PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Optimizing the Execution of Biological Models

Roberto Larcher

Advisor:

Prof. Corrado Priami

Università degli Studi di Trento

March 2011

Abstract

Modelling biological systems allows us to understand how their components interact and give rise to complex behaviour. Initially, biology relied on *mathematical models* based on systems of differential equations whose solution describes the concentration of the molecules in time. Recently, spurred by the metaphor of "cells as computation" by Regev and Shapiro, the scientific community adapted concurrent languages to describe biological systems. This led to the creation of *computational models* which are executable and not simply solvable. Executable models offer some advantages over systems of differential equations, such as allowing the modeller to capture the causality relations among the events that constitute the dynamics of the model evolution. However, these new approaches introduce new issues; for example executing a model is more computationally intensive than solving a system of differential equations, especially if the model has to be executed several times because of statistical constraints. In this thesis we focus on reducing the execution time of biological models by applying static analysis techniques like control flow analysis and abstract interpretation.

Keywords
[systems biology, process calculi, executable models, control flow analysis, abstract interpretation]

Acknowledgement

First of all I would like to thank Michael Smith and Davide Prandi for their precious advices in writing this thesis. I'm also grateful to Alessandro Romanel, Sean Sedwards, Federico Vaggi, Lorenzo Dematté and Alida Palimsano who, in different ways, contributed in improving the quality of this work.

I would also like to thank my advisor Corrado Priami who gave me the possibility to work in the COSBI team. Thanks to my reviewers Flemming Nielson and Chiara Bodei for their constructive observations and critiques.

I also thank my office mates Michele Forlin, Alessandro Romanel, Lorenzo Dematté, Alida Palmisano, Judit Zámborszky, Nerta Gjata, Archana Bajpai and Sree Harish Muppirisetty for the nice time we had together. Thanks also to all the other colleagues and friends who brighten up even the hardest day I had during my Ph.D.

I thank my family who supported me in these years making possible the achievement of this result. Finally, thanks to Valentina who showed enormous patience and gave me the strength to overcome all the obstacles I found in these years.

"Science is like sex: sometimes something useful comes out,
but that is not why we are doing it"

[Richard Phillips Feynman]

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Context

In the last decades, the amount of data produced by *molecular biology* has increased exponentially. New technology and experimental techniques enabled projects like *The Human Genome Project* which were unthinkable a few years ago. These projects shifted the focus of the discipline from the generation of data to their collection and analysis calling for a new discipline, *bioinformatics*. Bioinformatics applies statistics and computer science to molecular biology and it has carved itself a dominant role in both the generation and the interpretation of biological data. It is producing results which are both fascinating and extremely concrete, however it is still based on a reductionist approach. Reductionism is based on the assumption that the single components contain enough information to explain the complexity of a whole system. Although effective, this assumption represents one of the main limits of the reductionist approach [111] because, even once we have a complete network of connections between the components, we do not necessarily understand the overall behaviour of a system. Biological systems are extremely complex and show emergent properties and behaviours that cannot be explained or predicted by only studying the structure of their individual components. As Kitano points out in [61] 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 shifting on understanding the system structure and dynamics. Although it subverts 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*. The scientific community is still elaborating a unique definition [53, 60, 61, 86, 114], but we can capture its main principles and goals in the worlds of Kirschner [60]:

> "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."

One of the challenges of systems biology is the coherent representation of the studied systems. This is an open issue and scientific community is still exploring different approaches. A viable proposal comes from the observation that biological systems share many similarities with *reactive systems* [46], which have been widely studied in computer science. These similarities have been highlighted for the first time in [101], where the cell is presented as a system where independent agents, through interactions, give rise to a global behaviour. That work proposes *process calculi*, originally intended for the description of reactive systems, for modelling biology. The adaptation of process calculi to the modelling of biological systems, leads to the definition of new languages (see e.g. [20, 23, 33, 94, 96, 100, 104]). These languages differ from more traditional mathematical models used in systems biology because they allow the definition of computational models which are *executable* and not simply solvable [43]. Executable, in this context, means that we can describe and predict the flow of control between species and reactions; for instance, the formalism not only captures the kinetics, but also the causality relation among the events that constitute the dynamics of the model evolution.

This process calculus based approach, makes quite difficult to automatically extract information from the defined models. In fact, process calculi are extremely expressive and often Turing complete [15, 72, 75, 106, 107]. This implies that undecidability makes it not possible to fully investigate all the properties of a model [103]. A possible solution to this issue consists in the application of static analysis techniques which, at the price of a loss in precision, makes some interesting properties decidable. Valuable examples of static analysis applied to bio-inspired languages are *control flow analysis* [8, 91], *data flow analysis* [91] and *abstract interpretation* [32]. All of those are adaptation of methodologies originally introduced in the computer science area of *compiler construction*. In particular, optimising compilers rely on automatic procedures to maximize or minimize some attributes of an executable program (e.g. execution time). Compiler optimization is generally implemented using a sequence of optimizing transformations, i.e. algorithms which take a program and transform it to produce an output program that uses less resources but produces the same output.

In this thesis we observe that, in the context of process calculi, static analysis techniques have been used prevalently to formally verify qualitative properties of the models. Indeed, the original purpose of these techniques, which was to speed up the execution of a program, has not been investigated. However optimization of execution has several advantages in the biological context. In fact, when interesting properties cannot be verified through formal methods, simulations remains the only viable solution. Due the stochastic nature of biological systems a single simulation cannot give reliable information, and, in order to have informative statistics, it is sometimes necessary to execute the same model from hundreds to thousands of times [3] and therefore an optimized execution enables to save time, which immediately reflects in saving resources. In this work we investigate how to apply and adapt static analysis techniques in order to speed up the simulation of biological models.

1.2 The Problem

A process calculus can be characterized in different ways. One of the most common approaches consists in the definition of a *syntax* and an *operational semantics* [92]. The former defines how to write a process, the latter what the process does, i.e. how it executes. The behaviour of a process is well described by a *graph* which goes under the name of *transition system*. The nodes

of this graph represent the configurations which a process can assume and the arcs the actions which trigger the configuration changes.

Bio-inspired process calculi are usually equipped with quantitative information that define the speed and the probability of executing an action. This quantitative information are usually the rates of a negative exponential distribution [44]. In this way, when multiple actions are active, the distribution associated with each action is used to randomly choose the one to execute. Some calculi extend this idea by introducing two kinds of actions: standard actions which are associated with rates (low priority) and *immediate actions* (high priority). If a standard action is active together with an immediate action, it has no chance to be be executed. If more immediate actions are active different mechanisms can be chosen in order to establish which of them will fire.

In the case of process calculi without immediate actions, the transition system associated with a process is a *Markov stochastic process* and it is effectively described by a *continuous time Markov chain* (CTMC) [4]. If the calculus offers the possibility to specify immediate actions, the transition system is a *semi-Markov stochastic process*. In particular it is a process of the kind generated by generalized stochastic Petri nets (GSPNs) [56], where the states are partitioned in *vanishing* (which perform immediate actions) and *tangible* (which do not). Particularly interesting is the possibility to associate these processes with an underlying CTMC enabling the use of well known mathematical results in order to extract interesting properties about the analysed model. Unfortunately, it is often the case that bio-inspired processes generate a transition system which cannot be computed because of its huge or even infinite size. The same holds for the underlying CTMC and in practice, researchers have to perform simulations in order to analyze the system.

The outcome of a simulation consists in a list of paired *simulation times* and configurations. Reading this list, we go through the configurations visited by the process together with the time instant in which the configuration has been entered. The list is computed through the execution of the following simulation algorithm:

1. The simulation time is set to zero and the current configuration of the process is set to its initial configuration;
2. The simulation time and the current configuration are stored in the result;
3. The operational semantics is used to extracts the set of active actions from the current configuration;
4. If the state is tangible:
   (a) The rate of the active actions is used to determine the action to be fired and the time it takes to execute;
   (b) Via operational semantics the selected action is applied to the current configuration;
   (c) The computed configuration becomes the current one and the execution time of the fired action is added to the simulation time;
   (d) The procedure restarts from point 2;
5. If the state is vanishing
   (a) According to the policy of the scheduler one of the active action is chosen;

(b) Via operational semantics the selected action is applied to the current configuration;

(c) The computed configuration becomes the current one (note that the simulation time remains unchanged);

(d) The procedure restarts from point 2;

If the simulation is not interrupted it proceeds until the process enters a deadlock configuration, i.e., at point 3, there are no active actions.

Observing that the sojourn time of vanishing configurations is null we can consider them as non informative intermediate steps to move from tangible to tangible configurations. Jumping among tangible configurations opens to the interpretation of the simulation as a walk through the states of a CTMC, in particular of the CTMC associated with the transition system generated by the process. This enables the possibility of modifying the simulation procedure in order to make it faster, while, at the same time, generating the same results in terms of visited tangible configurations. The idea is to define a simulation algorithm which visits only tangible configurations, without traversing the vanishing ones. To obtain a method which makes the simulation visit exclusively tangible configurations is extremely challenging. In practice, we explore a hybrid solution, i.e. in some cases the simulation jumps from tangible to tangible configurations, in others, it traverses the vanishing configurations. In this case, the simulations "travels" in a transition system which is a reduced version of that generated by the process, but with the same underlying CTMC.

1.3 Contribution

BlenX [104] is a recent language intended for modelling biological systems. This formalism builds on standard approaches a notion of *structural interaction* between species. More specifically, a species has both an internal state and some interfaces: the internal state evolves over time mimicking the structural modification of a species and the interfaces interact with other species to form *complexes*. This approach allows the modelling of complex biological structures such as chains or grids of proteins.

BlenX represents species with objects named *boxes*. A box is equipped with a set of interfaces and a number of internal processes running in parallel. The processes are written in a language similar to the π-calculus [76] and can communicate over channels. Boxes also have the capability to communicate with each other by sending messages through their interfaces. The same interfaces are used to create bindings between boxes. The interaction capability of the interfaces are defined by the *sort* they are associated with. Sorts define binding, unbinding and communication capabilities of the interfaces. In some cases communication through two interfaces can happen only if they are bound together, in other cases instead it can happen even if they are free (i.e. not bound), depending on the modeler's choice. BlenX allows the modeler to specify immediate actions, introducing the distinction between vanishing and tangible configurations. The importance of immediate actions is shown, for instance, in the case of models which describe self-assembly mechanism [66]. In this context, boxes can be programmed to build long filaments and tree structures: immediate actions allow computations inside boxes which are invisible in terms of simulation outcome but are necessary to implement spatial and physical constraints which cannot be expressed with a single primitive of the language.

In this thesis we develop a method based on different static analysis techniques which recognize when a system is going through the execution of a sequence of immediate actions and

directly jumps it to the next tangible system. This approach allows us to define an optimized simulation algorithm which minimizes the number of visited vanishing systems.

The version of BlenX we consider is the one presented in [104], with a slightly different definition of *event*. Events are a feature of the language which allows the modeller to substitute some free boxes (i.e. not bound to anything) with others. Here, we limit the power of this construct by restricting to substitutions of a single box with a set of boxes. With this change, we can define a structural operational semantics instead of the reduction semantics of [104]. The introduction of this new semantics, in the style of PEPA [50], makes the presentation of our results more intuitive and easier to understand.

The key of our work is a radical change of viewpoint. Instead of looking at the whole BlenX system, we focus our efforts in the analysis of the boxes populating it. The intuition is that the changes in the configuration of BlenX systems are driven by actions which involve one or more boxes. This implies that, if we compute the configurations reachable by a box and the actions which cause its configuration changes, we can collect information about the behaviour of the whole system. For example, if we know that a box can independently perform an immediate action, we can deduce that the same holds in any system containing such a box. For this reason, we set up a method which computes an over-approximation of the configurations a box can assume with respect to a BlenX system. This over-approximation process is supported by *control flaw analysis* which has been successfully applied to many process calculi (e.g. [8, 10, 36, 82]).

In order to be able to compute all the configurations which boxes can assume, we have to ensure that the total space of configurations is finite. This is not always the case in BlenX. However, we observe that, in nature, molecules assume a finite set of interesting configurations. Therefore, if our boxes are intended to mimic such molecules it makes sense to think about the introduction of some syntax constraints which prevent them from generating infinite configurations. In this work we present such constraints and we prove that they actually limit the boxes in the way we desire.

A key point in understanding which actions boxes can perform is to understand how they can bind together. In fact, the activation of some actions depends on interactions over bound interfaces. To get this information we exploit the abstract interpretation method proposed in [32]. In that work, starting from a disconnected graph and a set of rewriting rules, a method is defined to over-approximate the obtainable connected components. The formalism used is the κ-calculus [33], a bio-inspired language which represents molecules as connected graphs and biological reactions as graph rewriting rules. Therefore, to exploit the results of [32], we propose an encoding of the BlenX language in κ-calculus. Exploiting the encoding we gain information about the possible binding interactions occurring among boxes in a BlenX system.

Using the methodology described, given a system, we identify groups of boxes which can potentially appear during its evolution and which perform atomic sequences of immediate actions to finally reach a tangible configuration (i.e., a configuration where no immediate actions are active). We define sequences of actions as atomic if whatever the boxes involved in the system are, they cannot be interrupted. The interesting aspect of the identified groups of boxes is that when one of them appears in a given system we know that, through an atomic sequence of immediate actions, they always undergo through the same transformation. This leads to the definition of an optimized simulation algorithm that, once identified a group of boxes obeying the desired properties, recognizes that the system is entering a sequence of vanishing configurations, and, instead of computing them, it immediately jumps to the next tangible system. This can be done by replacing the identified group of boxes with those resulting from the execution of the

immediate actions (which can be statically precomputed before the simulation starts).

1.4 Structure of the Thesis

Chapter 2 introduces biological systems and process calculi; moreover it gives the mathematical preliminaries necessary to understand the static analysis techniques we apply;

Chapter 3 presents BlenX and shows some examples which clarifies the role and the importance of immediate actions;

Chapter 4 presents a running example (actin polymerization) which is used to make more intuitive the presentation of the techniques defined in the thesis;

Chapter 5 starts with the definition of the method to generate all the configurations which a BlenX box can assume. After that, it introduces the syntax constraints which ensure the computability of the configurations reachable by a box;

Chapter 6 defines a control flow analysis for BlenX and explains how refine the technique presented in Chapter 5;

Chapter 7 presents the encoding of BlenX into κ-calculus;

Chapter 8 shows how to exploit the presented techniques to speed up the execution of BlenX models;

Chapter 9 concludes the thesis with a summary and the proposal of future works.

1.5 Related Publications

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

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

Chapter 2

Background

2.1 Biological Systems

The cell is the fundamental unit of life. Even the simplest unicellular organism, in fact, can grow, reproduce, process information, respond to stimuli, and carry out chemical reactions. We divide cells in two big families, prokaryotes and eukaryotes. Prokariotic cells consists of a unique compartment surrounded by the plasma membrane where all the molecules necessary for carrying on the life activity float together. Eukaryotic cells have a more complex structure. They have a defined membrane-bound nucleus and extensive internal membranes that enclose other compartments, the organelles. Even though two kind of cells have several differences, they can live and reproduce through mechanisms that are based on the same molecules and concepts. The molecules that are the base of life are *Proteins*, *DNA*, *RNA*, and *metabolites* [2].

Proteins: Proteins are the machines that carry out the activities inside cells. They play fundamental roles in structuring the cell, in reaction catalysis and regulation, transmission of signal between different compartments, movement of molecules from one location to another and many other tasks. Proteins are made of a chain of amino acids, whose linear sequence (also called primary structure) is stored in the DNA. There are twenty-one different amino acids whose characteristics drive the folding of the protein and determine its chemical properties. Depending on the amino acids composition, a protein can have regions with positive or negative charge, regions which are hydrophobic or hydrophilic and regions which are basic or acidic. 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 *domains* that allow the protein 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, we obtain the so called quaternary structure. When a protein interacts with other molecules, even if its amino acid composition remains the same, its structure changes. This can result in the modification of its properties and in a different behaviour. The different three-dimensional shapes of a protein are called *conformational states*.

DNA and RNA: DNA is an abbreviation for *Deoxyribonucleic Acid*. The three-dimensional structure of DNA consists of two long helical strands that are coiled around a common

axis, forming a double helix. DNA strands are made of small monomers called nucleotides. There are four different nucleotides: A, T, C and G. Each DNA double helix has a simple construction: wherever there is an A in one strand there is a T in the other, and each C is matched with a G. This complementary matching of the two strands is so strong that if complementary strands are separated, they will spontaneously zip back together. The information in DNA is stored as a code by the sequence of the four different nucleotides. The portion of DNA which carries information is divided in units called genes. A gene can be divided in two regions: one region contains the linear sequence of amino acids necessary to form a protein and the other region contains regulatory elements which determine expression levels and timing.

Even if the information for building proteins is stored in the DNA, they are not produced directly starting from the DNA. An intermediate step is necessary and it is called *transcription*. Such a process, starting from the DNA, produces a single strain of nucleotides which is called RNA (*Ribonucleic Acid*). RNA is produced using the DNA as a template, and the machine that takes care of creating this molecule is a large enzyme called RNA polymerase. Once RNA is created, the last step for the creation of the protein takes place. This process is called *transcription* and is carried out by an enormously complex molecule called ribosome. The ribosome reads the sequence of nucleotides and translate it into a sequence of amino acids which composes the coded protein.

