the behaviour to the original one.

9.1.2 Measure of fitness

When analyzing evolution of specific biological systems, one needs to consider the "fitness" benefit of that system to the organism (i.e. to its reproductive success). While it is usually complicated to define and measure such fitness contribution, network dynamics can provide a good proxy in case of biological networks. As the concentrations of the proteins involved in such networks will define the proper functioning of the network, how these concentrations fit a specific time course would determine how well the network "operates".

Here, we include some common operations that can be performed on concentration traces, and a way of finding entities based on their characteristics, such as the number and binder identifiers, or their state. This is important in a language like BlenX where the whole system, and all the entities that can appear in a simulation, are not specified in the program but can be generated dynamically during the simulation. We will illustrate later how fitness can be computed using the integration of a response.

9.1.3 Constraints

We understand that with our framework it is possible to generate countless combinations, interactions and mutations. Many interactions or mutation can be possible and make sense from the point of view of a program syntax and semantics, but have little or no sense from the biological perspective. We addressed this issue by providing a configurable way of specifying constraints on mutations, their probability and which class, or type, of protein or domain they can affect.

9.2 Implementation

We implemented our evolutionary framework using the BWB.

Each *individual* in our evolutionary framework is represented by a BlenX program. We used the BWB simulator to execute the models and obtain their time courses; we built a new tool to compute the fitness based on the simulator output files, and a new tool to manipulate and mutate the BlenX programs, based on the BWB libraries.

The challenging part was to implement mutations of BlenX programs. The first three kind of mutations introduced in Sec. 9.1 act on the sorts file. However, mutations of the fourth kind (duplication and deletion of domains) lead to changes in the internal behaviour. These changes are done directly on the program.

The BWB compiles the model *Just in Time*: the simulator takes the source code for the model and compiles it into an Abstract Syntax Tree (AST). This tree is the object model of the processes discussed in Sec. 9.1. *Transformations* discussed in that section are implemented by manipulating and navigating in a programmatic way the AST. Our libraries allow us to access the AST and write it to the disk, generating a new and perfectly valid BlenX program.

9.3 Case study

We use this simplified MAPK cascade presented in Sec. 8.2 as a starting point for testing our evolutionary framework. In particular, we want to analyse the evolution of a population according to a fitness function which captures the essential behaviour of our MAPK cascade model.

In detail, we generate an initial population of 500 individuals containing the network shown in Fig.9.4a. We set up very general initial conditions, with a single kinase K, a single phosphatase $P1$, an activation signal $E1$ and a deactivation signal $E2$; the model lacks any interactions among entities. In other words, we consider an ancestral organism that possessed all the base proteins but lacked a signalling system similar to the MAPK

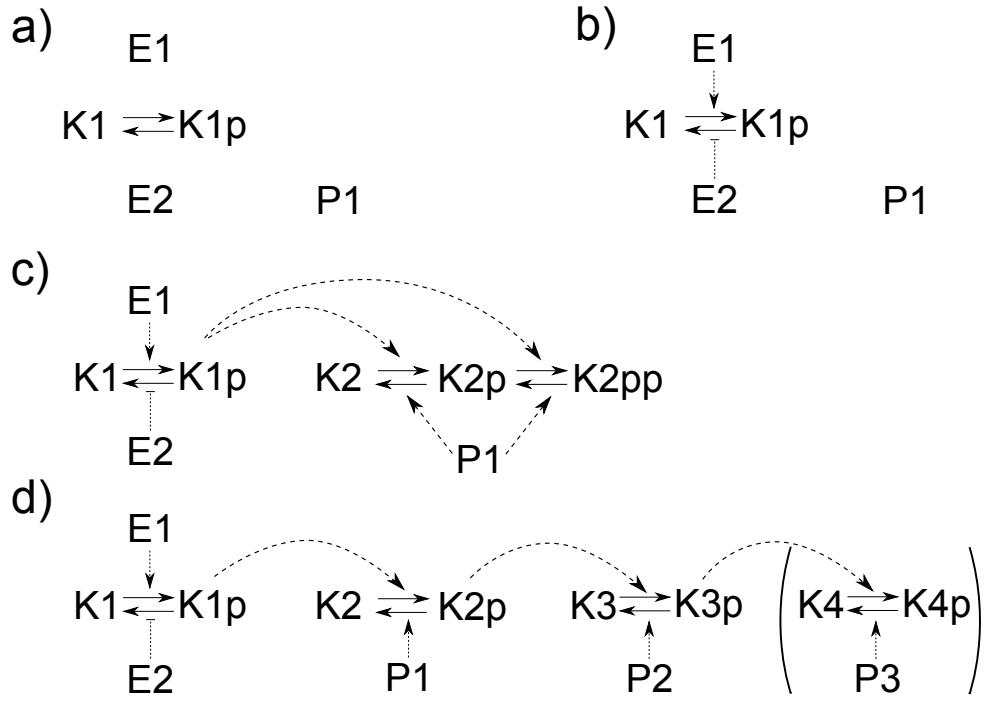


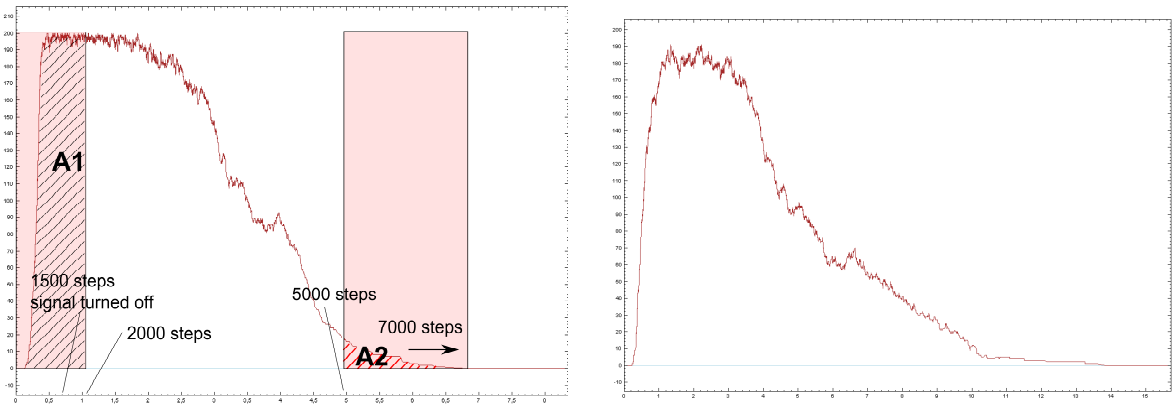
Figure 9.4: (a) Basic individual of the initial configuration. (b) Only signals $E1$ and $E2$ are enabled. (c) A particular individual we obtained, with a two-level phosphorilation. (d) An alternative evolution, with single phosphorilation kinases but a longer cascade.

cascade as observed today. The dynamic of each individual is then simulated; we run each individual for 7000 simulation steps and we remove the signal $E1$ at the step 1500 using a time-triggered *delete* event. Using the output of the simulation, we then measure for each individual the corresponding fitness.

The fitness function we implemented measures how rapidly the output of an active kinase increases, how much the output of the same kinase persists after removing the signal $E1$ before returning back to the initial condition. Let $out = \{n_0, n_1, ..., n_{7000}\}$ be the tuple representing the active kinase K^* dynamics in time of an individual, then the fitness for *out* is formally computed by the following formula:

$$fitness(out) = \mu + \left(\frac{\sum_{j=i_1}^{e_1} n_j}{K_M^* \times (e1 - i1)} - \left(\gamma \times \frac{\sum_{j=i_2}^{e_2} n_j}{K_M^* \times (e2 - i2)} \right) \right)$$

The two sums, that we denote respectively with $A1$ and $A2$, represent discrete integrals and are normalised with respect to their possible maximum values (see Fig.9.5). The values i_1, e_1, i_2 and e_2 are changeable parameters that define the boundaries for the computation of the two discrete integrals present in the formula, and the value K_M^* represents the maximum value for the K^* response.



(a) (b)

Figure 9.5: (a) Time course of the Kpp concentration over the simulation time, superimposed to the integral areas for the fitness function we implemented. The fitness parameters are $i_1 = 0$, $e_1 = 2000$, $i_2 = 5000$ and $e_2 = 7000$. (b) Time course of Kpp for a network with high fitness.

Moreover, μ represents the minimum fitness and γ controls the relative importance to responding to a signal and turning the response off after its removal. The reported results are for $i_1 = 0$, $e_1 = 2000$, $i_2 = 5000$, $e_2 = 7000$, $K^*_M = 200$, $\mu = 0.1$ and $\gamma = 0.75$.

According to the algorithms presented in the previous section, the population is evolved. In order to maintain a biological validity for the new individuals, possible mutations are the one that satisfies the following constraints: (1) signals $E1$ and $E2$ cannot be removed; (2) a kinase can only activate other kinases or itself; (3) kinases are specific (e.g. they do not phosphorilate multiple proteins); (4) phosphatases are not specific but can only deactivate kinases.