Metabolite: a metabolite is an intermediate product of 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 perform very specific tasks and their structure and chemical properties are relatively simple if compared, for example, with proteins. We can think of them as having a single identity and state. Indeed, if a metabolite reacts it becomes a different metabolite with a different identity and function.

The combination of mechanisms that starting from DNA results in the production of a protein goes under the name of *The Central Dogma of Molecular Biology*, and was theorized for the first time by Crick in 1958 [27].

2.1.1 Biochemical Reactions

The molecules inside a cell interact in thousands of different ways. We can describe these complex networks of interactions as a set of chemical reactions. An example of reaction is:

$$2A + B \rightarrow C$$

Molecules A and B are *reactants*, while C is the *product*. When this reaction happens two molecules of type A and one of type B are consumed in order to generate one molecule of type C. The number of molecules of each kind involved in the reaction (in this case 2 for A, 1 for B and 1 for C) are called *stoichiometries*. A reaction that can happen in both directions is called reversible. They are frequent in biology and are written adding a reverse arrow for the backward reaction:

$$2A + B \leftrightharpoons C$$

Chemical kinetics is concerned with the time-evolution of a reaction system specified by a set of chemical reactions. In particular, it is concerned with the behaviour of the system away form 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 reaction occurs, together with the initial concentration of the reacting molecules, are also necessary. The rate of a reaction is a measure of how the concentration of the involved substances changes over time. To better understand the relationship between rate and concentration, let us consider a single state reaction in which one molecule A reacts with one molecule B, giving one molecule C.

$$A + B \to C$$

According to collision theory, C is produced with a rate that is proportional to the hit frequency of A and B. If we have a bounded volume with a unique molecule A and several molecules B, the hit frequency depends on how many molecules B we have. The higher their number, the higher the probability of a collision with the molecule A that leads to the formation of a C molecule. For the same reason, if we add a molecule A, the rate of formation of C 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 state that *the rate of a chemical reaction is directly proportional to the product of the effective concentration of each participating molecule.* Hence in our case we 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. The basal rate is a constant that depends on several factors, e.g. the temperature of the solution where the molecules move and the probability that the hit between the reactants actually cause the formation of the product. In general, the reaction speed is proportional to the concentration of the reactants involved raised to the power of their stoichiometry. For example, given the homogeneous reaction[1]

$$nA + mB \to P$$

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

2.1.2 Combinatorial Complexity of Biological Systems

Biological systems are driven by chemical reactions. Products and reactants involved in such reactions are genes, RNA strands, metabolites and proteins.

These molecules (with the exception of metabolites) have a complex structure which can change in time and influence the way they bind with the others molecules. Let us consider the case of proteins and how they can influence their structure by interacting among themselves. Protein-protein interactions can lead to various posttranslational protein modifications, such as phosphorylation, ubquitinaton, acetylation, or methylation ([89, 109]). The effect of such modifications may be a change in a protein's enzymatic[2] or binding activity.

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

[2] To have an enzymatic activity means to have the capability to boost a reaction

There are proteins which have several sites that can be modified and this gives them the capability of assuming a large number of conformational states. If we consider that these molecules can also bind together (forming a *complex*) and undergo conformational change, we begin to appreciate that in a biological systems the number of different molecules that can potentially appear is huge. This is a well known problem and it is known as *combinatorial complexity* (as in [51]).

A good example of a protein which can assume several configurations is the EGFR receptor. This molecule, according to [55], during its activity can be modified in nine different sites. As a simplification, let us assume that only one of the nine sites of EGFR can be phosphorylated at a time. Under this assumption there are still 10 different phosphoforms and if we consider that two EGFR receptors can bind together we realize that they can generate 55 distinct configurations. These numbers grow to 512 and 131,328 if more than one site of EGFR can be phosphorylated at a time. The number of phosphorylation states relevant for signaling in particular contexts is unknown, and it seems unlikely that all states are functionally relevant or even realized in a cell. Indeed, some phosphorylation states may be prohibited [57]. However, even if several configurations are not admitted, if we want to give an effective description of such a system through the list of reactions that can take place the task is very hard. There are several chemical species to be considered and even more ways in which these species react.

The elevated number of configurations assumable by a protein is not the only cause of combinatorial explosion. Another case in which such an event takes place is the presence of proteins which can bind together forming long chains made of the same bricks repeated hundreds or even thousands of times. This binding process is called *polymerization* and the resulting molecules are called *polymers*. An example of *polymerization* is carried on by the actin protein. Actin is a small globular protein present in almost all known eukaryotes. It is one of the most conserved proteins among species because it is involved in fundamental cell processes, e.g. muscle contraction, cell motility, cell division and cytokinesis, vesicle and organelle movement, cell signalling, and the establishment and maintenance of cell junctions and cell shape. All these tasks are performed thanks to the ability of actin to polymerize in long filaments. These filaments are polarized, thus it is possible to distinguish the two ends, called *pointed* and *barbed*. Actin can interact with a multitude of molecules; among these we have the ARP2/3 complex, a seven subunit complex. This complex has the capability to bind existing actin filaments and become a nucleation site for a new filament. This leads to the formation of branched filament networks that are important for process like cell locomotion, phagocytosis, and intracellular motility of lipid vesicles. For more information regarding actin refer to [2]. This polymerization process leads to the creation of a huge number (theoretically infinite if we do not limit the dimension of a filament) of molecules which differ from each other by the number of actin monomers and by the location of Arp2/3 branches. Therefore, we have an example of the combinatorial explosion, making it impossible to describe the system by listing all elementary reactions between the individual species.

2.2 Modelling Biological Systems

2.2.1 Ordinary differential equations

Ordinary differential equations (ODEs) have been widely used for modelling several kind of systems such as population growth, epidemiology, electric circuits and fluid dynamics. Thanks

to their success in other fields, biologists started modelling biology with this mathematical tool. This approach associates each chemical species of the system with a different variable. Each variable represents a concentration and is associated with an equation which describes how it changes during the time in relation with the concentration of the other species in the system. We implicitly used an ordinary differential equation when we introduced biochemical interactions. The chemical reaction considered was:

$$A + B \rightarrow C$$

For describing the change rate of the concentration concentration of C, we use the following equation:

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

where k is the basal rate of the reaction. The equations describing the concentration of A and B are the following:

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

Given an initial concentration of the species involved in the system it is possible to define a system of differential equations which shows how the concentration of the species changes over the time. To find an approximate solution of a system of ordinary differential equation through numerical analysis is an well-studied problem. Furthermore, in order to decrease the number of the equations and thus the complexity of the system, biologists introduced several approximations which simplify a set of reactions with only one equation. Among the most famous there are the Michaelis-Menten equation [74] (for describing enzyme activity) and the Hill equation [49] (for describing cooperative binding).

Dealing with ODEs can be quite difficult for people without a mathematical background, and this leads the scientific community to produce several tools in order to simplify this procedure (e.g. [52, 108]). Those tools take as input a list of reactions and an initial state of the system and produce and solve the corresponding ODE system. Several kinds of analysis are possible in order to obtain interesting information from the model.

2.2.2 Stochastic Simulation

Given a set of chemical reactions describing a biological system, a simulation is one of the most immediate and natural analyses that can be performed on it. 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 number of molecules of each involved species. 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.

This approach is particularly interesting in systems where the quantity of some molecules can be very low. In this case, the thermodynamic limit cannot be always assumed and microscopic random effects arise making the system naturally stochastic. Under this condition ODE systems, which are deterministic (i.e. given an initial configuration, their dynamics is univocally determined), are an unsuitable approach. 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 realizations can be generated through Monte Carlo sampling methods. The most famous of these methods for chemical reactions is the SSA algorithm of Gillespie [45, 115].

For stochastic simulations, there are similar tools which take as input a list of chemical reactions and an initial configuration of the system and automatically carry on stochastic simulations [69, 99].

2.2.3 Handling the Combinatorial Complexity

Both of the considered approaches require the modeller to list all the species and the reactions involved in the system. This means that they are not suitable to handle the combinatorial complexity of some biological systems. To handle this problem it is possible to introduce assumptions in order to limit the number of species and chemical reactions to be considered.

If we consider again the EGFR receptor, an example of such approach is the model of Kholodenko *et. al* [58]. This model is based on mechanistic assumptions that result in focusing the attention on a fraction of the protein complexes that could potentially arise from the protein-protein interactions considered in the model. For example, one assumption is that ligand-induced dimers of EFGR are unable to dissociate when receptors are phosphorylated, this assumption is reasonable, but lacks clear empirical support. The risk is that assumptions, which are difficult to justify, are introduced for the sake of avoiding combinatorial explosion. For example, in the Kholodenko model, if we remove some of the simplifications, the size of the model starts growing and combinatorial explosion rears its ugly head again [7], making manual model specifications impractical.

A more effective solution to the combinatorial explosion is based on the following idea: instead of listing the chemical reactions, we can list the protein-protein interactions and use them as rules that serve for generating the chemical reactions and species. The idea is to exploit the modularity of the protein interaction domains. In one approach to rule-based modelling ([38],[5]), which is implemented in BioNetGen software package, rules are used for generating the chemical reactions and species. A rule comprises patterns for recognizing reactants, a mapping of reactants to products, and a rate law. Given a set of initial chemical species, represented with text strings or graphs, each rule is used to identify, through pattern matching, the species that have features required to undergo the transformation from reactants to products specified in the rule. Each transformation is assigned a rate law, the one associated with the corresponding rule. With this approach, more than one species can be involved in a reactions because of the same rule. The number of reactions generated by a rule depends on the initial state of the system and on the other rules. The application of the rules, in order to get the list of the reactions and the species, can be performed with two approaches: the *generate-first* and the *on-the-fly* approach. With the generate-first approach all the species and the reactions are obtained through an iterative application of the rules until a specified condition is satisfied or no new reactions are generated [38]. With the on-the-fly approach, reactions are generated as new species become populated, which may be advantageous when the network is large or unbounded, as it is the case when rule application is nonterminating in the absence of an arbitrary halting condition [38, 71].

2.2.4 Rule-based Modelling with Process Calculi

There are many tools which use rules-based modelling in order to model biological systems. Among them we find StochSim [67], Moleculizer [71], BioNetGen [6, 38, 39], and Biocham [40]. In this thesis we are interested in approaches which differ from these because they belong to the family of *process calculi*.

Starting from the *Calculus of Communicating Systems* [75] (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 [92]. These rules allow to infer the possible evolutions 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 is that complementary actions are those that parallel processes can perform together to synchronise their (otherwise) independent behaviour.

A process is a computational unit which evolves by performing actions $(a, b, \dots)$ and co-actions $(\overline{a}, \overline{b}, \dots)$. 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 transition: $\overline{a}.P \mid (a.Q + a.R) \to P \mid Q$ and $\overline{a}.P \mid (a.Q + a.R) \to P \mid R$.

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, it 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 from the external world.

Infinite behaviours are usually obtained in process calculi by using operators like replication, denoted by !P , which allow 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 [14].

Above we recalled only the fundamental operators which are common to various process calculi. 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 us 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 $P = \overline{a}.P1 \mid a.P2 \mid \overline{a}.P3$ is drawn as:

$$P \rightarrow P1 \mid P2 \mid \overline{a}.P3$$

and

$$P \rightarrow \overline{a}.P1 \mid P2 \mid P3$$

The depicted transitions highlight that both a.P1 and a.P3 can communicate with a.P2. The evolution of the system depends upon the temporal order of the interaction. Since no assumption can be made about 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. 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 [27, 95]. 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.

In this introductory description of process calculi we can appreciate several similarities with the rule-based approach. The first intuition of this parallelism has been argued by Regev *et al.* in [102]. In their abstraction (see Table 2.1) they propose proteins are mobile processes (meaning agents that exchange messages that can affect their behaviour); protein sites are communication channels (ports through which messages are passed from senders to receivers); and protein-protein interactions are communications. The channels of a system and the messages that can be sent and received along these channels are essentially rules for protein-protein interactions. The proteins in a complex are linked by a backbone (that is, by co-location in a communication compartment), and the topology of a complex is indicated by pair-wise communications. In this framework, signal transduction can be viewed as asynchronous concurrent computation and studied by using pertinent methods from computer science.

Stochastic extensions of process calculi, moreover, allow the description of the same kind of stochastic processes used to directly simulate systems of chemical reactions. This analogy allows us to use stochastic simulation by means of SSA Gillespie's algorithm also in the context of process calculi, making process calculi suitable for the representation of biological systems.

After the work of Regev and Shapiro [101] a number of process calculi have been adapted or newly developed for applications in systems biology.

Biochemical π-calculus [98]: 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

Biology	Process calculi
Molecule	Process
Interaction Capability	Channel
Interaction	Communication
Binding	Colocation in a communication compartment
Dynamics	State Change

Table 2.1 – Process calculi abstraction for systems biology.

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 [98] and SPiM [90]; these simulators implement the DM of Gillespie. Moreover, interesting applications of biochemical π-calculus on real biological scenarios can be found in [17, 20, 65, 68].

Performance Evaluation Process Algebra (PEPA) [50]: 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 [22] the authors show how the combined use of PEPA and the probabilistic model checker PRISM [48] can be used to describe, simulate and analyse biochemical signalling pathways. Recently, an extension called Bio-PEPA [23] 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 [21].

BioAmbients [100]: is a variant of Mobile Ambients [18] for systems biology and it focuses 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. Ambients 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 nested compartments. 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.

Brane Calculi [16]: 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 [34], 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 between the calculus and biological membrane reactions, a new extension has been proposed in [35]. 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.

π@-calculus [113]: is an extension of the π-calculus, obtained with the addition of polyadic synchronisation and priorities. The expressiveness of the calculus is shown in [112] 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 attributed π-calculus [54]: 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.

Beta-binders [96]: 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 [94], 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 [93]. 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 open as possible, with the goal of accommodating possible distinct instances of the same macro-behaviour. A stochastic extension of Beta-binders for quantitative experiments is presented in [36].

Now we focus our attention on κ-calculus and BlenX. We will make extensive use of these calculi in this work and thus we will analyze them more extensively. κ-calculus is presented in the next paragraph, BlenX is presented in Chapter 3 where we also propose a new operational semantics for the language, which makes the work of the thesis easier to read and understand.

2.2.5 κ-calculus

The κ-calculus [33] introduces a formal way to express a certain type of graph rewriting system. In this language molecules are represented as graphs and molecules modifications and interactions through rewriting rules over these graphs.

In this context we introduce a simplified version of the calculus which is a mix of the versions presented in [32] and [29]. We do not provide the stochastic extension ([31]) and we present the calculus with a process algebra notation (used in [32] and [29]) and not with the graphical notation of [29].

In κ-calculus we have a finite set of agent types `Agent`, representing different kinds of proteins; a finite set of interfaces `Interface`, corresponding to protein domains and modifiable residues; a finite set of values `Value`, representing the modified states. The syntax of agents and expressions is given in Table 2.2; a signature Σ from `Agent` to a finite subset of `Interface` assigning a set of interfaces to each agent type.

An *interface* is associated with an internal and a binding state; we write x_ι^λ for the interface named x with internal state ι and binding state λ. If the binding state is ϵ_λ the interface is *free*,

$$
\begin{array}{lcll}
E & ::= & \varepsilon \mid a, E & \text{(expression)} \\
a & ::= & \emptyset \mid A\,(\sigma) & \text{(agent)} \\
\sigma & ::= & \varepsilon \mid s, \sigma & \text{(interface sequence)}
\end{array}
\qquad
\begin{array}{lcll}
s & ::= & x_{\iota}^{\lambda} & \text{(interface or site)} \\
\lambda & ::= & \epsilon_{\lambda} \mid i \mid - & \text{(binding state)} \\
\iota & ::= & \epsilon_{\iota} \mid v & \text{(internal state)}
\end{array}
$$

Table 2.2 – Syntax of κ-calculus where $A \in \texttt{Agent}$, $i \in \mathbb{N}$, $x \in \texttt{Interface}$ and $v \in \texttt{Value}$.

$$
\begin{array}{rcl}
E, A\,(\sigma, s, s', \sigma')\,, E' & \equiv_{\kappa} & E, A\,(\sigma, s', s, \sigma')\,, E' \\
E, a, a', E' & \equiv_{\kappa} & E, a', a, E' \\
E & \equiv_{\kappa} & E, \emptyset \\
i, j \in \mathbb{N} \wedge i \text{ does not occur in } E & \Rightarrow & E[i/j] \equiv_{\kappa} E \\
i \in \mathbb{N} \wedge i \text{ occurs once in } E & \Rightarrow & E[\epsilon_{\lambda}/i] \equiv_{\kappa} E
\end{array}
$$

Table 2.3 – Structural equivalence for expressions.

otherwise it is *bound*. The binding state is $i \in \mathbb{N}$ when we know the binding partner (which is also bearing the same i), if we only know that an interface is bound and we do not have further information about its partner we use the *wildcard* $-$. On the other hand, if the internal state is ϵ_{ι}, this means the internal state is left unspecified. In the concrete notation ϵ_{λ} and ϵ_{ι} are omitted. Finally an *agent* is either a *proper agent* or a *ghost agent* which we refer as $\emptyset$. A proper agent is given by a name in `Agent` and an interface sequence. In an expression two interfaces which share the same binding state $i \in \mathbb{N}$ are said to be bound.

The language is provided with a notion of structural equivalence which is defined as the smallest equivalence relation which satisfies the rules of Table 2.3.

This equivalence says that: the order of interfaces in interface sequences and of agents in expressions does not matter; ghost agents can be erased, binding labels can be injectively renamed and *dangling bonds* (i.e. binding labels that occur once) removed. Equivalence class of $\equiv_{\kappa}$ are called *solutions* and one write $[\![E]\!]$ for the class of expression E. We refer to the set of all solutions as `Sol`.

We now define some classes of expressions.

Definition 2.2.1 (Pattern) *A pattern is an expression E such that:*

1. *an interface x occurs at most once in any agent $A\,(\sigma)$ in E.*
2. *if x occurs in $A\,(\sigma)$ then $x \in \Sigma(A)$.*
3. *each binding label i in E occurs exactly twice. (there are no dangling bonds).*

Definition 2.2.2 (Proper pattern) *A pattern E is proper if it is made of proper agents.*

Definition 2.2.3 (Mixture) *A mixture E is a non-empty proper pattern that is fully specified, i.e. each agent occurrence A in E documents its full interface $\Sigma(A)$, and interfaces can only be free or bear a binding label $i \in \mathbb{N}$.*

$$x_\iota^\lambda \;\vDash_\kappa\; x_\iota^\lambda \qquad x_\iota^i \;\vDash_\kappa\; x_\iota^- \qquad \sigma \;\vDash_\kappa\; \varepsilon$$

$$\frac{s \vDash_\kappa s_l \qquad \sigma \vDash_\kappa \sigma_l}{s, \sigma \vDash_\kappa s_l, \sigma_l} \qquad \frac{\sigma \vDash_\kappa \sigma_l}{A\,(\sigma) \vDash_\kappa A\,(\sigma_l)}$$

$$\emptyset \;\vDash_\kappa\; \emptyset \qquad E \;\vDash_\kappa\; \varepsilon$$

$$\frac{a \vDash_\kappa a_l \qquad E \vDash_\kappa E_l}{a, E \vDash_\kappa a_l, E_l}$$

$$\begin{aligned}
\lambda_r = i, \epsilon_\lambda \;&\Rightarrow\; x_\iota^\lambda[x_{\iota_r}^{\lambda_r}] = x_{\iota_r}^{\lambda_r} \\
x_\iota^\lambda[x_{\iota_r}^-] \;&=\; x_{\iota_r}^\lambda \\
\sigma[\varepsilon] \;&=\; \sigma \\
s, \sigma[s_r, \sigma_r] \;&=\; s[s_r], \sigma[\sigma_r] \\
A\,(\sigma)\,[A\,(\sigma_r)] \;&=\; A\,(\sigma[\sigma_r]) \\
E[\varepsilon] \;&=\; E \\
\emptyset[a_r] \;&=\; a_r \\
a[\emptyset] \;&=\; \emptyset \\
(a, E)[a_r, E_r] \;&=\; a[a_r], E[E_r]
\end{aligned}$$

Table 2.4 – Definitions of $\vDash_\kappa$ (left) and substitution of an expression in another one (right).

Definition 2.2.4 (Disconnected mixture) *A mixture E is disconnected if $E = E', E''$ for some mixture E', E''*

Definition 2.2.5 (Complex) *A solution $[\![E]\!]$ is a complex if E is a mixture and it is not disconnected. We write* `Com` *for the set of all the complexes.*

Now we introduce the *rules*. A rule is an ordered pair of patterns E_l, E_r, sometimes written $E_l \to E_r$, with additional constraints (explained below). The left hand side (lhs) E_l of a rule describes the agents taking part in it and various conditions on their bindings states for the rule to apply. The right hand side (rhs) E_r describes what the rule does. Ghost agents are used for agent creation in the lhs and agent removal in the rhs. Rules do not require the usage of fully specified patterns, they only have to respect the following constraints. In a rule E_l, E_r, the pattern E_r must be obtainable form the pattern E_l in the following stepwise fashion (the order matters):

- some wildcard and pairs of binding labels are removed (edge deletion);
- some ghost agents in E_l are replaced by agents with full free interfaces (as specified by Σ)(agent creation);
- some agents with only free interfaces are replaced with ghost agents (agent deletion);
- some free interfaces are bound using fresh labels in $\mathbb{N}$ (edge creation).

It follows from the above constraints that both sides $E_l = a_1, \ldots, a_n$, and $E_r = a'_1, \ldots, a'_n$ must have the same number $n(r) \geq 0$ of agents (which is the reason of the existence of ghost agents).

The application of a rule $r = E_l, E_r$ to a mixture E starts with the alignment of E with E_l. This step relies on structural congruence to bring the participating agents to the front of E with their interfaces ordered as in E_l, renaming binding labels and introducing ghost agents as necessary (for agents created by r). This yields an equivalent expression $E' \vDash_\kappa E_l$ (where $\vDash_\kappa$ is defined in Table 2.4).

Once a match is realized, we replace in E' the lhs E_l by the rhs E_r. We write $E'[E_r]$ for the result of this substitution and this operation is formally defined in Table 2.4. This may produce dangling bonds (if r unbinds a wildcard bond, there is a "side effect" as the other side of the bond needs to be removed as well) and/or ghost agents (if r delete agents), which one can clean up using $\equiv_\kappa$ afterwards.

Now we can define the transition system generated by a set of rules $\mathcal{R}$. Suppose E_0, E_1 are mixtures, $r = E_l \to E_r$ is a rule in $\mathcal{R}$, $E_0 \equiv_\kappa E_0'$, $E_0' \vDash_\kappa E_l$, and $E_0'[E_r] \equiv_\kappa E_1$ then we write $E_0 \xrightarrow[\kappa]{r} E_1$. We can naturally lift the transition system to solutions saying that $[\![E_0]\!] \xrightarrow[\kappa]{r} [\![E_1]\!]$ iff $E_0 \xrightarrow[\kappa]{r} E_1$.

Here we propose a simple example in order to see in practice the application of a rule to a mixture. The example involves the following rule and mixture:

$$\begin{array}{rcl} r & := & B\left(c^-\right) \to B\left(c\right) \\ E & := & A\left(b^1\right), B\left(a^1, c^2\right), C\left(b^2\right) \end{array}$$

In order to apply the rule r to the expression E, we rewrite E to the equivalent form $E' = B\left(c^2, a^1\right), A\left(b^1\right), C\left(b^2\right)$. Then given that $E' \vDash_\kappa B\left(c^-\right)$ is true we can proceed with the substitution of the rhs $[B\left(c\right)]$ in the mixture E':

$$\begin{array}{rcl} E'[B\left(c\right)] & = & B\left(c^2, a^1\right)[B\left(c\right)], A\left(b^1\right), C\left(b^2\right) \\ & = & B\left(c^2[c], a^1\right), A\left(b^1\right), C\left(b^2\right) \\ & = & B\left(c, a^1\right), A\left(b^1\right), C\left(b^2\right) \\ & \equiv_\kappa & B\left(c, a^1\right), A\left(b^1\right), C\left(b\right) \end{array}$$

In the last step we get rid of the dangling bond generated by the application of the rule.

2.3 Static Analysis Techniques

A lot of languages we are used to deal with are Turing complete (for a result regarding the computational power of BlenX see [106]). This property, which gives them the possibility to express any computable function, makes in practice impossible to verify interesting property over programs [103].

The impossibility to give an answer to these properties, which are defined *undecidable*, does not mean that we have to give up from extracting interesting information from programs. What we can do is to introduce some approximations in our analysis, i.e., instead of accepting as answer to a question only "yes" or "no", we also accept "may be".

In order to formalize the correctness of approximations and to express properties about them, many mathematical frameworks have been set up. Among them we have Data Flow Analysis [59] which tracks how data moves through a collection of atomic computations, and Control Flow Analysis [110] which tracks how the point of control traverses a program.

Traditionally Data Flow Analysis has been developed in the context of imperative languages while Control Flow Analysis in that of functional languages. At the beginning they have been applied in the development of optimizing compilers, which transform a program in another program whose execution gives the same results but in a shorter time. These techniques, even if with different intents, have been transferred to process calculi. For example Data Flow Analysis has been applied to CCS [83], π-calculus [84], bKlaim [79], and BioAmbients [91] while Control

Flow Analysis to π-calculus [10, 11], Mobile Ambients [80, 81], BioAmbients [82, 91], Beta Binders [8] and Brane Calculi [9].

These approaches are defined *semantic based*, i.e. the analysis information can be proved correct with respect to a semantic specification. We distinguish semantic based approaches from *semantic directed* ones where the specification is calculated from a semantic specification; this is the approach of *Abstract Interpretation* [25, 26].

In this dissertation we will make use of Control Flow Analysis and Abstract Interpretation, therefore, in the rest of the chapter we introduce them together with the mathematics necessary for their understanding.

2.3.1 Mathematical Preliminaries

We briefly introduce some basilar notions regarding ordered sets.

Definition 2.3.1 (Partial order) *A relation $\sqsubseteq$ is a partial order on a set A if*

1. $\forall x \in A, (x, x) \in \sqsubseteq$ *(reflexive)*
2. $\forall x, y \in A, (x, y) \in \sqsubseteq \wedge (y, x) \in \sqsubseteq \Rightarrow x = y$ *(antisymmetric)*
3. $\forall x, y, z \in A, (y, z) \in \sqsubseteq \Rightarrow (x, z) \in \sqsubseteq$ *(transitive)*

Definition 2.3.2 (Poset) *A poset is an ordered pair $(P, \sqsubseteq)$ where the relation $\sqsubseteq$ is a partial order on the set P.*

Definition 2.3.3 (Least element) *a is the least element of a partial order $(P, \sqsubseteq)$ if $\forall x \in P, a \sqsubseteq x$. If such an element exists we denote it as $\bot$.*

Definition 2.3.4 (Greatest element) *a is the greatest element of a partial order $(P, \sqsubseteq)$ if $\forall x \in P, x \sqsubseteq a$. If such an element exists we denote it as $\top$.*

Definition 2.3.5 (Lower bound) *Given a partial order $(P, \sqsubseteq)$ and $Y \subseteq P$, $a \in P$ is a lower bound of Y if $\forall x \in Y, a \sqsubseteq x$.*

Definition 2.3.6 (Upper bound) *Given a partial order $(P, \sqsubseteq)$ and $Y \subseteq P$, $a \in P$ is an upper bound of Y if $\forall x \in Y, x \sqsubseteq a$.*

Definition 2.3.7 (Least upper bound) *Let $(P, \sqsubseteq)$ be a partial order and $Y \subseteq P$ then a is the least upper bound (lub) of Y if*

1. *a is an upper bound of Y.*
2. *$\forall x : x$ is an upper bound of Y it holds that $a \sqsubseteq x$.*

We denote the least upper bound of Y with $\bigsqcup Y$. The binary least upper bound of $p, p' \in P$ is written $p \sqcup p'$.

Definition 2.3.8 (Greatest lower bound) *Let $(P, \sqsubseteq)$ be a partial order and $Y \subseteq P$ then a is the greatest lower bound (glb) of Y if*

1. *a is a lower bound of Y.*

2. *$\forall x : x$ is a lower bound of Y it holds that $x \sqsubseteq a$.*

We denote the greatest lower bound of Y with $\bigsqcap Y$. The binary greatest lower bound of $p, p' \in P$ is written $p \sqcap p'$.

Definition 2.3.9 (Lattice) *A poset $(P, \sqsubseteq)$ is a lattice if $\forall x, y \in P, x \sqcap y \in P$ and $x \sqcup y$ belong to P.*

Definition 2.3.10 (Complete lattice) *A poset $(P, \sqsubseteq)$ is a complete lattice if $\forall S \subseteq P, \bigsqcap S$ and $\bigsqcup S$ belongs to P.*

Note that if $(P, \sqsubseteq)$ is a complete lattice, then $\bot = \bigsqcup \emptyset = \bigsqcap P$ is the least element and $\top = \bigsqcap \emptyset = \bigsqcup P$ is the greatest element. Moreover note that every finite lattice is complete.

Definition 2.3.11 (Moore family) *A subset Y of a complete lattice $(P, \sqsubseteq)$ is a Moore family if it is closed under greatest lower bound, i.e.,*

$$\forall S \subseteq Y, \bigsqcap S \in Y$$

Note that a Moore family always contains a least element, $\bigsqcap Y$, and a greatest element, $\bigsqcap \emptyset = \top_P$; thus it is never empty.

Definition 2.3.12 (Monotone function) *A function $f : P \to Q$, with $(P, \sqsubseteq_P)$ and $(Q, \subseteq_Q)$ posets, is monotone if*

$$\forall x, y \in P, x \sqsubseteq_P y \Rightarrow f(x) \sqsubseteq_Q f(y)$$

Definition 2.3.13 (Extensive operator) *A function $f : P \to P$ with $(P, \sqsubseteq_P)$ a poset is an extensive operator if*

$$\forall x \in P, x \sqsubseteq_P f(x)$$

Definition 2.3.14 (Reductive operator) *A function $f : P \to P$ with $(P, \sqsubseteq_P)$ a poset is a reductive operator if*

$$\forall x \in P, f(x) \sqsubseteq_P x$$

Definition 2.3.15 (Idempotent operator) *A function $f : P \to P$ is an idempotent operator if*

$$\forall x \in P, f(f(x)) = f(x)$$

Definition 2.3.16 (Chain) *A subset Y of a partial order $(P, \sqsubseteq)$ is a chain if it is totally ordered, i.e.*

$$\forall x, y \in Y, x \sqsubseteq y \text{ or } y \sqsubseteq x$$

A sequence $(p_n)_n = (p_n)_{n \in N}$ of elements in P is an ascending chain *if $n \leq m$ implies $p_n \sqsubseteq p_m$.*

Definition 2.3.17 (Continuous function) *Consider a function $f : P \to Q$ with $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$ partial orders. The function f is continuous if for all the ascending chains $(p_n)_n$ of $(P, \sqsubseteq_P)$ it holds that $f(\bigsqcup_n p_n) = \bigsqcup_n (f(p_n))$.*

Definition 2.3.18 (Fixed point) *Consider a monotone function, $f : P \to P$ on a complete lattice $(L, \sqsubseteq)$. A fixed point of f is an element $p \in P$ such that $f(p) = p$.*

Theorem 2.3.19 (Knaster-Tarski [63]) *Any monotone and continuous function $f : P \to P$ on a complete lattice $(P, \sqsubseteq)$ has a unique least fixed point $lfp(f) = \bigsqcup_{i \geq 0} f^i(\bot)$.*

Note that if we say the least fixed point greater than p with $p \in P$ we refer to $\bigsqcup_{i \geq 0} f^i(p)$ and we refer it as $lfp_p(f)$.
Note that if L satisfies the ascending chain condition, then $LFP(f)$ can be computed in a finite number of step.

2.3.2 Control Flow Analysis

One of the main advantages of the Flow Logic is that it makes a clear separation between the specification of the analysis and its implementation. In this way it is possible to concentrate on the specification without thinking about issues related with its implementation.

In order to reason about programs we need a language for representing their features. We call the universe of discourse *analysis domain*. The analysis domain L is usually required to be a complete lattice.

The elements $\mathcal{A} \in L$ are in relationship with those of LANG which represents the set of all the writeable programs. This connection is expressed through the *acceptability judgment*:

$$\mathcal{A} \vDash P$$

which means that $\mathcal{A}$ is an acceptable analysis estimate for the program $P \in$ LANG.

The judgment is usually defined going through the syntactic constructs ϕ of LANG and specifying a clause for each of them:

$$\mathcal{A} \vDash \phi(P_1 \cdots P_n) \text{ iff (some formula } \varphi \text{ with } \mathcal{A} \vDash P' \text{ for some sub-programs } P' \in \{P_1, \ldots, P_n\})$$

With these ingredients we can define a Control Flow Analysis as a Flow Logic. We are only interested in specification which are *syntax directed*, i.e. each P' sub-program which occurs in φ is one of the P_i occurring in $\phi(P_1 \cdots P_n)$, in this particular case the acceptability judgment can be adequately defined by ordinary induction. Such specification are called *compositional*.

A Control Flow Analysis in order to be considered a Flow Logic must have some properties:

Well-definedness: the analysis must be well-defined, i.e. for every combination of $P \in$ LANG and $\mathcal{A} \in L$ it must hold that $\mathcal{A} \vDash P$ or $\mathcal{A} \nvDash P$. This result for compositional specification comes immediately by induction on the structural definition of P.

Semantic correctness: we say that an analysis is semantically correct if, in the case it is acceptable for P, the same holds also for all the possible evolutions of P. Usually this property is shown through a *subject reduction* result:

$$\text{if } \mathcal{A} \vDash P \text{ and } P \to Q \text{ then } \mathcal{A} \vDash Q$$

Clearly the full semantics correctness, sometimes called *semantic soundness* follows by the transitive closure of the subject reduction result.