We iterated the evolution algorithm for 2000 generations, for different values of fitness function parameters. We then inspected the generated models using the Plotter and Designer tools introduced in Chapt. 7. The dynamic behaviour of one of the obtained networks is shown in Fig.9.5(b); examples of obtained individuals are in Fig. 9.4. In particular, we did not obtain individuals with a perfect MAPK cascade network, but individuals in (c) and (d) have very good fitness values and show the two directions in which evolution went to build an ultra-sensitive switch, namely forming longer cascades with multiple kinases or having multiple phosphorylation sites. We did not obtain in our runs individuals whose networks combined the two characteristics; we suspect that this fact may be due to the fitness function, that reached its maximal values with the two configurations in (c) and (d). As a final note, interactions within kinases and phospatases shown in (c) and (d) are only an example: we obtained also individuals with very complex relations (self-activations, "reverse" activations -where for example K activated KK and KK activated K- and so on).

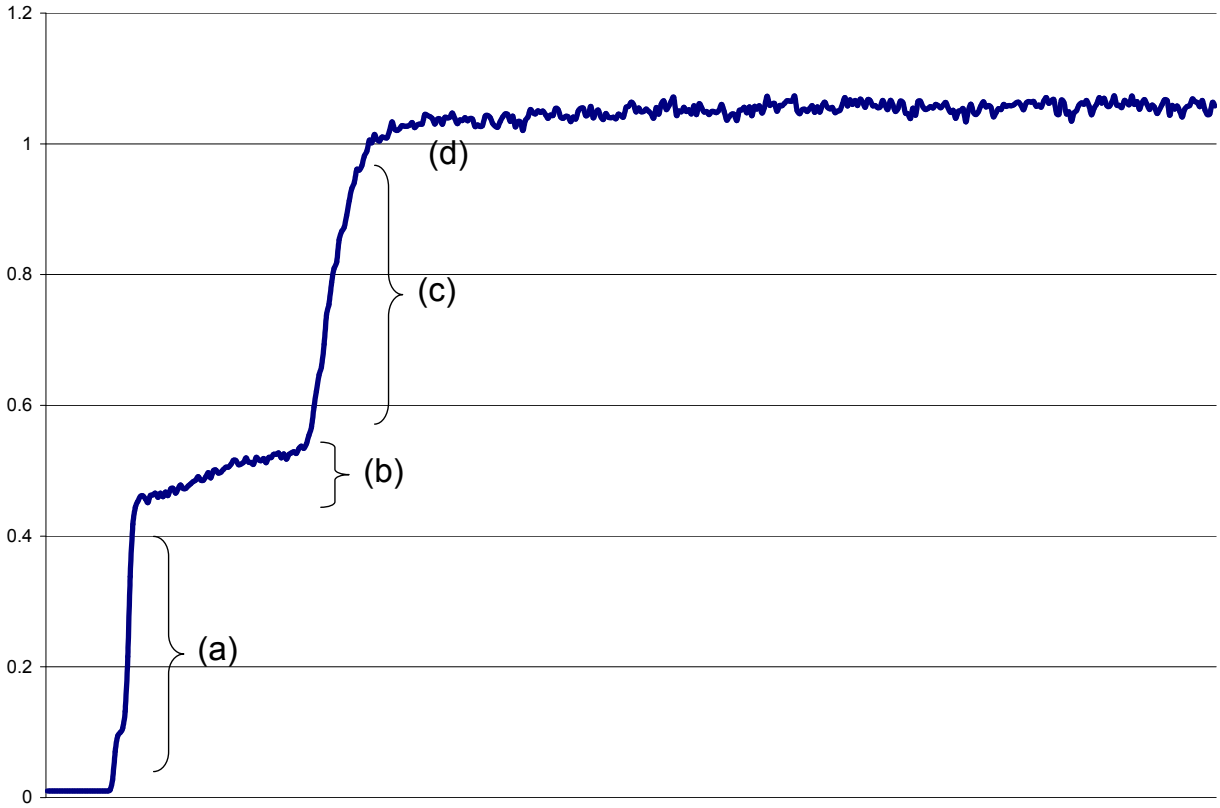


Figure 9.6: Changes in fitness during a typical evolutionary simulation.

The variation of fitness during a simulation is depicted in Fig. 9.6. Note the "steps" in the fitness. We observed this typical behaviour in almost all our runs. In the first generations, individuals have to find the correct signal: the *jump* in (a) is realized when the the activation signal $E1$ hits one of the kinases. In (b) instead we have the slow adaption to the introduction of the deactivation signal: the presence of the signal allows the cascade to be switched off, but reduces the gain of the switch in response to an activation signal. The second *jump*, in (c), is where double phosphorylations or more kinases are added to the cascade, allowing the network to re-gain the lost efficacy and react in a steep way to the activation signal. The last phase, (d), is where more phosphatases are added in order to switch off the response in a quicker way.

Chapter 10

Self-assembly

The term self-assembly indicates a process in which disordered components form an organized structure or pattern only through local interactions among them, without an external coordination. Debates of how complex structures and functions can emerge from local interactions between simple components can be found in plenty of different fields (e.g. nanotechnologies [31], robotics [58], molecular biology [44, 111], autonomous computation [78]). Here we concentrate on molecular self-assembly, the process through which molecules assemble in *complexes* without guidance from an outside source. We are particularly interested in inter-molecular self-assembly (i.e., the ability of proteins to form quaternary structures), a process which is crucial to the functioning of cells.

Conventional modelling approaches for molecular and systems biology (e.g., ODEs) represent molecules and complexes by associating them with species identifiers or variables. This representation is extremely simple and effective. However there are modelling scenarios in which it is not always feasible or even possible the explicit definition of all the possible complexes acting in a biological system and all the possible reactions in which they are involved. An example is self-assembly, a process in which small molecules combine to produce large complexes like chains or graphs of molecules (e.g., actin polymerization [2]).

The aim of this chapter is to show how the BlenX allows to model in an intuitive and effective way non-trivial structures, like filaments, trees and symmetric rings and how to introduce controls over their shapes.

10.1 Filaments and trees

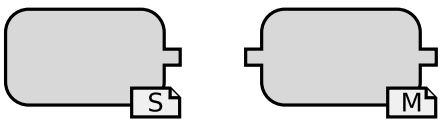
An important feature of polymers is their structure. The simplest structure is a *linear* chain, characterized by a single backbone with no branches. A related unbranching structure is a *ring* polymer. A *branched* polymer is composed of a main chain with one or

more side chains. Examples of polymers are actin, DNA and microtubules.

We exploit here BlenX programming to model linear and branched polymerization processes, namely the self-assembly of *filaments* and *trees*. Ring polymers are postponed to the next section. For the sake of simplicity, the models presented in this section can only grow, i.e., we do not consider reversible processes in which boxes can detach from a filament. Although this is a simplification the programs presented give the flavor of the BlenX potential. In all programs the only reactions associated with finite rates are the bindings between boxes. All the other actions (e.g. intra-communications, changes, complex-communications) are executed as immediate actions that manage the set of conformational changes caused by the formation of complexes. Conformational changes are assumed to be atomic with the complexation because they are so fast that the time they take to execute can be ignored.

10.1.1 Filaments

A polymerization process consists in the creation of big molecules starting from small molecules that can bind together. We start by presenting a simple scenario in which a box `M`, representing a monomer, can bind to boxes of its own species and generate filaments of boxes. For simplicity, and for a fine control over the creation of the filaments, we introduce also a *seed* box `S`. Its role is recruiting the first monomer and starting the formation of a filament. The graphical intuition of the structures of boxes `S` and `M` is given in the following picture:



Box `S` has only one interface, used to bind to a free monomer, while `M` is equipped with two interfaces, the *left* one used to bind to the last box of a growing filament (can be both a box `S` or `M`) and the *right* one used to bind to a free monomer. The dynamics of the model is depicted in Fig. 10.1 (arrows decorated with a star represents more then one step).

Notice that a filament can only grow on the right side. The seed `S` starts the creation of the filament, while the growing process involves only the binding of `M` monomers. A box `M` can accept the binding of another monomer on its right interface only if it is already part of a growing filament, i.e., its left interface is bound to a filament.