Moore family Property: it is desirable for a program P that among all the elements of L there is an acceptable analysis estimate, and moreover if more than one exists it would be nice to ensure that one of this is the best. This is the case if:

$$\{\mathcal{A} \mid \mathcal{A} \vDash P\} \text{ constitutes a Moore Family for all } P$$

In fact a Moore family cannot be empty, as it always contains a greatest element $\bigsqcap \emptyset = \top_L$ (which is the less informative acceptable analysis estimate, and it always has a least element $\bigsqcap\{\mathcal{A} \mid \mathcal{A} \vDash P\}$ which instead is the most informative acceptable analysis estimate.

The implementation of a Flow Logic specification is enabled by a simple change of viewpoint: Intuitively, the acceptability judgement

$$\mathcal{A} \vDash P \text{ iff } \varphi$$

associates each program, P, with a formula, φ, such that an analysis estimate, $\mathcal{A}$, is acceptable for P if and only if, A constitutes a model of φ. Thus, as long as we are able to compute the appropriate φ, the remaining task of finding a suitable model, A, can be left to an auxiliary logical solver. We precise that this holds only for syntax directed Flow Logics (those we are interested in).

2.3.3 Abstract Interpretation

The formal verification of a program consists in proving that its semantics satisfies its specification. The problem is that often to perform this verification is not trivial. One of the approaches for formally verifying properties of program is a technique called *abstract interpretation* [24, 25, 26]. Abstract interpretation is a mathematical framework which allows to abstract from details which are not necessary in order to verify the property we are interested in.

A simple application of such a technique is the sign rules for the multiplication operation on integer numbers. We know that when we multiply two numbers in order to know the sign of the result of the multiplication, we do not need to know the exact values of the multiplied numbers (which in this case is the useless detail), we only need to know theirs sign (which in this case is the interesting information). This is a very simple example of abstract interpretation but it is intuitive and we will use it in order to introduce the terminology related with this technique.

If we analyse the problem from a computer science point of view, we are in the following situation: we have a program which computes the multiplication between two numbers whose semantics consists in the classical multiplication operation. Now the idea is to start from this semantics, which we call *concrete semantics*, and define an *abstract semantics* which only takes care of the details we are interested in (i.e. the signs of the numbers). Our concrete semantics can be thought as a function f which maps two sets of integers X and Y to another set of integers $Z = \{z \mid \exists x \in X\ y \in Y : x \cdot y = z\}$. The domain over which the concrete semantics works is called *concrete set*, in this example it is $\mathcal{P}(\mathbb{Z})$, the power set of the set of all integer numbers. We work on $\mathcal{P}(\mathbb{Z})$ rather then on $\mathbb{Z}$ because we want the concrete semantics works on a complete lattice which in this case is defined as $(\mathcal{P}(\mathbb{Z}), \subseteq)$ (in this way it is possible to ensure some interesting properties, as we will appreciate later).

The abstract semantics $f^\sharp$, in contrast with f, is defined on the *abstract domain*. We obtain the abstract domain starting from the concrete one and eliminating the uninformative details.

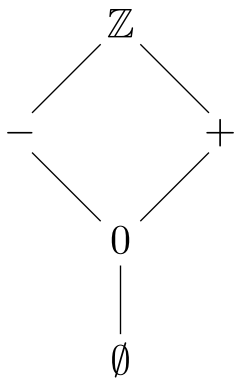


Figure 2.1 – Abstract domain.

$f^\sharp$	$\mathbb{Z}$	$+$	$-$	0	$\emptyset$
$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}$	$\mathbb{Z}$	0	$\emptyset$
$+$	$\mathbb{Z}$	$+$	$-$	0	$\emptyset$
$-$	$\mathbb{Z}$	$-$	$+$	0	$\emptyset$
0	0	0	0	0	$\emptyset$
$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Table 2.5 – Abstract semantics.

In this case we want to drop the values of the numbers and keep their signs. Our purpose is to define an abstract domain which has a representative for each element of the concrete domain. Therefore we define a + element which represents sets only made of positive numbers, a − element for sets made of negatives numbers, and 0, which represent the number 0 which is neither positive nor negative. In order to complete the abstract domain we add the $\mathbb{Z}$ element which represents set made of both negative and positive numbers and then the empty set $\emptyset$ which represents the empty set. Also this elements are organized in a complete lattice as shown if Figure 2.1.

The fact that all the elements of the concrete set are mapped to elements of the abstract domain is formalized with the definition of a function which we call *abstraction function* and we usually regard as α. We also have the counterpart of α which maps all the elements of the abstract domain to elements of the concrete domain. This function is called *concretization function* and is usually regarded as γ.

Now we define the abstract semantics as the function which given the signs of the elements involved in the multiplication operations (which means two elements of the abstract domain) returns the sign of the results. The definition of the function is given by the double entry Table 2.5.

Which is the advantage of setting up such a framework? Let us image that we have two infinite sets of integer numbers and we want to know which are the signs of the results which we obtain by multiplying the numbers of the two different sets. In order to do this with our concrete semantics we would have to perform an infinite number of multiplications. Obviously

this is not possible. Instead, if we compute the abstraction of the two sets with the α function, we obtain their abstract images. These images, taken as input by the abstract semantics, tell us in one step which is the sign of the results.

In the following section we propose an application of the abstract interpretation technique to the κ-calculus. Before starting we need to introduce some mathematical notions and theorems which formalize the correctness of an abstraction and express the relations which exist among concrete and abstract domains, concretization and abstraction functions and concrete and abstract semantics.

In the example of the sign rule we explained the existence of a relation which binds the elements of the concrete domain to the elements of the abstract domain and vice versa. This relation is formalized through the notion of *Galois insertion* which is a particular case of the more general notion of *Galois connection*.

Definition 2.3.20 (Galois connection) *If $(C, \sqsubseteq_C)$ and $(A, \sqsubseteq_A)$ are complete lattices then (C, γ, A, α) is a Galois connection if:*

1. *$\alpha : C \to A$ and $\gamma : A \to C$ are monotone.*
2. *$\gamma \circ \alpha$ is extensive.*
3. *$\forall a \in A, \alpha \circ \gamma(a) \sqsubseteq_A a$.*

Definition 2.3.21 (Galois insertion) *If $(C, \sqsubseteq_C)$ and $(A, \sqsubseteq_A)$ are complete lattices then the tuple (C, γ, A, α) is a Galois insertion if:*

1. *$\alpha : C \to A$ and $\gamma : A \to C$ are monotone.*
2. *$\gamma \circ \alpha$ is extensive.*
3. *$\forall a \in A, \alpha \circ \gamma(a) = a$.*

Through the notion of Galois insertion we give one of the possible definition of correctness of an abstraction.

Definition 2.3.22 (Correct abstraction) *Given a complete lattice $(C, \sqsubseteq)$, a function $f : C \to C$, (C, γ, A, α) a Galois insertion and a function $f^\sharp : A \to A$ we say that (C, γ, A, α) and $f^\sharp$ are a correct abstraction of f if $f \circ \gamma \sqsubseteq \gamma \circ f^\sharp$.*

We now introduce some theorems which are necessary in order to present the work of the next section. The following results are presented in [24, 25].

Theorem 2.3.23 *Given a complete lattice $(C, \sqsubseteq)$, a function $f : C \to C$ and a Galois insertion (C, γ, A, α) then the best correct approximation of f is the function $\alpha \circ f \circ \gamma$.*

Theorem 2.3.24 *Given a Galois insertion (C, γ, A, α), a function $f : C \to C$ and a correct abstraction $f^\sharp$ of f if f and $f^\sharp$ are monotone than $\forall x \in C$ it holds that*

$$\alpha(lfp_{\{x\}} f) \subseteq lfp_{\alpha(\{x\})} f^\sharp$$

and

$$lfp_{\{x\}} f \subseteq \gamma(lfp_{\alpha(\{x\})} f^\sharp)$$

Definition 2.3.25 (Additive function) *Given a complete lattice $(P, \sqsubseteq)$, a function f is additive if:*

$$\forall Y \subseteq P, f(\bigsqcup Y) = \bigsqcup_{x \in Y} f(x)$$

Theorem 2.3.26 *Given two complete lattices $(C, \sqsubseteq_C)$ and $(A, \sqsubseteq_A)$, and an additive function $\alpha : C \to A$ then exists $\gamma : A \to C$ such that (C, γ, A, α) is a Galois connection and we can obtain γ starting from α as follows:*

$$\gamma(c) = \bigsqcup \{x \mid \alpha(x) \sqsubseteq c\}$$

Definition 2.3.27 (Upper closure) *Given a poset $(P, \sqsubseteq)$, the function $f : P \to P$ is an upper closure of P if f is extensive, monotone and idempotent.*

Theorem 2.3.28 *(C, γ, A, α) is a Galois insertion iff $\gamma \circ \alpha$ is an upper closure operator for C.*

2.3.4 Abstract Interpretation applied to κ-calculus

The work we present here is taken from [32]. In that paper the considered κ-calculus is a subset of the one we presented in Section 2.2.4. In particular agent creation and deletion is not considered and, in the rules, it is not possible to use the wildcard $(-)$. However, as stated by the authors, the framework can be easily extended in order to work with these features of the language.

Our purpose is to compute the set of complexes which can be generated by the evolution of a system S_0 associated with the set of rules R. The most immediate approach consists in the extraction of the complexes from all the solutions which can be generated through the application of the rules of R to S_0.

However it is possible for S_0 to generate an infinite number of solutions and thus we cannot apply this method. In order to solve the problem we accept to obtain an over-approximation of the complexes that can be generated via abstract interpretation.

In this example the concrete set is $\mathcal{P}(\texttt{Sol})$. The function which we want to abstract is the function $POST^R : \mathcal{P}(\texttt{Sol}) \to \mathcal{P}(\texttt{Sol})$ which is our concrete semantics and can be defined as:

$$POST^R(X) := X \cup \{S' \mid \exists S \in X, \exists r \in R : S \xrightarrow[\kappa]{r} S'\}$$

Definition 2.3.29 (Derivative) *We define the set of derivatives of S_0 with respect to the set of rules R as the least fix point of $POST^R$ greater then $\{S_0\}$ written as*

$$lfp_{\{S_0\}} POST^R$$

Notation 2.3.30 *We write $[\![F]\!] \in [\![E]\!]$ if there is a mixture E' such that $E \equiv_\kappa F, E'$.*

We choose as abstract domain $\mathcal{P}(\texttt{Com})$ and define the abstraction function $\alpha_c : \mathcal{P}(\texttt{Sol}) \to \mathcal{P}(\texttt{Com})$ and the concretization function $\gamma_c : \mathcal{P}(\texttt{Com}) \to \mathcal{P}(\texttt{Sol})$ as:

$$\begin{array}{rcl} \alpha_c(X) & := & \{c \in \texttt{Com} \mid \exists S \in X : c \in S\} \\ \gamma_c(Y) & := & \{S \in \texttt{Sol} \mid c \in S \Rightarrow c \in Y\} \end{array}$$

Note that if we follow what before we called "the most intuitive approach" we can obtain the set of the reachable complexes as

v	::=	$A(\sigma)$	(*view*)	λ	::=	$\epsilon_\lambda \mid x.A$	(binding state)
σ	::=	$\varepsilon \mid s, \sigma$	(interface sequence)	ι	::=	$\epsilon_\iota \mid m$	(internal state)
s	::=	x_ι^λ	(interface)				

Table 2.6 – Syntax of views where $A \in \texttt{Agent}$, $i \in \mathbb{N}$, $x \in \texttt{Interface}$ and $m \in \texttt{Value}$.

$$\texttt{Com}_{S_0}^R = \alpha_c(lfp_{\{S_0\}} POST^R)$$

At this point we can easily verify that $(C, \gamma_c, A, \alpha_c)$ is a Galois insertion and thus by Theorem 2.3.23 we know that the best abstraction of $POST^R$ is $POST_c^R = \alpha_c \circ POST^R \circ \gamma_c$. We can write $POST_c^R$ as:

$$POST_c^R(X) = X \cup \{c \in \texttt{Com} \mid \exists [c_1], \ldots, [c_m] \in X \; \exists r \in R \; \exists S \in \texttt{Sol} : [c_1, \ldots, c_m] \xrightarrow[\kappa]{r} S \wedge c \in S\}$$

By Theorem 2.3.24 we can now state that $\alpha_c(lfp_{\{S_o\}} POST^R) \subseteq lfp_{\alpha_c(\{S_o\})} POST_c^R$ and thus that $\texttt{Com}_{S_0}^R \subseteq lfp_{\alpha_c(\{S_o\})} POST_c^R$. Therefore we obtained an over-approximation of the reachable complexes. Now the problem is that $lfp_{\alpha_c(\{S_o\})} POST_c^R$ can be infinite as well. For example it is the case if we have a model which allows agent polymerization. Consider the case where we have the set of rules $R = \{A(a), A(b) \rightarrow A\left(a^1\right), A\left(b^1\right)\}$ and $\Sigma(A) = \{a, b\}$. Here $lfp_{\alpha(\{x\})} POST_c^R$ contains all the rings and chains made of an arbitrary number of agents A.

At this point the purpose is to set up a finite approximation of $lfp_{\alpha_c(\{S_o\})} POST_c^R$. The idea is to only retain from a solution the information which is local to the agents and which we call *agent view* (as in [42]). Specifically, we replace each binding state in an expression with its associated *typed link*, i.e. the site and agent names of the opposite end of the link. The function which generates the view starting from a pattern is β. Here is an example of the application of β to a pattern:

$$\beta(A\left(a^1, b_m\right), B\left(c, d^1\right)) = A\left(a^{c.B}, b_m\right), B\left(c, d^{a.A}\right)$$

The syntax of views is given in Table 2.6, and the structural equivalence $\equiv_\kappa^\sharp$ which allows to reorder interfaces in a view is given by the smallest equivalence relation which satisfies the following:

$$A\left(\sigma, s, s', \sigma\right) \equiv_\kappa^\sharp A\left(\sigma, s', s, \sigma\right)$$

Operations on solutions transfer naturally to sequences of views. In particular one can define an abstract transition step between sequences of views (see Table 2.7) that tests from conditions over the view relation $\vDash_\kappa^\sharp$, and either changes the internal state of a site, or adds/removes the appropriate types links in the binding state of two modified views.

Now we define the abstraction that collects the set of views that can be built during a computation sequence. As a first step, referring to the set of equivalence classes of views as $\texttt{View}$, we define the abstraction function $\alpha : \mathcal{P}(\texttt{Com}) \rightarrow \mathcal{P}(\texttt{View})$ as:

$$\alpha(X) = \{[v_i] \mid \exists [c] \in X, \beta(c) = v_1, \ldots, v_n\}$$

$$x_\iota^\lambda \;\vDash_\kappa^\sharp\; x_\iota^\lambda \qquad \sigma \;\vDash_\kappa^\sharp\; \varepsilon$$

$$\frac{s \vDash_\kappa s_l \qquad \sigma \vDash_\kappa^\sharp \sigma_l}{s, \sigma \vDash_\kappa^\sharp s_l, \sigma_l} \qquad \frac{\sigma \vDash_\kappa^\sharp \sigma_l}{A(\sigma) \vDash_\kappa^\sharp A(\sigma_l)}$$

$$\begin{aligned} x_\iota^\lambda[x_{\iota_r}^{\lambda_r}]^\sharp &= x_{\iota_r}^{\lambda_r} \\ x_\iota^\lambda[x_{\iota_r}^{-}]^\sharp &= x_{\iota_r}^{\lambda} \\ \sigma[\varepsilon]^\sharp &= \sigma \\ s, \sigma[s_r, \sigma_r]^\sharp &= s[s_r]^\sharp, \sigma[\sigma_r]^\sharp \\ A(\sigma)\,[A(\sigma_r)]^\sharp &= A\left(\sigma[\sigma_r]^\sharp\right) \end{aligned}$$

$$\frac{\beta(E_l) = v_l^1, \ldots, v_l^n \qquad \beta(E_r) = v_r^1, \ldots, v_r^n \qquad v^i \vDash_\kappa^\sharp v_l^i}{[v^1], \ldots, [v^n] \xrightarrow[\kappa]{r}{}^\sharp [v^1[v_r^1]^\sharp], \ldots, [v^n[v_r^n]^\sharp]}\; r = E_l \to E_r$$

Table 2.7 – Abstract semantics.

By construction of α we can verify that it is addictive and thus we can apply the result of Theorem 2.3.26 and say that (C, γ, A, α) is a Galois connection where the function $\gamma : \mathcal{P}(\texttt{View}) \to \mathcal{P}(\texttt{Com})$ is defined as:

$$\gamma(Z) = \bigcup \{X \in \mathcal{P}(\texttt{Com}) \mid \alpha(X) \subseteq Z\}$$

Starting from this we can easily see that $\gamma \circ \alpha$ is an upper closure operator for C and thus conclude by Theorem 2.3.28 that (C, γ, A, α) is a Galois insertion. Now we define the abstract function $POST_v^R$:

$$POST_v^R(Z) := Z \cup \{u_i \in \texttt{View} \mid \exists v_1, \ldots, v_n \in Z\ \exists r \in R : v_1, \ldots, v_n \xrightarrow[\kappa]{r}{}^\sharp u_1, \ldots, u_n\}$$

This function is monotone and we can verify that it, together with $(\mathcal{P}(\texttt{Com}), \gamma, \mathcal{P}(\texttt{View}), \alpha)$, is a correct abstraction of $POST_c^R$ given that it holds that

$$\forall x \in \mathcal{P}(\texttt{View}), (POST_c^R \circ \gamma)(x) \subseteq (\gamma \circ POST_v^R)(x)$$

Now we have all the necessary for applying the result of Theorem 2.3.24 and obtaining the soundness of our abstraction:

$$lfp_{\alpha_c(S_0)} POST_c^R \subseteq \gamma(lfp_{\alpha(\alpha_c(S_0))} POST_v^R)$$

Thus, starting from the complexes which appear in the system S_0 the views generated by the abstract system reconstruct, via γ, a superset of the generated complexes. Given that `Agent` is finite the same holds for `View`, this implies that $lfp_{\alpha(\alpha_c(S_0))} POST_v^R$ is finite as well, given that it is included in `View`. The same is not true for its concretization, in any case we have the possibility to generate a finite set which describe an over-approximation of the complexes that can be generated starting from an initial solution S_0 which evolves through the rules of R.

2.4 Markov and Semi-Markov Processes

In this section we briefly recall two particular classes of stochastic processes: continuous time Markov processes and a particular case of semi-Markov processes. We are interested in them because models defined with stochastic process calculi have an underling stochastic process belonging to these classes (as it is for BlenX).

	A	B	C
A	$-(r_1 + r_2)$	r_1	r_2
B	0	$-r_3$	r_3
C	0	0	0

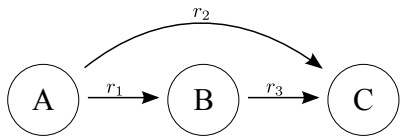


Figure 2.2 – Q-matrix of a CTMC (on the left) and its graphical representation (edges with rate zero are omitted).

2.4.1 Continuous Time Markov Processes

A *continuous time Markov process* is a particular case of stochastic process $\{X_t \mid t \geq 0\}$ (see [87] for an exhaustive definition of stochastic process) which takes values from a countable or finite set $S = \{s_1, \dots, s_n, \dots\}$ called *state space* and satisfies the *Markov property*. This means that for all different n, t_i, s_i the *memoryless condition* holds:

$$P(X_{t_n} = s_n \mid X_{t_0} = s_0, \dots, X_{t_{n-1}} = s_{n-1}) = P(X_{t_n} = s_n \mid X_{t_{n_1}} = s_{n_1})$$

In words we have that a process is Markovian if the transition from one state to another does not depends on the past history but depends only on the current state.

Continuous time Markov processes are described with *continuous time Markov chain* (CTMC). CTMCs are characterized by their *generator matrix*, or *Q-matrix*, which is a matrix whose rows and columns are indexed by S, and whose entry $q_{ij} \geq 0, i \neq j$ represents the rate of an exponential distribution associated to the transition from state s_i to state s_j. The value q_{ii} (*exit rate*) is set equal to $-\sum_{j \neq i} q_{ij}$ (assumed to be finite for each i), so that each row of the Q-matrix sums up to zero. A good way to visualize a CTMC is to associate it with a complete graph, whose nodes are indexed with elements of S, and whose edges are labeled with the corresponding rates of the Q-matrix (edges with rate zero can be omitted). The time spent from a process in a state is called *sojourn time*. The sojourn time of each state is distributed as an exponential distribution with rate equal to its exit rate.

The memoryless condition of CTMC can be described as a race condition. We consider all the positive entries in the generator matrix Q, associating to each $q_{ij} > 0$ an exponentially distributed random variable $T_{ij} \sim Exp(q_{ij})$. When the chain is in state s_i, then a race condition begins among all random variables $T_{ij},\ s_j \in S, q_{ij} > 0$. Notice that each random variable T_{ij} corresponds to an edge in the support graph associated to the CTMC. The race is won by the fastest variable, i.e. the one realizing $T_i = \inf_{s_j \in S}\{T_{ij}\}$. If T_{ik} is such variable, then the system moves in state s_k in $T = T_{ik}$ units of time. When the system reaches state s_k, then a race condition between random variables exiting from s_k begins anew.

2.4.2 Semi-Markov Processes

A *semi-Markov process* is a stochastic process $\{X_t \mid t \geq 0\}$ which takes values from a countable or finite set $S = \{s_1, \dots, s_n, \dots\}$ and for which the following property holds:

$$\begin{aligned} &P(X_{t_n} = s_n, \theta_n - \theta_{n-1} < \tau \mid X_{t_0} = s_0, \dots, X_{t_{n-1}} = s_{n-1}) \\ &\quad = P(X_{t_n} = s_n, \theta_n - \theta_{n-1} < \tau \mid X_{t_{n_1}} = s_{n_1}) \end{aligned}$$

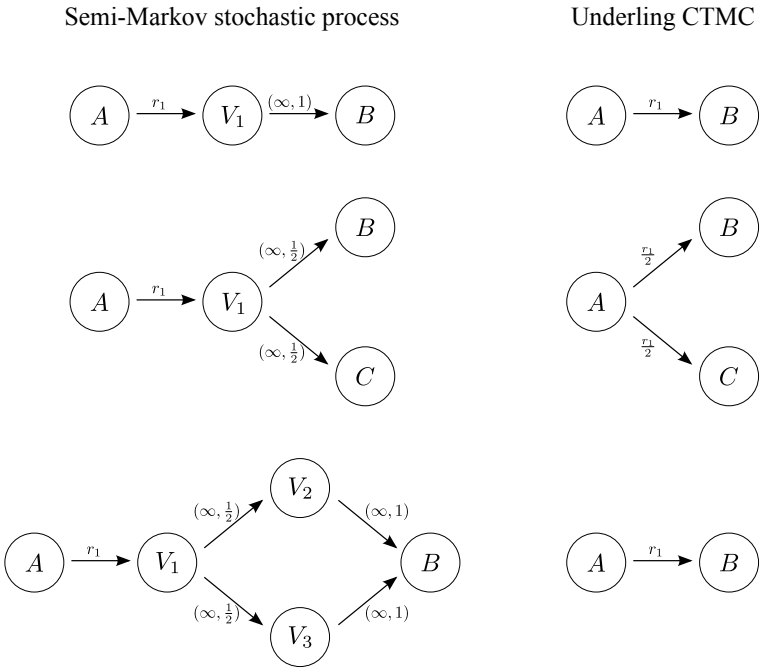


Figure 2.3 – Semi-Markov stochastic processes (on the left) and their underling CTMCs (on the right).

where θ_n is the time when the process change state from X_{t_n-1} to X_{t_n}. Thus a semi-Markov process differs from a continuous time Markov process because the probability to move in a state s_n does not depend only from the state in which the process is but it also depends on the time the process has already spent in that state. Thus we loose the memoryless property and we can deal with transition times which are not exponentially distributed.

Our interest for these processes arise because in some process calculi (and not only, see for example GSPN in [56]) two kinds of transitions exist: those fired following an exponential distribution and those which are immediate, i.e., as soon as they become active they fire (with different strategies to solve conflicts arising from simultaneous activations). This makes semi-Markov processes the right candidate to describe their behaviour.

This particular category of semi-Markov processes partition the state set S in *vanishing states* (which perform immediate actions) and *tangible states* (which does not). The former have a null sojourn time, the latter a negatively exponentially distributed one with rate equal to its exit rate.

Intuitively the behaviour of this processes can be described as a sequence of immediate "state jumps" interleaved by "rest times" in some of these states (the tangible states). In many approaches (and this is the case of BlenX) the system is fully described by its permanence in tangible states, as the visits of vanishing states do not add any information. This, under the sufficient assumption that the semi-Markov process starts in a tangible state, allows us to give a characterization of it through a CTMC. In Figure 2.3 we have three examples of semi-Markov processes with their associated underling CTMC. An interesting observation is that different semi-Markov processes (e.g. the first and the third processes of Figure 2.3) can have the same underling CTMC, therefore, even if they visit different sequences of vanishing states when moving

from a tangible state to another, they can be considered to have the same behaviour. In order to read some more details about the computation of the underling CTMC look at [56].

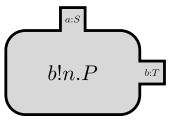
Chapter 3

BlenX

In recent years there has been a lot of interest in using process algebras — traditionally used for describing concurrent computation systems — to model *biological systems*. A number of languages have been developed, including stochastic π-calculus [98], BioAmbients [100], beta-binders [105], and BioPEPA [23]. A common feature in all of these is that they are *stochastic* — that is to say, they describe systems that evolve probabilistically over time. The semantics is usually given in terms of a Continuous-Time Markov Chain (CTMC).

BlenX is a recent language that continues in this tradition, but with the addition of a useful paradigm that is biologically motivated. Rather than just describing a system of species that can react with one another and evolve over time, it introduces a notion of *structural interaction* between species. More specifically, a species has both an internal state and interfaces — not only can its internal state evolve over time, but it can interact with other species to form *complexes*. This allows to model biological structures such as chains of proteins or clusters of cells [66].

Let us consider an example of *box*, the elementary component of a BlenX system. Boxes are defined formally using a textual description (which we present in the next section). To make it easier to understand the structure of the boxes, we will often draw them graphically. In the following we see how a box looks using both approaches:



$$\{(a,\, S)(b,\, T)\}[\, b!n.\, P\,]$$

A box consists of a number of *interfaces*, and an internal process made of a number of *processes* running in parallel. Processes are written in a language similar to the π-calculus, and can communicate with one another over *channels*. Each interface has a *name* and a *sort* — in the example above, there are an interface a of sort S, and an interface b of sort T. Internal processes can communicate between boxes over interface names. We use the sorts to determine which interfaces can bind, unbind and communicate, and the sort of an interface can change over time.

There are three fundamental ways in which boxes can interact, and this is illustrated by the three stages shown in Figure 3.1:

(a) Two interfaces can *bind* together if their sorts have a *binding affinity* with one another, and neither interface is already bound. Figure 3.1(a) shows interface b of box A and c of box B binding together.

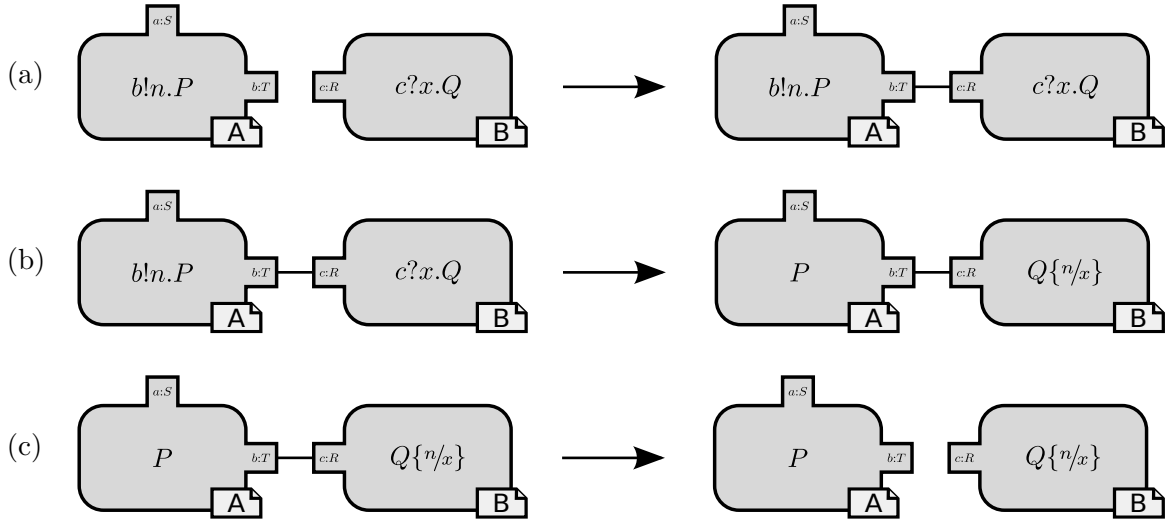


Figure 3.1 – Interaction between BlenX boxes. Note that we applied labels to boxes in order to easily refer them.

(b) Two processes in different boxes can *communicate* with one another, by sending and receiving over interface names. They can do so only if the sorts of the interfaces have *communication affinity*. Figure 3.1(b) shows the two boxes communicating over the binding b-c, such that the process in box A outputs n on interface b ($b!n$), and the process in box B receives n from interface c into the variable x ($c?x$).

(c) Two bound interfaces can *unbind* if their sorts have an *unbinding affinity* with one another. Figure 3.1(c) shows the b and c interfaces unbinding.

An *action* of an internal process (like outputs and inputs of Figure 3.1) can be prefixed by a *condition*. This is a Boolean expression, including atomic tests on the state of the interfaces of the form '(a, T)' (a has the sort T), '$(a, \odot)$' (a is unbound), and '$(a, \otimes)$' (a is bound). As an illustration, we could modify the internal process of A in Figure 3.1 to '$\langle(a, \odot)\rangle b!n.P$', which would mean that it can only output on interface b if interface a is unbound. Actions that are prefixed with a condition should be interpreted as *guarded commands*, in that they can only execute if the condition is true.

3.1 Syntax and Semantics

In the previous section we gave an informal description of the BlenX language. Now we will present a formal definition. The BlenX syntax is enriched with real numbers named *rates* — usually denoted with r — which are typical of languages used for describing systems from a quantitative point of view. These rates are associated with actions and are parameters of negative exponential distributions. We use them in order to resolve non determinism when more than one action can take place.

Let Name be a set of *names* ranged over by n, m, etc. and let Sort be a set of *sorts* ranged over by T, U, etc. such that $\mathsf{Name} \cap \mathsf{Sort} = \emptyset$. We present the syntax of bio-processes and

B	::=	**Bio-processes**
	$I[P]$	box
	$B \parallel B$	composition
P	::=	**Processes**
	M	capability
	$P \mid P$	composition
M	::=	**Capabilities**
	nil	empty
	$*\pi.\, P$	replication
	$\pi.\, P$	action
	$M + M$	choice
	$\langle C \rangle M$	condition

π	::=	**Prefixes**
	$n?y$	input
	$n!m$	output
	$\mathsf{ch}(r, a, S)$	change
C	::=	**Conditions**
	true	true
	(a, T)	test sort
	$(a, \odot)$	free
	$(a, \otimes)$	bound
	$\neg C$	negation
	$C \wedge C$	conjunction
e	::=	**Event**
	$I'[\,P'\,] \blacktriangleright_r B'$ with $B' = I[\,P\,] \parallel B$	event

Table 3.1 – Bio-processes and events syntax.

events in Table 3.1. Bio-processes of the set $\mathtt{Box}$ are generated by the non-terminal symbol B and we usually refer to them with B, B_1, B', etc. A bio-process can be either a box, $I[\,P\,]$, or a parallel composition of bio-processes, $B \parallel B$. In a box $I[\,P\,]$, the set I describes the interaction capabilities and P describes the internal behaviour. The non-empty set I contains interfaces of the form $(a, T)^r$. The name a is called the *subject* of the interface and is used by the internal process for performing actions over it and testing its state. T is a sort, and represents the interaction capabilities of an interface with respect to the other interfaces of the system. Finally, r describes the rate at which internal communication over the interface takes place. The sets $\mathsf{sub}(I)$ and $\mathsf{sorts}(I)$ represent the set of subjects and the set of sorts used in I, respectively. We often write $(a_1, T_1)^{r_1} \ldots (a_n, T_n)^{r_n}$ for $\{(a_1, T_1)^{r_1}, \ldots, (a_n, T_n)^{r_n}\}$. We refer the set of BlenX boxes as $\mathcal{B}$.

The internal process of a box is generated by the non-terminal symbol P. We denote the set of these processes as $\mathcal{P}$ and we usually refer to them with P, P_1, P', etc. A process can be either a capability (M) or a parallel composition of two processes ($P_1 \mid P_2$). A capability can be one of the following: the empty process (nil), an action-guarded process ($\pi.\, P$), the replication of an action-guarded process ($*\pi.\, P$), a non-deterministic choice between capabilities ($M_1 + M_2$) or a capability guarded by a condition ($\langle C \rangle M$). An action π can take the form $n!m$, $n?y$ or $\mathsf{ch}(r, a, T)$. The first two are symmetric actions corresponding respectively to input and output and are used for exchanging information between two processes which run in parallel: $n!m.\, P_1 \mid n?y.P_2$. In this particular case $n!m$ sends the channel name m over the channel n, and $n?y$ receives the name m which becomes bound to the variable y in the process P_2. The third action, $\mathsf{ch}(r, a, T)$, is used to change the sort associated with the interface a to T. If no interfaces have subject a or another interface already has the sort T, the change action cannot take place and the execution of the continuing process P is blocked.

The guard $\langle C \rangle$ allows the execution of a capability if the condition C evaluates to true.

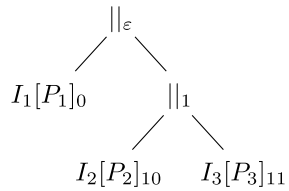


Figure 3.2 – Labelled syntax tree of a bio-process.

Atomic conditions check for properties of the interfaces, i.e. whether an interface has a given sort (a, T), is free $(a, \odot)$ or is bound $(a, \otimes)$. The set of conditions generated by the syntax is regarded as *Cond*.