We start with the definition of the two boxes, `S` and `M`:

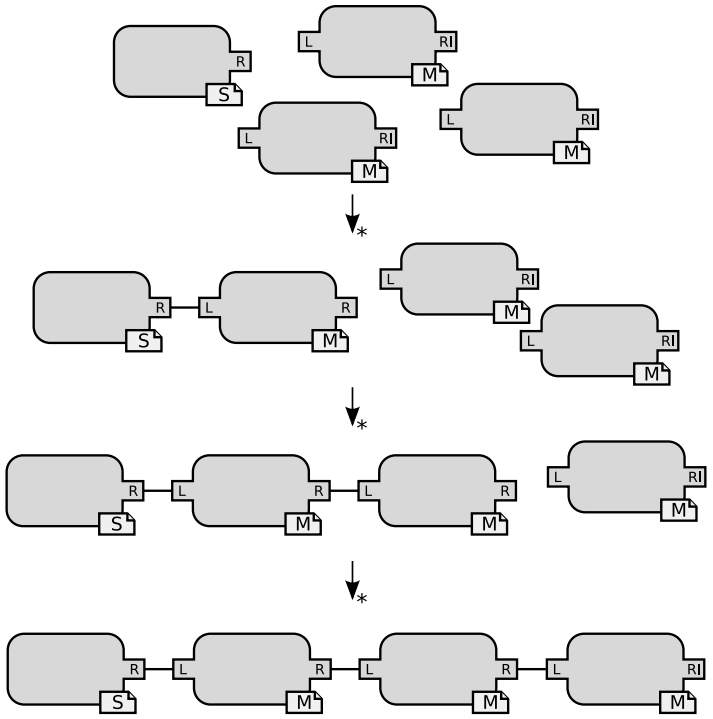


Figure 10.1: Example of filament formation.

```
let S : bproc =
 #(right,R)
 [ nil ];
```

```
let M : bproc =
 #(left,L), #(right,RI)
 [
   if (left,bound) then
     ch(right,R)
   endif
 ];
```

The interfaces definition file lists all the structures appearing over the interfaces of our boxes and specifies the compatibility `(L,R,1,0,0)`. Due to the compatibility between `L` and `R`, a monomer `M` can bind to the seed `S`. After the binding, the internal program of `M` recognizes that something is bound to the `left` interface and changes the structure of the right interface from `RI` (that stands for `R` inactive) into `R`. In this way, the `right` interface of the bound `M` gets the capability to bind to the left interface of another free monomer (see Figure 10.1).

10.1.2 Trees

Here we upgrade the previous program to create trees. We modify the monomer box `M` adding a third interface, the `branching` interface. Like the `right` interface, the `branching` one is activated via change actions only when `M` has something bound on its `left` interface.

```
let M : bproc =
   #(left,L), #(right,RI), #(branching,BI)
   [ if (left,bound) then ch(right,R).ch(branching,B) endif ];
```

We add also a box `T`. It has two interfaces and its role is to bind to the `branching` interface of a monomer in a filament and to start the formation of a branch. This species is introduced because we want the rate of branches formation over filaments to be independent from the number of free monomers in the model. `T` is defined as:

```
let T : bproc =
   #(left,TL), #(right,TRI)
   [ if (left,bound) then ch(right,TR) endif ];
```

Also for `T`, the internal program is instructed for changing the structure of the `right` interface once the box gets bound on the `left` interface. For trees, we specify two additional affinities: `(B,TL,1,0,0)` enables the complexation of a box `M` that is part of a filament with a free box `T`; `(L,TR,1,0,0)` enables the complexation of a box `T` that is bound to a monomer with a free monomer `M`. A possible run of this program with one box `S`, three boxes `M` and one box `T` is depicted in Fig. 10.2.

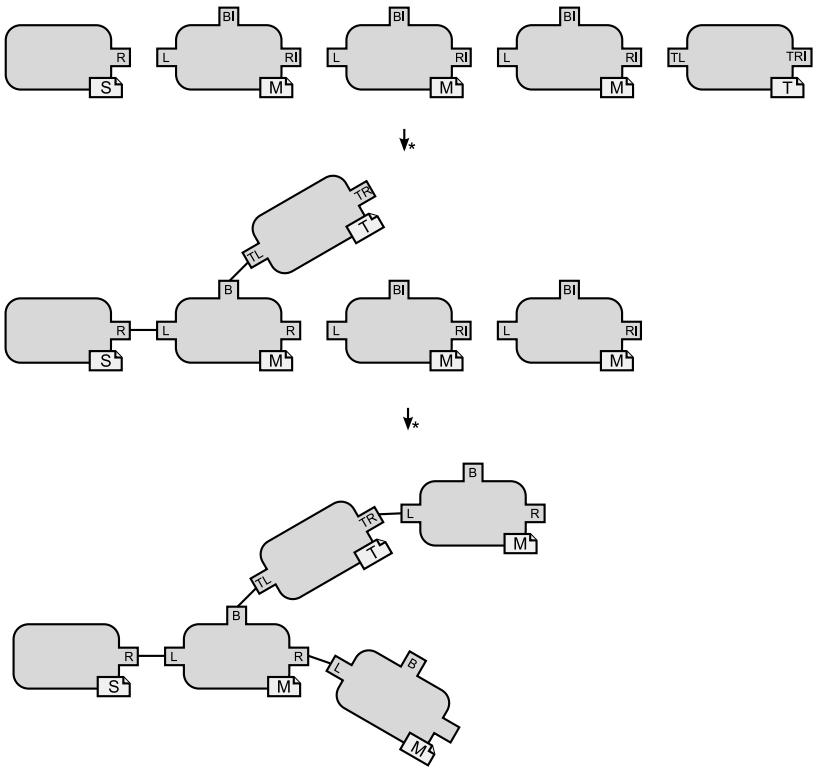


Figure 10.2: Generation of a branching tree.

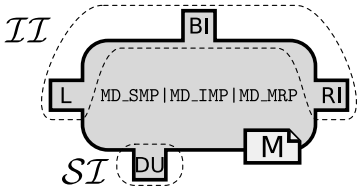
10.1.3 Introducing controls

We introduce a program to build trees with constraints over the depth of the branches. This programs shows how communication and immediate actions can be used to control the shape of the self-assembling structures. The design patterns and ideas we introduce can be used to model several similar kind of shape formation controls.

Each node of a tree can be associated with a branching depth level. A node has branching depth level n if there are exactly $n-1$ branches from the node to the root. A tree has branching depth less than n if all its nodes have branching depth level smaller than n.

We define a BlenX program that generates trees with branching depth less than five. We do it by introducing a new design patter. For each box we partition the interfaces in *interaction interfaces* and *state interfaces*, denoted by $\mathcal{II}$ and $\mathcal{SI}$, respectively. Interaction interfaces are used by a box to interact with other boxes, while the state interfaces are used to store information regarding the box state. Note that storing information in the interfaces simplifies the internal coding; indeed all the parallel processes can access this information easily by managing and checking interface structures through change actions and if statements. We call them *interaction modifier process* (IMP), *state modifier process* (SMP) and *messages receiver process* (MRP).

The boxes involved in this model are monomers, branches and seeds. The BlenX graphical and textual representation of monomers is:




```
let M : bproc =
 #(right,RI), #(left,L),
 #(branch,BI), #(depth,DU)
  [ MD_SMP | MD_IMP | MD_MRP ];
```

The only interface belonging to $\mathcal{SI}$ has subject `depth` and can be associated with five different structures: `DU`, `D1`, `D2`, `D3` and `D4`. Structure `DU` is used when the box is free, while the others are used when the box is bound and represent the node branching depth level. We refer to this interface as *depth* interface.

The *interaction modifier process* `MD_IMP` changes the structure of the right interface when the box binds on the left interface and changes the structure of the branching interface depending on the structure of the depth interface. In particular, the structure has to be changed in order to let the box branch only if the structure exposed on the depth interface is one among `D1`, `D2` and `D3`. Hence, `MD_IMP` is defined as follows:

```
let MD_IMP : pproc =
   if (left, bound) then ch(right,R) endif
   | if not ((depth,D4) or (depth,DU)) then ch(branch,B) endif;
```

We construct the *messages receiver process* MD_MRP mechanically, composing in parallel similar processes, one for each interface belonging to $\mathcal{II}$:

```
let MD_MRP : pproc =
   rep right?(channel).channel!()
   | rep left?(channel).channel!()
   | rep branch?(channel).channel!();
```

The last process composing the internal program is the *state modifier process* MD_SMP. Its role is to modify the structure associated with the depth interface in order to make it represent the number of branches from the root to the current box. This task is performed through an exchange of messages among the *state modifier* processes of different boxes. Thus MD_SMP is composed by two parallel processes, one receiving and using messages for updating the state interfaces, and another sending messages to bound boxes and trigger their update:

```
let MD_SMP : pproc =
   MD_r_SMP | MD_s_SMP
```

MD_r_SMP defines a sequential process composed by an input on a channel called depth_msg which guards a sum of inputs over specified channels whose firing enables proper modifications of the depth interface:

```
let MD_r_SMP : pproc =
   depth_msg?().(
      d1?().ch(depth,D1)
      + d2?().ch(depth,D2)
      + d3?().ch(depth,D3)
      + d4?().ch(depth,D4)
   );
```

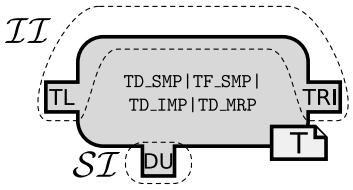
The interacting partner of MD_r_SMP is:

```
let MD_s_SMP : pproc =
     right!(depth_msg).(
      if (depth,D1) then right!(d1) endif
      + if (depth,D2) then right!(d2) endif
      + if (depth,D3) then right!(d3) endif
      + if (depth,D4) then right!(d4) endif
     )
   | branch!(depth_msg).(
      if (depth,D1) then branch!(d1) endif
      + if (depth,D2) then branch!(d2) endif
      + if (depth,D3) then branch!(d3) endif
     );
```

This process is composed by two parallel processes. The output on `right` can be consumed only when the box binds to another box on the corresponding right interface. When consumed, the output sends the name `depth_msg` to `MD_MRP` of the bound box, and it is propagated inside the box by an output on `depth_msg` without object. This output synchronizes with the `MD_r_SMP` process of that box and enables a sum of input processes on channels `d1`, `d2`, `d3` and `d4`. The choice operator is resolved according to the next message sent by `MD_s_SMP` and only one input is fired. The initial output of `MD_s_SMP` enables a sum of mutually exclusive `if`, each one sending through the right interface a different channel depending on the depth interface structure. `MD_MRP` sends a message on this channel that synchronizes with one of the inputs of `MD_s_SMP` causing the properly update of the depth interface. Notice that this communication protocol propagates the depth information to the newly attached box and is executed atomically with a sequence of immediate actions.