Replication makes it possible to implement processes with infinite behaviour; it is comparable with the concept of recursion. A better understanding of this construct will be possible later on, when we introduce the semantics of the language and give some examples of its usage.

The syntax in Table 3.1 allows to write more bio-processes then we would like to. For this reason we introduce a notion of well-formedness, which gives a number of constraints that a bio-process must fulfill in order to be part of a system. The set of well-formed boxes is regarded as `Box`.

Definition 3.1.1 *A bio-process is well-formed if its component boxes are well-formed. A box* $I[\,P\,]$ *is well-formed if* $I \neq \emptyset$ *and for all distinct elements of* I, $(a_1, T_1)^{r_1}$ *and* $(a_2, T_2)^{r_2}$, *it holds that* $a_1 \neq a_2$ *and* $T_1 \neq T_2$.

Two boxes of a bio-process B can be linked through their interfaces. The environment component ξ records these links. To do this it needs to have pointers to the boxes involved in the link. Formally, these pointers are labels that linearly describe the position of a box inside the syntax tree of the bio-process it belongs to. Given that boxes can only be composed using the $\|$ operator, the abstract syntax tree of a bio-process is a binary tree. To describe the positions of its nodes we label each box with a string in the language generated by the regular grammar $(0|1)^*$. We denote the set of label as *Label*. The empty string ε is associated with the root of the node, and the other nodes are associated with sequences of zeros and ones. The left child of the root is associated with 0, and the right child with 1. In general each node is labelled by the concatenation of the label of its parent with 0 if it is the left child, and with 1, if it is the right child. Figure 3.2 shows an example illustrating the abstract syntax tree of the bio-process $(I_1[\,P_1\,] \parallel (I_2[\,P_2\,] \parallel I_3[\,P_3\,]))$, whose nodes are labelled as described.

Using these labels the environment ξ stores links among boxes as sets made up of two pairs: $\{(\gamma_1, S_1), (\gamma_2, S_2)\}$. Such a set says that the box labelled with γ_1 is linked through the interface associated with the sort S_1 to the interface associated with S_2 of the box labelled γ_2. Note that in the context of a well-formed bio-process, (γ_1, S_1) identifies a unique interface on the box labelled with γ_1, since a well formed box cannot have two interfaces of the same sort.

Boxes can also use interfaces for exchanging messages. A box can send/receive messages through an interface by performing an output/input action over its subject. Note that interface subjects can also be used for exchanging messages inside a box; in this case they act like an internal communication channel.

Two interfaces can also bind and unbind. The interaction capabilities of the interfaces depend on the sorts they are associated with. Information regarding the compatibilities between sorts

a) axioms for processes

1. $(\mathcal{P}, |, \mathsf{nil})$ is a commutative monoid under $\equiv_p$
2. $(\mathcal{P}, +, \mathsf{nil})$ is a commutative monoid under $\equiv_p$

b) axioms for bio-processes

1. $B_1 \parallel B_2 \equiv_b B_2 \parallel B_1$
2. $B_1 \parallel (B_2 \parallel B_3) \equiv_b (B_1 \parallel B_2) \parallel B_3$
3. $I[\,P\,] \equiv_b I[\,P'\,]$ if $P \equiv_p P'$

Table 3.2 – Structural congruence axioms.

is stored in three functions: α_b, α_u and α_c. $\alpha_b(T, U)$ gives the rate at which the interfaces associated with T and U can bind together; $\alpha_b(T, U) = 0$ means that interfaces associated with T and U cannot bind. Similarly these interfaces can unbind if $\alpha_u(S, T) > 0$. Communication is always allowed if $\alpha_c(S, T) > 0$ and the two interfaces involved are bound together. Otherwise, if the two interfaces are free, a communication can happen only if $\alpha_b(S, T) = 0$, $\alpha_u(S, T) = 0$ and $\alpha_c(S, T) > 0$.

We use events to substitute a box with two or more boxes. Their definition is based on a notion of structural congruence that makes it possible to identify bio-processes with the same behaviour, even if they are not syntactically equal. We define structural congruence for boxes and events, respectively, as follows:

Definition 3.1.2 *The structural congruence $\equiv_b$ among boxes is the smallest congruence relation that satisfies the rules of Table 3.2b and is based on the congruence $\equiv_p$ over processes, that, in turn, is defined as the smallest congruence relation that satisfies the rules of Table 3.2a*[1]*.*

Definition 3.1.3 *We define $I_1[\,P_1\,] \blacktriangleright_{r_1} B_1 \equiv_e I_2[\,P_2\,] \blacktriangleright_{r_2} B_2$ iff $I_1[\,P_1\,] \equiv_b I_2[\,P_2\,] \wedge B_1 \equiv_b B_2 \wedge r_1 = r_2$*

An event $I'[\,P'\,] \blacktriangleright_r B'$ describes the possibility of modifying a bio-process by substituting the occurrence of a box congruent to $I'[\,P'\,]$ with the bio-process B'. Note that with the side condition $B' = I[\,P\,] \parallel B$ we constrain B' to be composed at least of two boxes. The boxes involved in an event must not participate in the structure of any complex, in other words, all of their interfaces must be unbound.

Finally, a BlenX system. It is a triple (B, E, ξ) where B is a *bio-process*, E an event set and ξ an environment.

We will now introduce the concepts and the notations necessary for describing the operational semantics of the language.

Definition 3.1.4 *A substitution is a function on names that is the identity except on a finite set.*

[1] Remember that if $(P, |, \mathsf{nil})$ is a commutative monoid under $\equiv_p$, it means that for all $P, P_1, P_2, P_3 \in \mathcal{P}$, $P \,|\, \mathsf{nil} \equiv_p P$, $P_1 \,|\, P_2 \equiv_p P_2 \,|\, P_1$ and $(P_1 \,|\, P_2) \,|\, P_3 \equiv_p P_1 \,|\, (P_2 \,|\, P_3)$.

Notation 3.1.5 *We use σ to range over substitution, and write $y\sigma$ for σ applied to y. The* support *of σ, $supp(\sigma)$, is $\{y \mid y\sigma \neq y\}$ and the* co-support *of σ, $cosupp(\sigma)$, is $\{y\sigma \mid y \in supp(\sigma)\}$. We write $\{^{m_1,\ldots,m_n}/_{y_1,\ldots,y_n}\}$ for the substitution of σ such that $y_i\sigma = m_i$ for each $i \in [1,n]$ and $y\sigma = y$ for $y \notin \{y_1,\ldots,y_n\}$.*

We write $P\sigma$ for the capture-avoiding replacement of each free occurrence of each name y in P by $y\sigma$. The functions of Figure 3.3 formally give the definition of *free* and *bound* names that, for example, in a bio-process B are denoted as $\mathsf{fn}(B)$ and $\mathsf{bn}(B)$, respectively. We can lift the function that computes the free names of bio-processes in order to work with systems. We define $\mathsf{fn}(S)$ as the union of the free names that appear in B and in the bio-processes of the events of E. Formally for $S = (B, E, \xi)$ the definition is:

$$\mathsf{fn}(S) = \mathsf{fn}(B) \cup \bigcup_{I[\,P\,] \blacktriangleright_r B' \in E} \mathsf{fn}(B')$$

We can also substitute elements inside an environment writing $\xi\{^{(\gamma_1, T_1)}/_{(\gamma_2, T_2)}\}$ to mean that we replace all the occurrences of the pair (γ_2, T_2) with the pair (γ_1, T_1) inside the environment ξ.

Related with the concept of substitution is the notion of an α-equivalence relation which we formally define as follows.

Definition 3.1.6 *α-equivalence is the smallest equivalence relation that satisfies the following rules:*

$$b \notin \mathsf{fn}(P_1) \cup \mathsf{sub}(I) \Rightarrow \{(a,\, T)^r\} \cup I[\,P_1\,] = \{(b,\, T)^r\} \cup I[\,P_1\{^b/_a\}\,]$$

Note that this is not a congruence, since we strictly identify α-equivalent processes and bio-processes (as in [107]).

The semantics of BlenX is defined on boxes labelled according to their position inside the bio-process they belong to. The logic used for the labelling is the one sketched in Figure 3.2. Formally we perform such an operation through a labelling function $\mathcal{L}$ that, through the call $\mathcal{L}(\varepsilon, B)$, explores the bio-process B labelling its leaves, i.e. the boxes, with their position in the abstract syntax tree:

$$\begin{array}{rcl} \mathcal{L}(\gamma, I[\,P\,]) & = & I[\,P\,]_\gamma \\ \mathcal{L}(\gamma, B_1 \parallel B_2) & = & \mathcal{L}(\gamma\|_0, B_1) \parallel \mathcal{L}(\gamma\|_1, B_2) \end{array}$$

Labelled bio-processes have the same metavariables as normal bio-processes to keep the semantics more readable. We introduced such a labelling because, when a binding between two boxes happens, the semantics modifies the system by adding a link between the involved boxes. This link is recorded by enriching the environment with a new element that, among other information, stores the position of the boxes involved with respect to the bio-process they belong to. Given that, locally, the semantics cannot compute the positions of the two boxes we store this information on each box with the labels.

The semantics of BlenX generates two kind of transitions: *box transitions* and *system transitions*.

Box transitions are defined by the rules of Table 3.4 and have the form

$$I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma$$

$\mathsf{fn}(I[\,P\,])$	$=$	$\mathsf{fn}(P) \setminus \mathsf{sub}(I)$	$\mathsf{bn}(I[\,P\,])$	$=$	$\mathsf{bn}(P) \cup \mathsf{sub}(I)$
$\mathsf{fn}(\mathsf{nil})$	$=$	$\emptyset$	$\mathsf{bn}(\mathsf{nil})$	$=$	$\emptyset$
$\mathsf{fn}(B_0 \parallel B_1)$	$=$	$\mathsf{fn}(B_0) \cup \mathsf{fn}(B_1)$	$\mathsf{bn}(B_0 \parallel B_1)$	$=$	$\mathsf{bn}(B_0) \cup \mathsf{bn}(B_1)$
$\mathsf{fn}(P_0 \mid P_1)$	$=$	$\mathsf{fn}(P_0) \cup \mathsf{fn}(P_1)$	$\mathsf{bn}(P_0 \mid P_1)$	$=$	$\mathsf{bn}(P_0) \cup \mathsf{bn}(P_1)$
$\mathsf{fn}(M_0 + M_1)$	$=$	$\mathsf{fn}(M_0) \cup \mathsf{fn}(M_1)$	$\mathsf{bn}(M_0 + M_1)$	$=$	$\mathsf{bn}(M_0) \cup \mathsf{bn}(M_1)$
$\mathsf{fn}(\langle C \rangle M)$	$=$	$\mathsf{fn}(M) \cup \mathsf{fn}(C)$	$\mathsf{bn}(\langle C \rangle M)$	$=$	$\mathsf{bn}(M)$
$\mathsf{fn}(*\pi.P)$	$=$	$\mathsf{fn}(\pi.P)$	$\mathsf{bn}(*\pi.P)$	$=$	$\mathsf{bn}(\pi.P)$
$\mathsf{fn}(\pi.P)$	$=$	$\mathsf{fn}(\pi) \cup (\mathsf{fn}(P) \setminus \mathsf{bn}(\pi))$	$\mathsf{bn}(\pi.P)$	$=$	$\mathsf{bn}(\pi) \cup \mathsf{bn}(P)$
$\mathsf{fn}(n?y)$	$=$	$\{n\}$	$\mathsf{bn}(n?y)$	$=$	$\{y\}$
$\mathsf{fn}(n!m)$	$=$	$\{n, m\}$	$\mathsf{bn}(n!m)$	$=$	$\emptyset$
$\mathsf{fn}(\mathsf{ch}(r, a, T))$	$=$	$\{a\}$	$\mathsf{bn}(\mathsf{ch}(r, a, T))$	$=$	$\emptyset$
$\mathsf{fn}(\neg C)$	$=$	$\mathsf{fn}(C)$			
$\mathsf{fn}(C_1 \wedge C_2)$	$=$	$\mathsf{fn}(C_1) \cup \mathsf{fn}(C_2)$			
$\mathsf{fn}((a, S))$	$=$	$\{a\}$			
$\mathsf{fn}((a, \odot))$	$=$	$\{a\}$			
$\mathsf{fn}((a, \otimes))$	$=$	$\{a\}$			
$\mathsf{fn}(\mathsf{true})$	$=$	$\emptyset$			

Table 3.3 – Definition of free and bound names for bio-processes.

where r is a rate, C is a condition, and a is an action of the form $k!t$, $k?t$, (T, U), and τ. We refer this set of actions as Act_i. $k!t$ means that we send the name t over k, $k?t$ means that we receive t via k (we will see shortly why we need to introduce these new metavariables k and t and what they stand for), (T, U) changes the sort of the interface associated with T to U, and τ represent an internal action.

System transitions are defined by the rules of Table 3.6 and Table 3.8. They can be a τ-transition $S \xrightarrow{\tau}_r S'$ or have the form $S \xrightarrow{(\gamma, a)}_r S'$ where r is a rate, γ is the label of the box that performs the action in S and a can be either $T!n$, $T?n$, or T. $T!n$ means that we send the free name n through the interface associated with T, $T?n$ means that we receive n through the interface associated with T, and T means that a bind or an unbind happened by means of the interface associated with T.

We now go through the rules of the operational semantics. Rule r1 describes the execution of an output prefix. The label stores the channel or the interface name used for communicating and the name that is sent. It is important to highlight that for referring to an interface, within a label, we do not use its subject but we use its sort. This choice has been made because we want a bijection between derivation trees and the transitions they infer. If we were to use subjects to refer to interfaces, because of the identification of α-equivalent systems, we would be able to derive the same transition with infinitely many different derivation trees. Hence, there would not be a bijection between transitions and derivations trees.

$$\text{r1}\ \frac{}{I[\,n!m.\,P\,]_\gamma \xrightarrow{\mathsf{true},k!t}_0 I[\,P\,]_\gamma}\quad \text{provided: } ((k = n \wedge n \notin I) \vee (n,\,k)^r \in I) \wedge ((t = m \wedge m \notin I) \vee (m,\,t)^r \in I)$$

$$\text{r2}\ \frac{}{I[\,n?y.\,P\,]_\gamma \xrightarrow{\mathsf{true},k?t}_0 I[\,P\{^m/_y\}\,]_\gamma}\quad \text{provided: } ((k = n \wedge n \notin I) \vee (n,\,k)^r \in I) \wedge ((t = m \wedge m \notin I) \vee (m,\,t)^r \in I)$$

$$\text{r3}\ \frac{}{\{(a,\,U)^{r_1}\} \cup I[\,\mathsf{ch}(r,a,T).\,P\,]_\gamma \xrightarrow{\mathsf{true},(U,T)}_r \{(a,\,T)^{r_1}\} \cup I[\,P\,]_\gamma}\quad \text{provided: } T \notin \mathsf{sorts}(I)$$

$$\text{r4}\ \frac{I[\,\pi.\,P\,]_\gamma \xrightarrow{\mathsf{true},a}_r I'[\,P'\,]_\gamma}{I[\,*\pi.\,P\,]_\gamma \xrightarrow{\mathsf{true},a}_r I'[\,*\pi.\,P \mid P'\,]_\gamma}$$

$$\text{r5-l}\ \frac{I[\,P_1\,]_\gamma \xrightarrow{C_1,k!t}_0 I'[\,P_1'\,]_\gamma \qquad I[\,P_2\,]_\gamma \xrightarrow{C_2,k?t}_0 I'[\,P_2'\,]_\gamma}{I[\,P_1 \mid P_2\,]_\gamma \xrightarrow{C_1 \wedge C_2,\tau}_r I'[\,P_1' \mid P_2'\,]_\gamma}\quad \text{provided: } (((a,\,k)^r \in I) \vee (k \notin \mathsf{Sort} \wedge \delta(k) = r))$$

$$\text{r5-r}\ \frac{I[\,P_1\,]_\gamma \xrightarrow{C_1,k?t}_0 I'[\,P_1'\,]_\gamma \qquad I[\,P_2\,]_\gamma \xrightarrow{C_2,k!t}_0 I'[\,P_2'\,]_\gamma}{I[\,P_1 \mid P_2\,]_\gamma \xrightarrow{C_1 \wedge C_2,\tau}_r I'[\,P_1' \mid P_2'\,]_\gamma}\quad \text{provided: } (((a,\,k)^r \in I) \vee (k \notin \mathsf{Sort} \wedge \delta(k) = r))$$

$$\text{r6}\ \frac{I[\,M\,]_\gamma \xrightarrow{C',a}_r I'[\,M'\,]_\gamma}{I[\,\langle C\rangle M\,]_\gamma \xrightarrow{\wr C \wr_I^* \wedge C',a}_r I'[\,M'\,]_\gamma}$$

$$\text{r7-l}\ \frac{I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma}{I[\,P \mid Q\,]_\gamma \xrightarrow{C,a}_r I'[\,P' \mid Q\,]_\gamma} \qquad \text{r7-r}\ \frac{I[\,Q\,]_\gamma \xrightarrow{C,a}_r I'[\,Q'\,]_\gamma}{I[\,P \mid Q\,]_\gamma \xrightarrow{C,a}_r I'[\,P \mid Q'\,]_\gamma}$$

$$\text{r8-l}\ \frac{I[\,M\,]_\gamma \xrightarrow{C,a}_r I'[\,M'\,]_\gamma}{I[\,M + N\,]_\gamma \xrightarrow{C,a}_r I'[\,M'\,]_\gamma} \qquad \text{r8-r}\ \frac{I[\,N\,]_\gamma \xrightarrow{C,a}_r I'[\,N'\,]_\gamma}{I[\,M + N\,]_\gamma \xrightarrow{C,a}_r I'[\,N'\,]_\gamma}$$

Table 3.4 – Operational semantics for boxes.