The `MD_s_SMP` process performs a sequence of immediate actions on the branch interface, similar to the ones described for the right interface, to propagate the depth information also along the branches. In this case the depth `D4` case is not considered because a branch cannot grow from a monomer with branching depth equal to four.

Now we proceed with the description of the branching box `T`:




```
let T : bproc =
 #(right,TRI), #(left,TL), #(depth,DU)
  [ TD_SMP | TD_IMP | TD_MRP ];
```

Also in this case the interfaces of the box are the same we introduced in the previous section. The interaction modifier process `TD_IMP` has only to change the right interface structure when the box binds with another box on the left:

```
let TD_IMP : pproc =
   if (left, bound) then ch(right,TR) endif;
```

The message receiver process `TD_MRP` is defined mechanically following the pattern previously described.

The state modifier process `TD_SMP` propagates the depth information to the new filament nodes. In this case, given that it is a branch node, we increase the depth level and we do not consider the depth level one (`D1`), because the branch node contributes to its depth level and so it has at least depth level two:

```
let TD_SMP : pproc =
```

```
depth_msg?().(
   d1?().ch(depth,D2)
   + d2?().ch(depth,D3)
   + d3?().ch(depth,D4)
)
| right_site!(depth_msg).(
   if (depth,D2) then right!(d2) endif
   + if (depth,D3) then right!(d3) endif
   + if (depth,D4) then right!(d4) endif
);
```

The seed box S does not have any state interface and its BlenX graphical and textual description is:


```
let S : bproc =
 #(right,R)
 [ SD_SMP | SD_IMP | SD_MRP ];
```

The interaction capabilities of this box never change and therefore the interaction modifier process is the empty process `nil`. The state modifier process `SD_SMP` has only to propagate the initial depth information when a filament is started, i.e., the first box it binds to is informed that its depth level is one:

```
let SD_SMP : pproc =
   right!(depth_msg).right!(d1);
```

Fig. 10.3 shows an example of computation of the branching depth control program.

10.2 Rings

Here we investigate how BlenX can be used to model the formation of ring polymers. In this section we do not make use of the previously defined design pattern because we want to underline the flexibility of the language in allowing different programming styles. Here we introduce also *reversibility*, an important characteristic of many biological processes omitted in the previous modelling scenarios.

We start our ring modelling by pointing out an example that shows which are the difficulties we have to face. Consider a scenario in which we have a population of boxes `A` equipped with two distinct interfaces with structures `L` and `R`. If we specify a binding affinity for `L` and `R` structures, initialize the system with an initial amount of boxes `A` and run stochastic simulations, we can observe in the output of the simulations the formation of various kinds of complexes as depicted in Fig.10.4. The figure reveals the

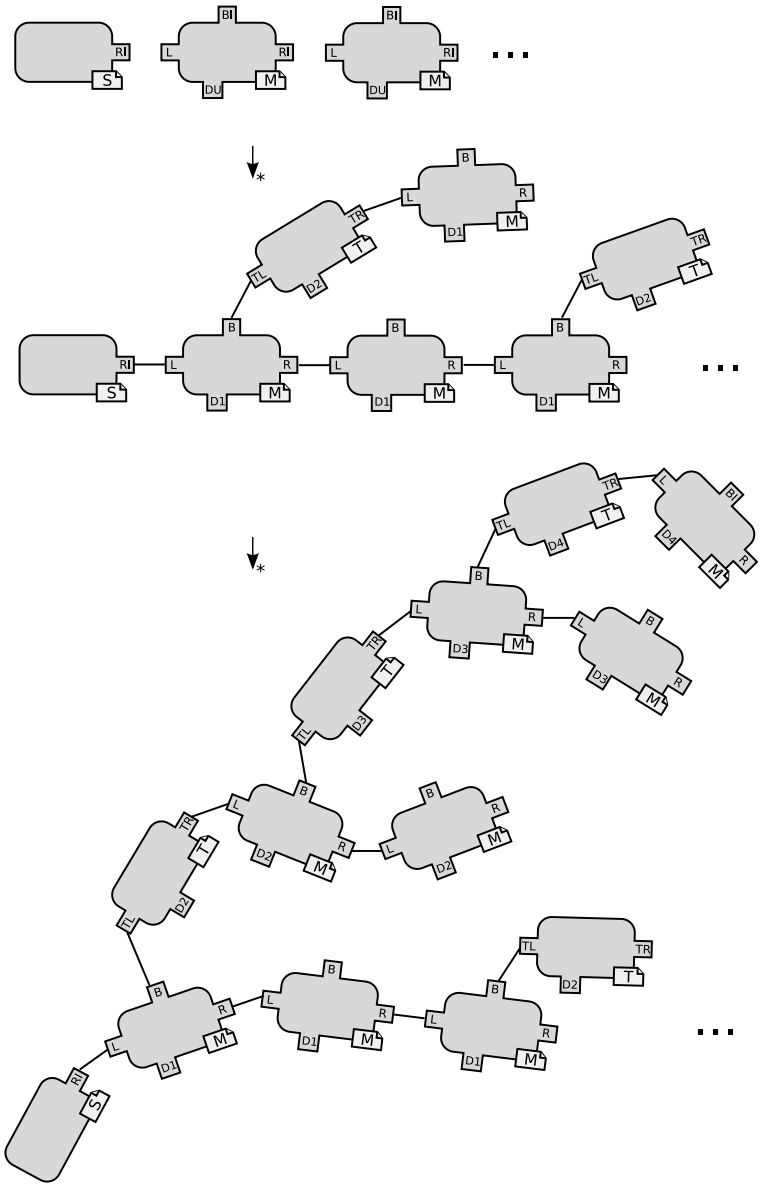


Figure 10.3: Example of a tree generated with the branching depth control program. Notice how at level four the monomer branching interfaces are inactive. The dots indicate the presence of other boxes in the system.

difficulties we have in controlling the formation of non-trivial complexes by only acting on the specification of interface capabilities (similar aspects and problems have been discussed in [22] for the κ-calculus). In this section we define a model that generates rings of fixed size avoiding the creation of long chains. We start without introducing reversibility and then we extend the model in order to introduce this capability.

Assume to have an initial population of boxes `A`, each containing the same internal

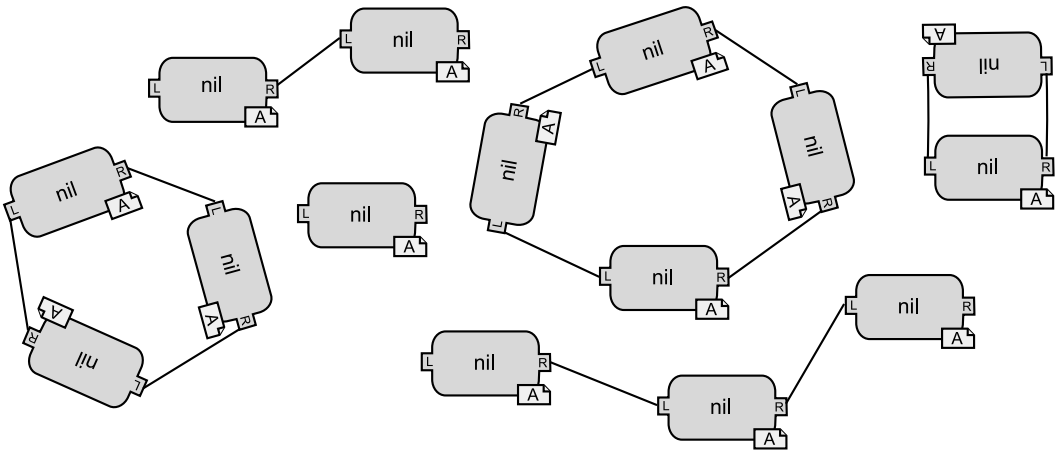
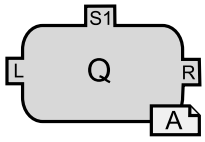


Figure 10.4: Examples of ring complexes.

program. Each box has an interface with structure `L` (left) and an interface with structure `R` (right), which are compatible, i.e. `(R,L,1,0,inf)`. Moreover, each box has an interface used to store the information regarding the length of the chain to which the box belongs; initially, the structure of that interface is `S1`:



Boxes `A` can bind together, forming chains that grow till they reach a size in which they close, forming a ring. We consider the ring being the stable form. The size of the ring can be controlled at programming level. As a characteristic of self-assembly, we want rings to be only the result of local interaction between boxes, avoiding any global coordination.