For making this point clear, we propose an example. Consider the following two derivation trees where, within the labels, we store the subjects of the interfaces instead of their sorts:

$$\text{r1}\ \frac{}{(a,\,T)^r[\,a!y.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{a!y}_0 (a,\,T)^r[\,\mathsf{nil}\,]_\varepsilon} \qquad \text{r1}\ \frac{}{(b,\,T)^r[\,b!y.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{b!y}_0 (b,\,T)^r[\,\mathsf{nil}\,]_\varepsilon}$$

The boxes involved in the two transitions are exactly the same. Also the actions that cause the transitions are exactly the same because of the definition of α-equivalence. However the derivation trees are different; in fact the label $a!y$ is different from the label $b!y$ breaking the bijection constraint: we have *two* derivation trees to infer the same transition. Instead storing sorts we obtain we obtain the same derivation tree in both cases:

$$\text{r1}\ \frac{}{(a,\,T)^r[\,a!y.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{T!y}_0 (a,\,T)^r[\,\mathsf{nil}\,]_\varepsilon} \qquad \text{r1}\ \frac{}{(b,\,T)^r[\,b!y.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{T!y}_0 (b,\,T)^r[\,\mathsf{nil}\,]_\varepsilon}$$

$$
\begin{array}{rcl}
\wr \mathsf{true} \int_I^* & = & \mathsf{true} \\
\wr (a,T) \int_I^* & = & \begin{cases} \mathsf{true} & \text{if } (a,T) \in I \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\wr (a,\odot) \int_I^* & = & \begin{cases} (T,\odot) & \text{if } (a,T) \in \mathsf{sub}(I) \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\wr (a,\otimes) \int_I^* & = & \begin{cases} (T,\otimes) & \text{if } (a,T) \in \mathsf{sub}(I) \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\wr C_1 \wedge C_2 \int_I^* & = & \wr C_1 \int_I^* \wedge \wr C_2 \int_I^* \\
\wr \neg C \int_I^* & = & \neg \wr C \int_I^*
\end{array}
$$

Table 3.5 – Partial evaluation of a condition.

Rule r2 describes the execution of an input prefix and is similar to rule r1.

Rule r3 describes how the sort of an interface can be changed through an action $\mathsf{ch}(r,a,T)$. The transition is allowed only if the sort T is not associated with another interface in I. This check is important for maintaining the well-formedness of the box that performs the action (a box cannot have two interfaces associated with the same sort). The label stores the pair of sorts that are involved in the change action, information that we need when we lift the transition to the system level.

Rule r4 describes how replication is unfolded. It states that a process guarded by an action under replication cannot be consumed. When the action π that guards the process P fires, instead of being consumed and causing the transition of the process in P', it is not affected, and the process P' is added in parallel. This mechanism gives the possibility to implement recursion, and thus describes a box with an internal process that never deadlocks. Here an example of how we can get this result:

$$
\mathrm{r12}\ \dfrac{\mathrm{r5\text{-}l}\ \dfrac{\mathrm{r4}\ \dfrac{\mathrm{r1}\ \dfrac{}{(a,\,T)^r[\,a!y.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{T!y}_0 (a,\,T)^r[\,\mathsf{nil}\,]_\varepsilon}}{(a,\,T)^r[\,*a!y.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{T!y}_0} \quad \mathrm{r4}\ \dfrac{\mathrm{r2}\ \dfrac{}{(a,\,T)^r[\,a?z.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{T?y}_0 (a,\,T)^r[\,\mathsf{nil}\,]_\varepsilon}}{(a,\,T)^r[\,*a?z.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{T?y}_0}}{(a,\,T)^r[\,*a!y.\,\mathsf{nil}\,|\,\mathsf{nil}\,]_\varepsilon \qquad (a,\,T)^r[\,*a?z.\,\mathsf{nil}\,|\,\mathsf{nil}\,]_\varepsilon}}{(a,\,T)^r[\,*a!y.\,\mathsf{nil}\,|\,*\,a?z.\,\mathsf{nil}\,]_\varepsilon \xrightarrow{\tau}_r (a,\,T)^r[\,*a!y.\,\mathsf{nil}\,|\,\mathsf{nil}\,|\,*\,a?z.\,\mathsf{nil}\,|\,\mathsf{nil}\,]_\varepsilon}}{((a,\,T)^r[\,*a!y.\,\mathsf{nil}\,|\,*\,a?z.\,\mathsf{nil}\,]_\varepsilon, \mathsf{Nil}, \emptyset) \xrightarrow{\tau}_r ((a,\,T)^r[\,*a!y.\,\mathsf{nil}\,|\,\mathsf{nil}\,|\,*\,a?z.\,\mathsf{nil}\,|\,\mathsf{nil}\,]_\varepsilon, \mathsf{Nil}, \emptyset)}
$$

In this example the transition is caused by two processes under replication. Given that they will never be consumed, the result is that the bio-process never deadlocks, and will keep performing the same action and accumulating nil processes.

Rules r5-l and r5-r describe the intra-communications capability of a box, i.e. the possibility to exchange information between two processes within the same box. The transition involves two processes with the capability of performing symmetric actions. By symmetric we mean an input and an output action which communicate on the same channel k (or the same interface associated with the sort k) and exchanges exactly the same information, i.e. the channel t (or the interface associated with the sort t). Note that if the communication happens over an interface, the rate r is stored in the definition of the interface. On the contrary, in the case of a free name, the rate is given by the function δ which associates a rate with each free name.

Rule r6 collects in the label the partial evaluation $\wr C \int_I^*$ of the condition in front of a capability M. $\wr C \int_I^*$, whose definition is in Table 3.5, evaluates any atomic condition in C that

$$\text{r9}\ \frac{I[\,P\,]_\gamma \xrightarrow{C,T!n}_0 I[\,P'\,]_\gamma}{(I[\,P\,]_\gamma, E, \xi) \xrightarrow{(\gamma,T!n)}_0 (I[\,P'\,]_\gamma, E, \xi)}\ \text{provided } \wr C \int_{\gamma,\xi} = \mathsf{true} \qquad \text{r10}\ \frac{I[\,P\,]_\gamma \xrightarrow{C,T?n}_0 I[\,P'\,]_\gamma}{(I[\,P\,]_\gamma, E, \xi) \xrightarrow{(\gamma,T?n)}_0 (I[\,P'\,]_\gamma, E, \xi)}\ \text{provided } \wr C \int_{\gamma,\xi} = \mathsf{true}$$

$$\text{r11}\ \frac{I[\,P\,]_\gamma \xrightarrow{C,(T,U)}_r I'[\,P'\,]_\gamma}{(I[\,P\,]_\gamma, E, \xi) \xrightarrow{\tau}_r (I'[\,P'\,]_\gamma, E, \xi\{^{(\gamma,U)}/_{(\gamma,T)}\})}\ \text{provided } \wr C \int_{\gamma,\xi} = \mathsf{true} \qquad \text{r12}\ \frac{I[\,P\,]_\gamma \xrightarrow{C,\tau}_r I[\,P'\,]_\gamma}{(I[\,P\,]_\gamma, E, \xi) \xrightarrow{\tau}_r (I[\,P'\,]_\gamma, E, \xi)}\ \text{provided } \wr C \int_{\gamma,\xi} = \mathsf{true}$$

Table 3.6 – Operational semantics, from boxes to systems.

$$\begin{array}{rcl}
\wr \mathsf{true} \int_{\gamma,\xi} & = & \mathsf{true} \\
\wr (T, \odot) \int_{\gamma,\xi} & = & \begin{cases} \mathsf{true} & \text{if } \exists \{(\gamma, T), (\gamma', T')\} \in \xi \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\wr (T, \otimes) \int_{\gamma,\xi} & = & \begin{cases} \mathsf{true} & \text{if } \nexists \{(\gamma, T), (\gamma, T')\} \in \xi \\ \mathsf{false} & \text{otherwise} \end{cases} \\
\wr C_1 \wedge C_2 \int_{\gamma,\xi} & = & \wr C_1 \int_{\gamma,\xi} \wedge \wr\, C_2 \int_{\gamma,\xi} \\
\wr \neg C \int_{\gamma,\xi} & = & \neg \wr\, C \int_{\gamma,\xi}
\end{array}$$

Table 3.7 – Evaluation function.

can be solved by looking at the interface set I. In this way we store in the label a condition whose satisfaction depends only on the binding state of the interfaces. Note that, because of issues with α-equivalence, the conditions stored in the label refer to the interfaces using their sort instead of their subject. This condition is evaluated later on in the derivation tree — in particular as soon as the information regarding the links between the interfaces is available (i.e., when one of the rules of Table 3.6 is applied).

Rules r7-l, r7-r, r8-l, and r8-r, allow processes in a context to perform actions.

We will now describe the rules in Table 3.6. They lift the box-level transitions to the level of BlenX system only if the condition C which they are labelled with evaluates to true with respect to the environment ξ. Note that C is the result of the conjunction of *partially evaluated* conditions, and thus refers to interfaces using their sort and does not contain atomic condition that test the sort of an interface. This is the reason why the evaluation function defined in Table 3.7 does not consider the case $\wr (T, U) \int_{\gamma,\xi}$.

Rules r9, r10, r11 and r12 lift to the level of a system the transitions which involve boxes that output a channel name over an interface, input a channel name over an interface, change the sort associated with an interface, and perform an intra-communication, respectively. Given that the environment refers to interfaces using their sorts, in rule r11, where the sort of an interface changes to U, the substitution $\xi\{^{(\gamma,U)}/_{(\gamma,T)}\}$ is necessary to keep the information stored in the environment consistent.

We can now describe the group of rules in Table 3.8. Rule r13 simply states that one of the interfaces of the box γ is associated with sort T. This information is exploited by rule r14, which creates a link between two interfaces with compatible sorts if they are unbound and belong to different boxes. In a similar way, rule r15 destroys the link between two interfaces.

Rule r16-l and r16-r synchronize an inter-communication between two bio-processes. The

$$\text{r13}\ \frac{}{(\{(a,\,T)^r\} \cup I[\,P\,]_\gamma, E, \xi) \xrightarrow{(\gamma,T)}_0 (\{(a,\,T)^r\} \cup I[\,P\,]_\gamma, E, \xi)}$$

$$\text{r14}\ \frac{(B_1, E, \xi) \xrightarrow{(\gamma_1,T_1)}_0 (B_1, E, \xi) \qquad (B_2, E, \xi) \xrightarrow{(\gamma_2,T_2)}_0 (B_2, E, \xi)}{(B_1 \parallel B_2, E, \xi) \xrightarrow{\tau}_{\alpha_b(T_1,T_2)} (B_1 \parallel B_2, E, \xi \cup \{\{(\gamma_1,T_1),(\gamma_2,T_2)\}\})}$$
$$\text{provided: } \alpha_b(T_1,T_2) > 0 \wedge \forall L \in \xi : \{(\gamma_1,T_1),(\gamma_2,T_2)\} \cap L = \emptyset$$

$$\text{r15}\ \frac{(B_1, E, \xi) \xrightarrow{(\gamma_1,T_1)}_{r_1} (B_1, E, \xi) \qquad (B_2, E, \xi) \xrightarrow{(\gamma_2,T_2)}_{r_2} (B_2, E, \xi)}{(B_1 \parallel B_2, E, \xi \cup \{\{(\gamma_1,T_1),(\gamma_2,T_2)\}\}) \xrightarrow{\tau}_{\alpha_u(T_1,T_2)} (B_1 \parallel B_2, E, \xi)}$$
$$\text{provided: } \alpha_u(T_1,T_2) > 0$$

$$\text{r16-l}\ \frac{(B_1, E, \xi) \xrightarrow{(\gamma_1,T_1!n)}_0 (B_1', E, \xi) \qquad (B_2, E, \xi) \xrightarrow{(\gamma_2,T_2?n)}_0 (B_2', E, \xi)}{(B_1 \parallel B_2, E, \xi) \xrightarrow{\tau}_{\alpha_c(T_1,T_2)} (B_1' \parallel B_2', E, \xi)}$$
$$\text{provided: } \alpha_c(T_1,T_2) > 0 \wedge ((\{(\gamma_1,T_1),(\gamma_2,T_2)\} \in \xi) \vee (\alpha_b(T_1,T_2) = \alpha_u(T_1,T_2) = 0))$$

$$\text{r16-r}\ \frac{(B_1, E, \xi) \xrightarrow{(\gamma_1,T_1?n)}_0 (B_1', E, \xi) \qquad (B_2, E, \xi) \xrightarrow{(\gamma_2,T_2!n)}_0 (B_2', E, \xi)}{(B_1 \parallel B_2, E, \xi) \xrightarrow{\tau}_{\alpha_c(T_1,T_2)} (B_1' \parallel B_2', E, \xi)}$$
$$\text{provided: } \alpha_c(T_1,T_2) > 0 \wedge ((\{(\gamma_1,T_1),(\gamma_2,T_2)\} \in \xi) \vee (\alpha_b(T_1,T_2) = \alpha_u(T_1,T_2) = 0))$$

$$\text{r17}\ \frac{}{(I[\,P\,]_\gamma, E, \xi) \xrightarrow{\tau}_r (\mathcal{L}(\gamma, B), E, \xi)}$$
$$\text{provided: } I'[\,P'\,] \blacktriangleright_r B \in E,\ I'[\,P'\,] \equiv_b I[\,P\,] \wedge \nexists (a,\,T)^r \in I \wedge L \in \xi : \{(\gamma,T)\} \cap L \neq \emptyset$$

$$\text{r18-l}\ \frac{(B_1, E, \xi) \xrightarrow{a}_r (B_1', E, \xi')}{(B_1 \parallel B_2, E, \xi) \xrightarrow{a}_r (B_1' \parallel B_2, E, \xi')} \qquad \text{r18-r}\ \frac{(B_2, E, \xi) \xrightarrow{a}_r (B_2', E, \xi')}{(B_1 \parallel B_2, E, \xi) \xrightarrow{a}_r (B_1 \parallel B_2', E, \xi')}$$

Table 3.8 – Operational semantics for systems.

communication can happen only if the sorts of the involved interfaces have a non-zero communication rate ($\alpha_c(T_1,T_2) > 0$). Moreover one of the following must hold: the two interfaces must be linked together or their binding and unbinding capability must be equal to zero ($\alpha_b(T_1,T_2) = 0$ and $\alpha_u(T_1,T_2) = 0$).

Rule r17 allows the replacement of a box $I[\,P\,]_\gamma$ with a bio-process B. The Rule enables the replacement if the set of events E contains an event whose definition is $I'[\,P'\,] \blacktriangleright_r B$ with $I'[\,P'\,] \equiv_b I[\,P\,]$, and if the interfaces of the replaced box γ are not involved in any link. The rule applies the correct labels to the boxes of the bio-process B through the function $\mathcal{L}$.

Finally the rules r18-l and r18-r describes the evolution of bio-processes in a context.

3.2 Well-formed Systems

In order to work consistently with the operational semantics, a system must be well-formed. In the following we list the definitions of well-formedness for systems and for its components (with the exception of the notion of well-formed bio-processes that we introduced earlier for presentation constraint).

Definition 3.2.1 *A pair (B, ξ), where B is a bio-process and ξ is an environment, is well-formed if:*

1. *B is well formed.*
2. *All the box labels used in ξ are valid pointers to boxes of B.*
3. *For all $\{(\gamma_1, T_1), (\gamma_2, T_2)\} \in \xi, \gamma_1 \neq \gamma_2$, the box pointed by γ_1 has an interface associated with T_1, and the box pointed by γ_2 has an interface associated with T_2.*
4. *For all $L_1 \neq L_2 \in \xi, L_1 \cap L_2 = \emptyset$ (i.e. each interface can be involved in no more than one element of ξ).*

Definition 3.2.2 *A set of events E is well-formed if each element $I[\,P\,] \blacktriangleright_r B \in E$ belongs to a distinct congruence class and $I[\,P\,]$ and B are well-formed.*

Definition 3.2.3 *A system (B, E, ξ) is well-formed if the pair (B, ξ) and the event set E are well-formed.*

The definition of well-formedness for bio-processes and the pairs made of a bio-process and an environment is intuitive. That is not the same for the notion of well-formedness for set of events. We introduced it because a system with a generic set of events could compromise the existence of a bijection between transitions and their derivation trees. Let us consider the example $E = \{I_1[\,P_1\,] \blacktriangleright_r B, I_2[\,P_2\,] \blacktriangleright_r B\}$ with $I_1[\,P_1\,] \blacktriangleright_r B \equiv_e I_2[\,P_2\,] \blacktriangleright_r B$ that implies $I_1[\,P_1\,] \equiv_b I_2[\,P_2\,]$. In this case the system $(I_1[\,P_1\,]_\varepsilon, E, \emptyset)$ can perform two different transitions given that both the events of E can be applied to $I_1[\,P_1\,]_\varepsilon$. However we obtain the two transitions with the same derivation tree:

$$\text{r17}\ \frac{}{(I_1[\,P_1\,]_\gamma, E, \xi) \xrightarrow{\tau}_r (\mathcal{L}(\gamma, B), E, \xi)}$$

This eventuality cannot happen in the case of a well-formed set of events and hereafter we will consider only systems belonging to the set of well-formed systems $\mathcal{S}$.