Each box can be in one of these states: not bound to any other box; bound either on the left or on the right; bound both on the left and on the right. Hence, the internal program implements a state machine with the above three states. We start from the design pattern introduce in Chapt. 8 and refine it. In this case, indeed, a state change is controlled by the structure of the box interfaces. After any binding operation, the involved boxes start a protocol made up of a sequence of immediate actions, through which they exchange information (by using inter-communications) used to change the internal process structure and their interface structures. The definition of `A` is:

```
let A : bproc =
    #(left,L), #(right,R), #(num,S1)
    [ Bound0 | rep b0?().Bound0 | rep b1?().Bound1 | rep b2?().Bound2 ];
```

Processes `Bound0`, `Bound1` and `Bound2` implement the three different states of our state machine. In order to implement a recursive behaviour, we use the replication operator. For example, entering the state `Bound1` is implemented by performing an internal communication on the channel `b1`.

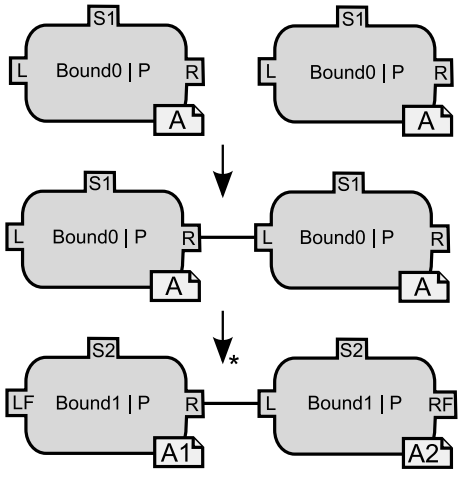
The process `Bound0` is defined by composing in parallel a sum of sequential processes performing inputs on the left and right interfaces and a sum of sequential processes performing outputs of the name `one` on the left and right interfaces:

```
let Bound0 :  pproc =
 left?(val).ch(right,RF).Set
 + right?(val).ch(left,LF).Set
 | left!(one)
 + right!(one);
```

```
let Set :  pproc =
 val!() |
 one?().ch(num,S2).b1!()
 + ...
 + n-1?().ch(num,SN).b1!();
```

After consuming the left input, the structure of the free right interface is changed from `R` to `RF` (where `RF` stands for `R` final). Similarly, after consuming the right input, the structure of the free left interface is changed from `L` to `LF`. Since `RF` is compatible only with `L`, and `LF` is compatible only with `R`, this means that we allow the chains to grow only by adding single boxes and not chains of boxes. Moreover, the non compatibility of `RF` and `LF` avoids a chain to close in a ring. Both inputs enable the process `Set` that changes the `num` interface according to the information received from the input. Notice that the value `N` reflects the length of the self-assembly rings we are programming (e.g. is equal to `3` in case of triangles) and that we cannot use it as a variable in the current version of BlenX. In order to obtain a runnable code we have to substitute it with a name respecting the actual size of rings we intend to create.

To better understand this mechanism, consider what happens when two initial boxes bind together to form a chain of length two:



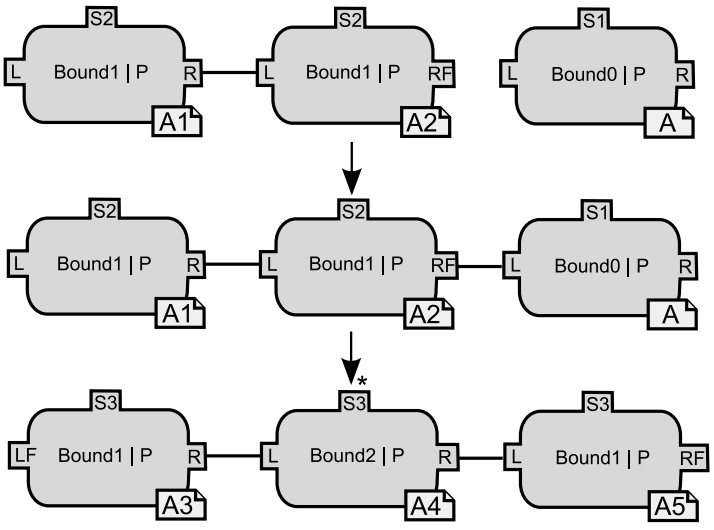
After binding together, the two boxes start a sequence of immediate actions that lead to the last complex depicted in the previous figure; each box has recorded in the structure of its interfaces that belongs to a chain of length 2 and the left and right limits of the chain have interface with structures `LF` and `RF`, respectively. Moreover, the two boxes are now in the state encoded by the process `Bound1` that embodies different controls composed with the choice operator. The first control process manages the flow of information incoming from the right interface, when its structure is `RF`:

```
if (right,RF) then
    right?().ch(right,R).left!().b2!().SIC_right
endif
```

The `if` condition is satisfied when a free box binds on the right. After consuming the right input, the process changes the right interface again into the structure `R`, propagates a signal on the left, activates the process `Bound2` and finally starts the process `SIC_right` that sends on the right the actual size of the chain and increments its own size:

```
let SIC_right : pproc =
    if (num,S2) then x!(two).ch(num,S3) endif
    + ...
    + if (num,SN-1) then x!(n-1).ch(num,SN) endif;
```

where `N` is the size of the rings we want to create. Notice that also in this case we use names to represent interface structures, and hence with `n-1` we mean the name representing the structure denoting size `N-1` (e.g. if `N` is 4 then `n-1` is `three` and `SN-1` is `S3`). To better understand this mechanism, consider the following graphical representation:



A single box binds on the right side of a chain of length two. After the binding creation, a sequence of immediate actions leads to the creation of the last complex depicted in the figure. Notice that the box in the middle now runs the process `Bound2`, that the external boxes run the process `Bound1` and that all the three boxes record the information on the length of the chain. This is obtained by creating a flow of immediate communications from `A2` to `A` passing through `A1`. In particular, using the second control process embodied in the process `Bound1`, `A1` changes its `num` interface after receiving a signal from the right and restarting `Bound1`:

```
if (right,R) then
  right?().b1!().IncreaseCounter
endif
```

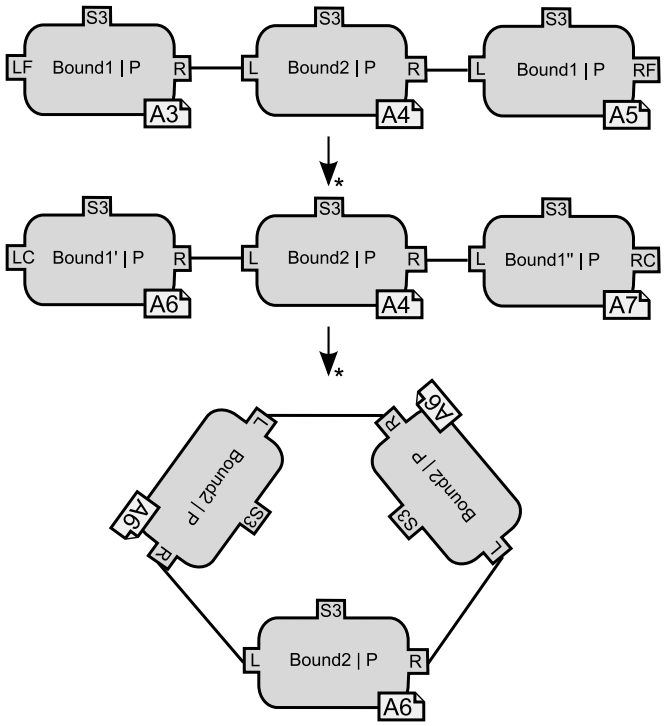
The process `IncreaseCounter` changes the `num` interface with a structure that represents the actual size incremented by one:

```
let IncreaseCounter : pproc =
    if (num,S2) then ch(num,S3) endif
    + ...
    + if (num,SN-1) then ch(num,SN) endif;
```

Specular control processes for the left interface are embodied in `Bound1`. Moreover, other two control processes are part of `Bound1` to control the growing of the chain on both sides and to close the chain in a ring when it reaches the appropriate length. The process controlling the right size has the form:

```
if (right,RF) and (num,SN) then ch(right,RC).
    if (right,bound) then ch(right,R).b2!() endif
endif
```

If the right interface has structure `RF` and the `num` interface has structure `N`, then the right structure is changed into `RC` and the process is blocked till the right interface is bound. Since the process controlling the left side is specular, when the chain reaches a length of `N` the right and left side interfaces assume structures `RC` and `LC`, respectively, and remain blocked till they bind together. However, since the compatibility of these two structures is defined as `(RC,LC,inf,0,0))` they immediately close and the two blocked processes can continue their execution, leading to a complex made up of structurally equivalent boxes and hence representing a completely symmetric ring. As an example, if `N` is equal to `3`, when a chain of size `3` is formed it immediately closes in a symmetric triangle:



Note that although in the triangle formation the state `Bound2` is not important, it becomes essential in the formation of rings of size greater then three. Indeed, the process `Bound2` has the structure

```
let Bound2 : pproc =
    right?().left!().b2!() + left?().right!().b2!();
```

and is in charge to propagate signals from the left to the right and viceversa. Note that in our actual implementation each signal received by a box in state `Bound2` cause only the change of the `num` interface with a structure corresponding to the increment of the current one.

Consider the previous program with `N` equal to 4. By running 100 different stochastic simulations with an initial population of 1000 boxes `A` we observe the average dynamics and the standard deviation reported in Fig. 10.5.

The results show that given an initial population of boxes, we reach a final configuration in which we have almost the same number of chains of size 3 and rings of size 4, and certain number of chains of length 2. We would like instead the system to generate as many rings as possible, because we assume rings be the only stable complex. In order to achieve this behaviour, we have to encode reversibility in our program, i.e., chains of size 2 and 3 can always disassembly.

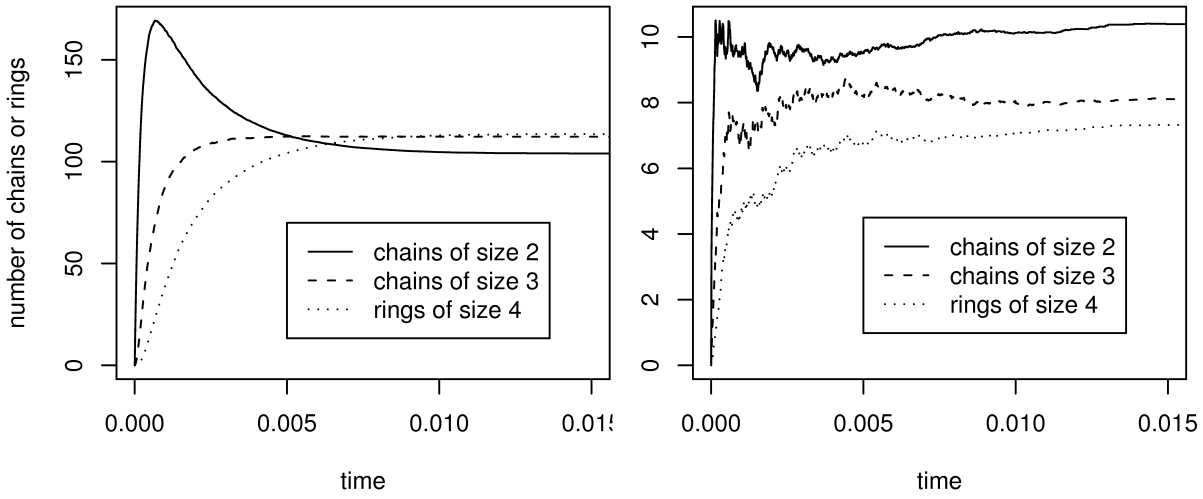


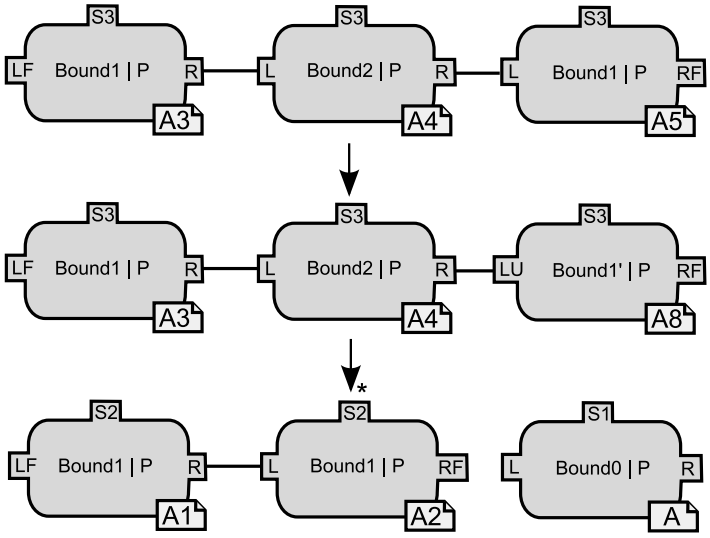
Figure 10.5: Average dynamics and standard deviation of 100 stochastic simulations regarding the formation of rings of size 4. Each simulation takes in average around 2 seconds.

Adding reversibility

We add reversibility by allowing only the unbinding of boxes located at the left and right sides of a chain. In order to implement this protocol we have first to modify the definition of the process `Bound1`, because it is the state characterizing the boxes located at the sides of a chain.

```
if (left,LF) then ch(r,right,RU).b1!() endif
+ if (right,RF) then ch(r,left,LU).b1!() endif
+ if (not (left,bound)) and (not (right,bound)) then
    ch(left,L).ch(right,R).ch(num,S1).b0!()
  endif
```

where `r` is a rate describing the speed of the reversible reaction. Each chain side can decide to unbind by changing the structure of the bound interface (left interface for the right side and viceversa). After executing the change action the box is immediately detached from the chain, because pairs `(LU,R,0,inf,0)` and `(RU,L,0,inf,0)` are defined in the compatibilities specification. After the unbinding, for both sides, the condition of the third `if` becomes true and hence the interfaces and the internal programs of the unbound boxes return back in the initial configuration (recall the recursive behaviour of box A):



The last complex of the picture shows how the remaining chain also returns back in a consistent configuration, where the boxes have the correct internal state and their interfaces `num` have the correct structures. In order to obtain this result, we have to modify the processes `Bound1` and `Bound2` as well. In particular, `Bound2` becomes a sum of processes, where one process controls the binding status of the right interface and recognizes when there is an unbinding:

```
if not(right,bound) then
    ch(right,RF).left!(dec).b1!().DecreaseCounter
endif
```

and another process does the same for the left interface (the `DecreaseCounter` process is very similar to the `IncreaseCounter` one but decrements the structure of the `num` interfaces). The process propagates on the left a name `dec`. This means that with respect to the program without reversibility here we propagate in the complex not only a signal, but names `inc` (increasing of one in the length) and `dec` (decreasing of one in the length). All the processes in `Bound1` and `Bound2` in charge of capturing and processing the communications are adapted accordingly. Indeed, in the `Bound2` sum, we have a process in charge to process and propagate an output from the right:

```
right?(act).left!(act).(
    act!().b2!()
    | inc?().IncreaseCounter + dec?().DecreaseCounter
)
```

and another for processing and propagating an output from the left. Notice that, depending on the received name, the increasing or decreasing process is enabled and process

`Bound2` is enabled again. Similarly, `Bound1` has an alternative in charge to process and propagate an output from the right

```
if (right,R) then
    right?(act).(
        act!().b1!()
        | inc?().IncreaseCounter + dec?().DecreaseCounter
    )
endif
```

and another from the left interface. Also in this case, depending on the received name, the increasing or decreasing process is enabled and the process `Bound1`, representing the actual box state, is enabled again.

Also in this case we consider the previous program with `N` equal to 4 and we run stochastic simulations with an initial population of 1000 boxes `A`. We can observe the average dynamics and standard deviation in Fig. 10.6. Note that by adding reversibility we have that the number of rings continue to increase, till no more rings can be formed.

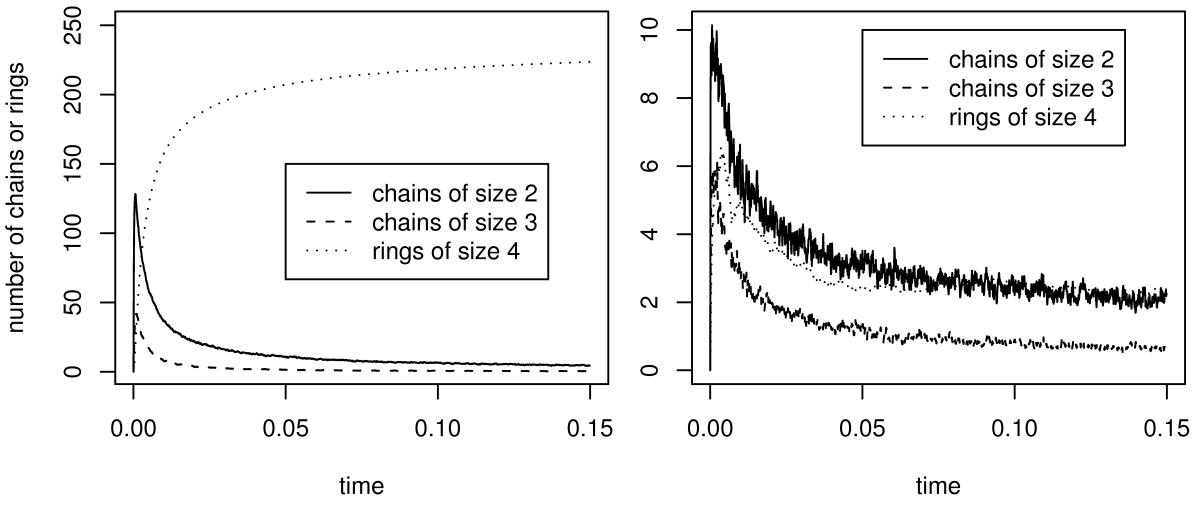


Figure 10.6: Average dynamics and standard deviation of 100 stochastic simulations regarding the formation of rings of size 4 with reversibility. Each simulation takes in average around 10 seconds.

Chapter 11

Conclusions

In this thesis we presented BlenX, a modelling language specifically designed to account for the complexity of protein-protein interactions.

BlenX is inspired by Beta-binders, a calculus explicitly thought to model biological systems. As well as for Beta-binders, BlenX abstracts biological entities as boxes composed by an internal *process unit* and a set of *interfaces*. Interfaces give boxes the capability to interact among them. As in Beta-binders, boxes interactions are driven by a notion of compatibility between interfaces. The four main features characterizing BlenX are the notion of events, the ability of boxes to form complexes, the use of global priorities and the presence of conditions.

The combination of these ingredients give to BlenX, and to its stochastic extension sBlenX, noteworthy modelling capabilities. At a first glance the language allows to describe signalling networks at different levels of details. Beyond the classical representation of proteins in active and inactive states, indeed, we can work on protein representations that distinguish between sensing domains, effecting domains and internal structures. This permits not only a detailed representation of proteins conformational states, but also a more effective description of protein state changes (e.g., activation after reception of multiple signals). Beyond the representation of signalling networks, the language allows for an effective description of other kind of scenarios characterized by a massive combinatorial complexity. Among these we identified the modelling of self-assembly processes, in which the number of possible protein complexes and combinations of protein modifications tends to increase exponentially.

From a theoretical point of view, we studied the expressivity of different subsets of BlenX. Using the theory of well structured transition systems, we showed that for a core subset of the language (which considers only communication primitives) termination is decidable. Moreover, we proved that by adding either global priorities or events to this core language, we obtain Turing equivalent languages. The proof is through encodings of

Random Access Machines (RAMs), a well known Turing equivalent formalism, into our subsets of BlenX. Moreover, we provided efficient procedures for establishing whether two boxes are structurally congruent and whether two graphs of boxes (two complexes) are isomorphic.

In particular, these two last results allowed us to implement an efficient variant of the Gillespie's algorithm, the stochastic engine kernel of the Beta Workbench. This software tool permitted us to test the language over several biological case studies, showing the effectiveness of the approach.

We first investigated the modelling of signalling networks. We presented different models and design patterns, of increasing complexity, of the MAPK cascade. We then discussed how to manage the composition of different signalling networks models and finally showed how to add spatial aspects to our design patterns.

Then, we proposed a framework for simulating the evolution of protein-protein interaction networks, where evolution of a population is implemented with an evolutionary algorithm which works in four main parts and is iterated for a specified number of steps; each iteration is called *generation*. Populations are represented by BlenX systems. The algorithm firstly generates the initial population. Each individual in the population is then simulated separately using the BWB stochastic simulator, and the outputs of the simulations are used to compute the fitness values of the individuals. Like in a real environment, individuals with the highest fitness values are more likely to survive, replicate and produce a progeny that resembles them, being not, however, completely equal to them. The different types of mutations we considered are based on real biological processes where mutations can happen at DNA and protein level. Variability is achieved by associating each of the considered mutations to a BlenX system modification.

The last investigation we presented is related with self-assembling structures. We first explored design patterns for modelling non-trivial structures like filaments and trees, showing also how to use immediate actions to control the shape of the forming structures. We then show how to model symmetric rings, considering also reversible mechanisms.

Summing-up, the contribution of this thesis can be seen as an effort to push ahead the application of informatics and particularly concurrency theory in the representation and the study of biological systems, in a way that could lead, one day, to the development of a formal foundation theory for systems biology.

Bibliography

[1] Cellucidate. Available at http://cellucidate.com/.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular biology of the cell*. Garland Science, 2002.

[3] J. Aranda, F. D. Valencia, and C. Versari. On the Expressive Power of Restriction and Priorities in CCS with Replication. In *FOSSACS*, volume 5504 of *Lecture Notes in Computer Science*, pages 242–256. Springer, 2009.

[4] M. M. Babu, S. A. Teichmann, and L. Aravind. Evolutionary dynamics of prokaryotic transcriptional regulators. *J. Mol. Biol.*, (358):614–633, 2006.

[5] J. C. M. Baeten, J. A. Bergstra, and J. W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *FUNINF*, IX(2):127–168, 1986.

[6] K. S. Booth and C. J. Colbourn. Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, Department of Computer Science, University of Waterloo, Ontario, Canada, 1977.

[7] N. Busi and R. Gorrieri. On the Computational Power of Brane Calculi. *T. Comp. Sys. Biology*, pages 16–43, 2006.

[8] N. Busi and G. Zavattaro. On the expressive power of movement and restriction in pure mobile ambients. *Theor. Comput. Sci.*, 322(3):477–515, 2004.

[9] J. Camilleri and G. Winskel. CCS with priority choice. *Inf. Comput.*, 116(1):26–37, 1995.

[10] L. Cardelli. Brane Calculi. In *Proc. Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–280. Springer, 2005.

[11] L. Cardelli and P. Gardner. Processes in Space. Technical Report 4, Imperial College London Department of Computing, 2009.

[12] L. Cardelli and A. D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.

[13] L. Cardelli and S. Pradalier. Where Membranes Meet Complexes. In *BioConcur 2005*, 2005.

[14] L. Cardelli and G. Zavattaro. On the Computational Power of Biochemistry. In *AB '08: Proceedings of the 3rd international conference on Algebraic Biology*, pages 65–80, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] D. Chiarugi, M. Curti, P. Degano, and R. Marangoni. VICE: A VIrtual CEll. In *CMSB*, volume 3082 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2005.

[16] F. Ciocchetta, A. Duguid, S. Gilmore, M. L. Guerriero, and J. Hillston. The Bio-PEPA Tool Suite. In *QEST*, 2009.

[17] F. Ciocchetta, S. Gilmore, M. L. Guerriero, and J. Hillston. Integrated Simulation and Model-Checking for the Analysis of Biochemical Systems. *Electr. Notes Theor. Comput. Sci.*, 232:17–38, 2009.

[18] F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theor. Comput. Sci.*, 2009.

[19] R. Cleaveland and R. Hennessy. Priorities in process algebras. *Inf. Comput.*, 87(1-2):58–77, 1990.

[20] A. Coletta, R. Gori, and F. Levi. Approximating Probabilistic Behaviors of Biological Systems Using Abstract Interpretation. *Electr. Notes Theor. Comput. Sci.*, 229(1):165–182, 2009.

[21] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[22] P. L. Curien, V. Danos, J. Krivine, and M. Zhang. Computational self-assembly. *Theor. Comput. Sci.*, 404(1-2):61–75, 2008.

[23] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-Based Modelling of Cellular Signalling. In *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 17–41. Springer, 2007.

[24] V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable Simulation of Cellular Signaling Networks. In *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 139–157. Springer, 2007.

[25] V. Danos and J. Krivine. Formal Molecular Biology Done in CCS-R. *Electr. Notes Theor. Comput. Sci.*, 180(3):31–49, 2007.

[26] V. Danos and C. Laneve. Formal molecular biology. *Theor. Comput. Sci.*, 325(1):69–110, 2004.

[27] V. Danos and S. Pradalier. Projective Brane Calculus. In *CMSB*, volume 3082 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2004.

[28] M. P. C. David, J. Y. Bantang, and E. R. Mendoza. A Projective Brane Calculus with Activate, Bud and Mate as Primitive Actions. *T. Comp. Sys. Biology*, 11:164–186, 2009.

[29] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation. *Indagationes Mathematicae*, 34:381–392, 1972.

[30] P. Degano, D. Prandi, C. Priami, and P. Quaglia. Beta-binders for Biological Quantitative Experiments. *Electr. Notes Theor. Comput. Sci.*, 164(3):101–117, 2006.

[31] E. K. Drexler. *Nanosystems: molecular machinery, manufacturing, and computation*. Wiley, 1992.

[32] J. Engelfriet. A Multiset Semantics for the pi-calculus with Replication. *Theor. Comput. Sci.*, 153(1&2):65–94, 1996.

[33] J. Engelfriet and T.E Gelsema. Multisets and structural congruence of the pi-calculus with replication. *Theor. Comput. Sci.*, 211(1-2):311–337, 1999.

[34] J. Engelfriet and T.E Gelsema. A new natural structural congruence in the pi-calculus with replication. *Acta Inf.*, 40(6-7):385–430, 2004.

[35] J. Engelfriet and T.E. Gelsema. The decidability of structural congruence for replication restricted pi-calculus processes. Technical Report 04-07, Leiden Institute of Advanced Computer Science, Leiden University, The Nederlands, 2004.

[36] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

[37] J. Fisher and T. A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–1249, 2007.

[38] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, 1966.

[39] P. François and V. Hakim. Design of genetic networks with specified functions by evolution in silico. *Proc. Natl. Acad. Sci.*, 2(101):580–5, 2004.

[40] N. Friedman, M. Linial, I. Nachman, and D. Peer. Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7(3):601–620, 2000.

[41] M. A. Gibson and J. Bruck. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. *J. Phys. Chem.*, 104:1876–1889, 2000.

[42] D. T. Gillespie. A General Method For Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *J. Phys. Chem.*, 22:403–434, 1976.

[43] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81(25):2340–2361, 1977.

[44] J. Hasty, D. McMillen, and J. J. Collins. Engineered gene circuits. *Nature*, 420(6912):224–230, 2002.

[45] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 319(3):239–257, 2008.

[46] M. Heiner, D. Gilbert, and R. Donaldson. Petri Nets for Systems and Synthetic Biology. In *SFM*, pages 215–264, 2008.

[47] G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.* (3), 2:326–336, 1952.

[48] J. Hillston. *A compositional approach to performance modelling.* Cambridge University Press, 1996.

[49] W. S. Hlavacek, J. R. Faeder, M. L. Blinov, R. G. Posner, M. Hucka, and W. Fontana. Rules for modeling signal-transduction systems. *Sci STKE*, 2006(344), 2006.

[50] C. A. R. Hoare. Communicating sequential processes. *Comm. ACM*, 21(8):666–677, 1978.

[51] C. Y. Huang and J. E. Ferrell. Ultrasensitivity in the mitogen-activated protein kinase cascade. *Proc Natl Acad Sci USA*, 93(19):10078–10083, September 1996.

[52] T. Ideker, T. Galitski, and L. Hood. A NEW APPROACH TO DECODING LIFE: Systems Biology. *Annual Review of Genomics and Human Genetics*, 2(1):343–372, 2001.

[53] M. John, R. Ewald, and A. M. Uhrmacher. A Spatial Extension to the pi Calculus. *Electr. Notes Theor. Comput. Sci.*, 194(3):133–148, 2008.

[54] M. John, C. Lhoussaine, J. Niehren, and A. M. Uhrmacher. The attributed pi calculus. In *CMSB*, volume 5307 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2008.

[55] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of theoretical biology*, 22(3):437–467, 1969.

[56] M. W. Kirschner. The meaning of systems biology. *Cell*, 121(4):503–504, May 2005.

[57] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, March 2002.

[58] E. Klavins. Automatic synthesis of controllers for assembly and formation forming. In *International Conference on Robotics and Automation*, 2002.

[59] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: its structural complexity*. Birkhäuser, 1993.

[60] J. Krivine, R. Milner, and A. Troina. Stochastic Bigraphs. *Electr. Notes Theor. Comput. Sci.*, 218:73–96, 2008.

[61] C. Kuttler and J. Niehren. Gene regulation in the pi calculus: simulating cooperativity at the lambda switch. *Transactions on Computational Systems Biology VII*, 4230:24–55, 2006.

[62] M. Kwiatkowska and J. Heath. Biological pathways as communicating computer systems. *Journal of Cell Science*, 122(16):2793–2800, 2009.

[63] M. Kwiatkowski and I. Stark. The continuous pi-calculus: A process algebra for biochemical modelling. In *CMSB*, volume 5307 of *Lecture Notes in Computer Science*, pages 103–122. Springer, 2008.

[64] C. Laneve and F. Tarissan. A simple calculus for proteins and cells. *Theor. Comput. Sci.*, 404(1-2):127–141, 2008.

[65] R. Larcher and C. Priami. From BlenX to chemical reactions via SBML. Technical report, TR-18-2008 CoSBi, 2008.

[66] N. Le Novère and T. S. Shimizu. STOCHSIM: modelling of stochastic biomolecular processes. *Bioinformatics*, 17(6):575–576, 2001.

[67] P. Lecca, C. Priami, P. Quaglia, B Rossi, C. Laudanna, and G. Constantin. A Stochastic Process Algebra Approach to Simulation of Autoreactive Lymphocyte Recruitment. *Simulation*, 80(6):273–288, 2004.

[68] L. M Loew and J. C. Schaff. The Virtual Cell: a software environment for computational cell biology. *Trends in Biotechnology*, 19(10):401–406, 2001.

[69] S. Maffeis and I. Phillips. On the computational strength of pure ambient calculi. *Theor. Comput. Sci.*, 330(3):501–551, 2005.

[70] P. Mendes. GEPASI: a software package for modelling the dynamics, steady states and control of biochemical and other systems. *Comput Appl Biosci*, 9(5):563–571, 1993.

[71] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[72] R. Milner. Function as processes. *Mathematical Structures in Computer Science*, 2:119–141, 1992.

[73] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[74] R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, New York, NY, USA, 2009.

[75] M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

[76] V. A. Muganthan, A. Phillips, and M. G. Vigliotti. BAM: BioAmbient machine. In *ACSD*, pages 45–49, 2008.

[77] Busi N., Gabbrielli M., and Zavattaro G. On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science*, 19(6):1191–1222, 2009.

[78] R. Nagpal. Programmable self-assembly using biologically-inspired multiagent control. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 418–425, 2002.

[79] A. Neumann. Graphical Gaussian Shape Models and Their Application to Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(3):316–329, 2003.

[80] A. Neumann. Graphical Gaussian Shape Models and Their Application to Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(3):316–329, 2003.

[81] F. Nielson, H. R. Nielson, P. Priami, and D. Rosa. Control flow analysis for bioambients. *Electr. Notes Theor. Comput. Sci.*, 180(3):65–79, 2007.

[82] D. Noble. The rise of computational biology. *Nat Rev Mol Cell Biol.*, 3(6):459–463, 2002.

[83] T. Pfeiffer and S. Schuster. Game-theoretical approaches to studying the evolution of biochemical systems. *Trends Biochem. Sci.*, 1(30):20–5, 2005.

[84] T. Pfeiffer, O. Soyer, and S. Bonhoeffer. The evolution of connectivity in metabolic networks. *PLoS Biol.*, 7(3), 2005.

[85] A. Phillips and L. Cardelli. Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus. In *CMSB*, volume 4695 of *LNCS*, page 184. Springer, 2007.

[86] A. Phillips, L. Cardelli, and G. Castagna. A Graphical Representation for Biological Processes in the Stochastic pi-Calculus. 4230:123–152, 2006.

[87] I. Phillips. CCS with Priority Guards. *Lecture Notes in Computer Science*, 2154, 2001.

[88] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[89] D. Prandi. *A Formal Study of Biological Interactions*. PhD thesis, International Doctorate School in Information and Communication Technologies, 2006.

[90] D. Prandi, C. Priami, and P. Quaglia. Communicating by compatibility. *JLAP*, 75:167, 2008.

[91] C. Priami. Stochastic π-Calculus. *The Computer Journal*, 38:578–589, 1995.

[92] C. Priami. Algorithmic systems biology. *Commun. ACM*, 52(5):80–88, 2009.

[93] C. Priami, P. Ballarini, and P. Quaglia. BlenX4Bio - BlenX for Biologists. In *CMSB*, volume 5688 of *Lecture Notes in Computer Science*, pages 26–51. Springer, 2009.

[94] C. Priami and P. Quaglia. Beta Binders for Biological Interactions. In *CMSB*, pages 20–33, 2004.

[95] C. Priami and P. Quaglia. Operational patterns in beta-binders. *T. Comp. Sys. Biology*, 1:50–65, 2005.

[96] C. Priami, A. Regev, W. Silverman, and E. Shapiro. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, 2001.

[97] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. BioAmbients: an abstraction for biological compartments. *Theor. comput. Sci.*, 325(1):141–167, 2004.

[98] A. Regev and E. Shapiro. Cells as Computations. *Nature*, 419:343, 2002.

[99] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2003.

[100] R. Sharan, S. Suthram, R. M. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. M. Karp, and T. Ideker. Conserved patterns of protein interaction in multiple species. *Proc Natl Acad Sci*, 6(102):1974–79, 2005.

[101] J. C. Shepherdson and H. E. Sturgis. Computability of Recursive Functions. *J. ACM*, 10(2):217–255, 1963.

[102] O. Soyer and S. Bonhoeffer. Evolution of Complexity in Signaling Pathways. *PNAS*, (103):16337–16342, 2006.

[103] A. M. Stock, V. L. Robinson, and P. N Goudreau. Two-Component Signal Transduction. *Annu. Rev. Biochem*, (69):183–215, 2000.

[104] Gillespie D. T. Simulation methods in systems biology. In *SFM*, volume 5016 of *Lecture Notes in Computer Science*, pages 125–167. Springer, 2008.

[105] M. Tomita, K. Hashimoto, K. Takahashi, T. S. Shimizu, Y. Matsuzaki, F. Miyoshi, K. Saito, S. Tanida, K. Yugi, J. C. Venter, and C. Hutchison. E-CELL: software environment for whole-cell simulation. *Bioinformatics*, 15(1):72–84, January 1999.

[106] M. H. Van Regenmortel. Reductionism and complexity in molecular biology. Scientists now have the tools to unravel biological and overcome the limitations of reductionism. *EMBO reports*, 5(11):1016–1020, 2004.

[107] C. Versari. A Core Calculus for a Comparative Analysis of Bio-inspired Calculi. In *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2007.

[108] C. Versari, N. Busi, and R. Gorrieri. On the Expressive Power of Global and Local Priority in Process Calculi. In *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 241–255. Springer, 2007.

[109] C. Versari and R. Gorrieri. pi@: A pi-Based Process Calculus for the Implementation of Compartmentalised Bio-inspired Calculi. In *SFM*, volume 5016 of *Lecture Notes in Computer Science*, pages 449–506. Springer, 2008.

[110] H. V. Westerhoff and B. O. Palsson. The evolution of molecular biology into systems biology. *Nature Biotechnology*, 22:1249–1252, 2004.

[111] P. Yin, H. M. T. Choi, C. R. Calvert, and N. A. Pierce. Programming biomolecular self-assembly pathways. *Nature*, 451(318-322), 2008.