By means of the semantics BlenX can be defined as a *multi-transition system* (in the style or PEPA [50]). In general a multi-transition system is defined as a pair made of a set of state Q and a transition *multi-relation* defined as a function $Q \times Q \mapsto n$. Thus BlenX may be regarded as a multi-transition system defined as $(\mathcal{S}, \rightarrow)$ where the multi-relation $\rightarrow$ is a function $\mathcal{S} \times r \times \mathcal{S} \mapsto \mathbb{N}$ with the following definition: $\rightarrow (S_1, r, S_2) = n$ if $S_1 \xrightarrow{\tau}_r S_2$ with $n > 0$ distinct derivation trees and $\rightarrow (S_1, r, S_2) = \perp$ if there are not derivation trees for the transition $S_1 \xrightarrow{\tau}_r S_2$. If $\rightarrow (S_1, r, S_2) = n$ we write $S_1 \rightarrow_r S_2$ with multiplicity n. If we are not interested in the multiplicity we simply write $S_1 \rightarrow_r S_2$.

3.3 Adding Priorities

Since now we associated BlenX actions with rates that belongs to $\mathbb{R}_0^+$. Here we introduce a typology of actions named *immediate actions*. In contrast with normal actions, that now we call *actions with finite rate*, they are associated with an *infinite rate* (denoted with ∞). This actions have high priority which means that a system can perform a transition with finite rate (i.e. a transition triggered by an action with finite rate) only if it cannot perform an *immediate transition* (i.e. a transition triggered by an immediate action).

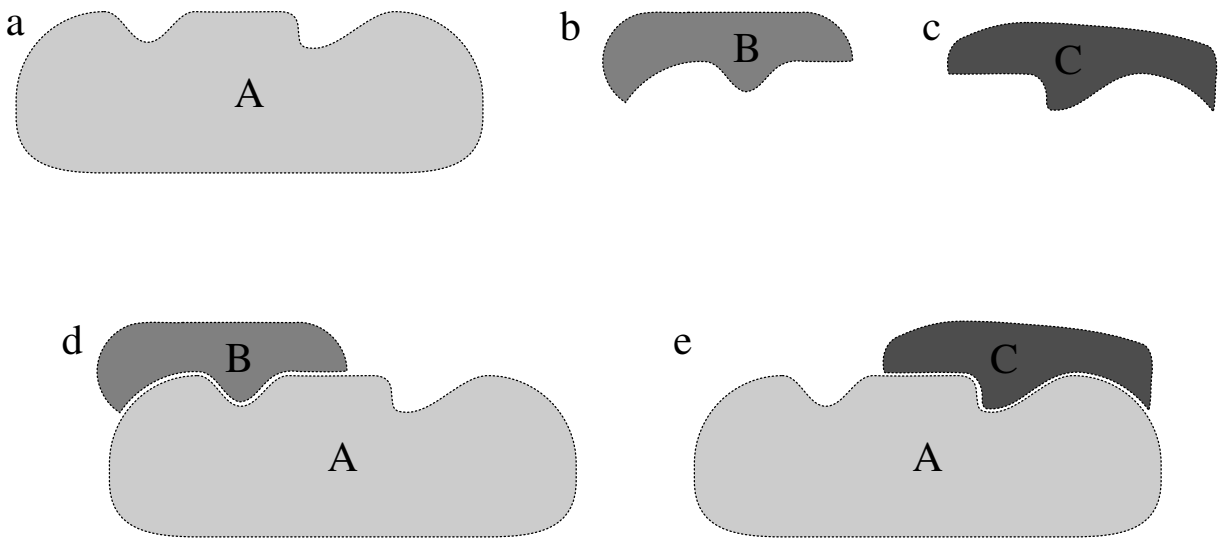


Figure 3.3 – Example of concurrent bindings.

The priority mechanism allows to define sequence of actions that are atomically executed (i.e. cannot be interrupted). We now propose an example (see Figure 3.3) for justifying the need of this mechanism with two priority levels from a biological point of view. In the example there are three kind of proteins. Protein `A` has two binding sites represented by the notches on its surface. The domain on the left can interact with the protein `B`, the one on the right with the protein `C`. This interactions lead to the formation of the complexes represented in Figure 3.3d and 3.3e, respectively. Note that the trimer made of the proteins `A`, `B` and `C` cannot form. In fact the proteins `B` and `C` are too big to be simultaneously bound to A. In the following we define a program describing such a scenario.

As shown in Figures 3.4b and 3.4c, we encode the proteins `B` and `C` as two boxes provided with one interface (they only have one binding site) and with an empty internal process (i.e. nil). The box encoding the protein `A` (represented in Figure 3.4a), has two interfaces which are named *left* and *right* and are associated with the sorts L and R, respectively. In order to allow the protein `A` to form a complex with the boxes that encode the proteins `B` and `C`, we define the binding capability function as follows: $\alpha_b = \alpha_0[(BS, L) \mapsto r_1][(CS, R) \mapsto r_2]$. Where α_0 is the function that maps every pair of sorts to zero, and r_1 and r_2 are the rates of binding between `B` and `A`, and `C` and `A`, respectively.

The internal process of the box encoding the protein `A` is defined in such a way that when an interface binds, the sort associated with the other one is changed. In this way we want to prevent that the boxes `B` and `C` can be bound to `A` at the same time:

$$P = \langle (\mathit{left}, \otimes) \rangle \mathsf{ch}(r_3, \mathit{right}, \mathit{RI}).\,\mathsf{nil} + \langle (\mathit{right}, \otimes) \rangle \mathsf{ch}(r_4, \mathit{left}, \mathit{LI}).\,\mathsf{nil}$$

Process P is made of two alternative sub-processes. The left one is guarded by the atomic condition $\langle (\mathit{left}, \otimes) \rangle$ that activates the process $\mathsf{ch}(r_3, \mathit{right}, \mathit{RI}).\,\mathsf{nil}$ once the *left* interface binds. This process associates the sort RI with the interface *right*. In this way, once a binding on the *left* interface occurs, it prevents a future binding of the box `A` with the box `C`, indeed the pair of sorts $(\mathit{RI}, \mathit{CS})$ does not have positive binding capability.

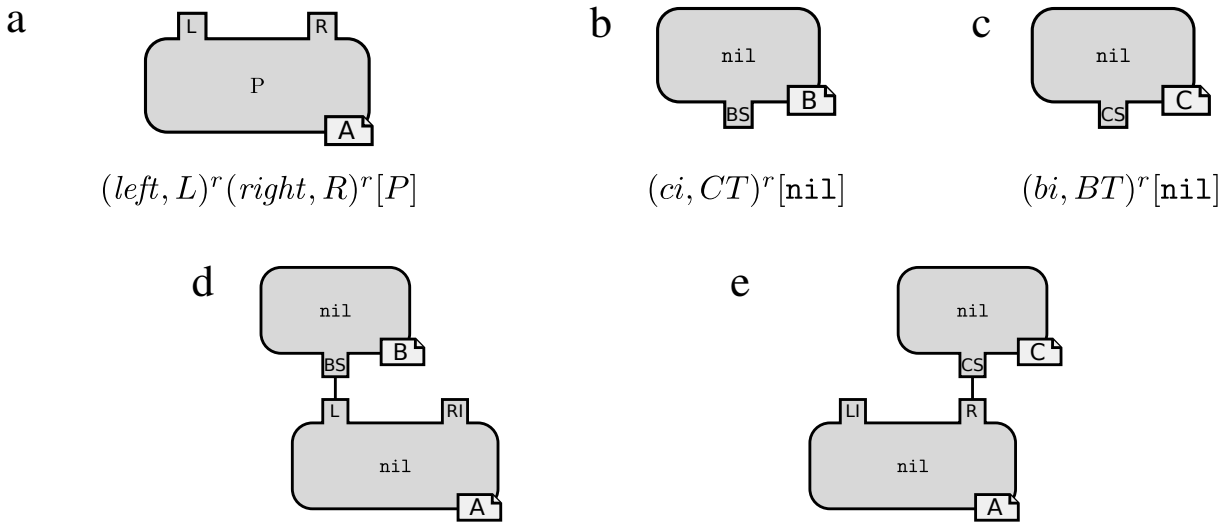


Figure 3.4 – BlenX encoding of the proteins represented in Figure 3.3. In the graphical representation of the boxes, we omit to write the name of the interfaces, we only specify their sort. We also omit to represent the internal process when it is too large or when it is not important for the explanation.

The right alternative process behaves as the left one but the roles of the interfaces are swapped. If the *right* interface binds than the binding capability of the *left* interface is inhibited.

At this point our attention focuses on rates r_3 and r_4. Let us associate a positive real value with these rates and look in Figure 3.5 how the represented system evolves respect to this choice. In the first state S1, two events can happen: `B` binds to `A` or `C` binds to `A`. In the states S2 and S3 we would like that the only enabled action is the one which disables the binding on the free interface and makes the system move in the state S4 and S5, respectively. However, these actions are concurrently enabled with the binding actions that lead to the formation of the undesired trimer of states S6. After the formation of the trimer both the alternative processes of the internal process P are enabled. Depending on which one executes before, the state of the system becomes S7 or S8. In order to prevent transitions that lead to state S6, we have to associate the change actions that cause the transition from S2 to S4 and from S3 to S5 with an infinite rate. In this way the transitions forming the trimer are disabled because concurrent with immediate transitions having higher priority.

Now the question is, why we need actions associated with infinite rate? Nature can make very fast reactions but not instantaneous reactions. How can we justify immediate actions from a biological point of view? The answer is that, with our language, we are not encoding nature but a small subset of it. For instance the language does not have primitives for encoding *space constraints*. We cannot neatly express that proteins `B` and `C` are too large to simultaneously bind to `A`. We have to use a trick to do that, and the trick consists in using immediate actions. In this way the language can encode things it is actually missing the primitive for.

We now give a formal definition of BlenX which takes into account the priorities mechanism. Extending the definition of section 3.2 we say that BlenX can be regarded as a *multi-transition system* $(S, \rightarrow)$, where S is the set of all the well-formed BlenX systems, and the multi relation $\rightarrow$ is defined as the function with the largest domain such that it holds that:

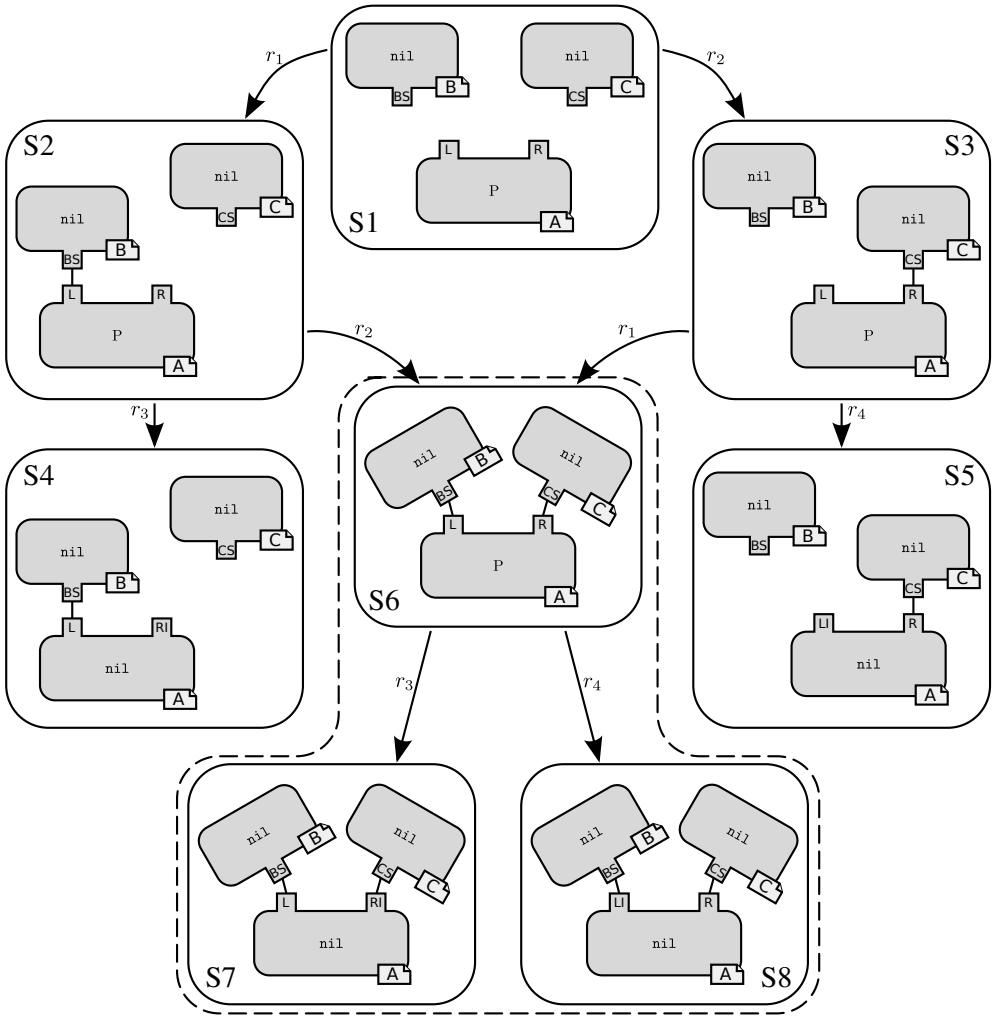


Figure 3.5 – Evolution of a system with concurrent bindings.

1. If $\rightarrow(S_1, r, S_2) = n$ then $S_1 \xrightarrow{\tau}_r S_2$ with $n > 0$ distinct derivation trees.
2. If $\rightarrow(S_1, r, S_2) = \perp$ then there are not derivation trees for $S_1 \xrightarrow{\tau}_r S_2$.
3. If $S_1 \xrightarrow{\tau}_\infty S_2$ then $\nexists S_3 : \rightarrow(S_1, r, S_3) \neq \perp$ with $r \neq \infty$.

If $\rightarrow(S_1, r, S_2) = n$ we write $S_1 \rightarrow_r S_2$ with multiplicity n. If we are not interested in the multiplicity we simply write $S_1 \rightarrow_r S_2$.

In Figure 3.5 we informally introduced the representation of an object effectively describing the behaviour of a system by listing the transitions that a system can perform. Given a system well-formed system S we call this object the multi-transition system generated by the system S, and here we introduce some notions necessary to give its formal definition.

Definition 3.3.1 *If $S_1 \rightarrow_r S_2$, then S_2 is a (one-step) derivative of S_1. More generally if $S_1 \rightarrow_{r_1} \ldots \rightarrow_{r_{n-1}} S_n$, then S_n is a derivative of S_1. We write $ds(S)$ for the set which contains S and its derivatives.*

Definition 3.3.2 *The multi-transition system generated by a system S and written as Φ^S is defined as the pair $(ds(S), \rightarrow^S)$ where the multi-relation $\rightarrow^S$ is defined as the largest subset of $\rightarrow$ such that it holds that if $\rightarrow^S(S_1, r, S_2) \neq \perp$ then $S_1, S_2 \in ds(S)$.*

As for the other multi-relations we introduce the compact notation $S_1 \rightarrow_r^S S_2$.

3.3.1 The Underlying Stochastic Process

In this section we use Φ^S to define the underlying stochastic process of S. We highlight the relationship of this process with those generated by GSNPs (General Stochastic Petri Nets) which implies it can be reduced to a CTMC (under some conditions) [56].

For the sake of clearness we introduce an equivalent representation of Φ^S which better fits our needs. It is a transition system whose state space is made of equivalence classes of labelled BlenX systems. In the following definition we introduce the function *get*, which is necessary to give the notion of structural equivalence between systems.

Definition 3.3.3

$$get(\gamma, S) = \begin{cases} I[\,P\,]_\gamma & \text{if } I[\,P\,] \text{ appears labelled with } \gamma \text{ in } S \\ \perp & \text{otherwise} \end{cases}$$

Definition 3.3.4 (Structural equivalence) *The structural equivalence $\equiv$ among systems is the smallest equivalence relation that satisfies the following:*
$S = (B, E, \xi) \equiv (B', E, \xi') = S'$ if one of the following holds

1. $\forall \gamma \in Label$ *s.t.* $get(S, \gamma) = I[\,P\,]_\gamma$ *and* $get(S', \gamma) = I'[\,P'\,]_\gamma$, $I[\,P\,] \equiv_b I'[\,P'\,]$ *and* $\xi = \xi'$
2. $\forall \gamma \in Label \setminus \{\gamma_1, \gamma_2\}, get(S, \gamma) = get(S', \gamma)$, $get(S, \gamma_1) = I[\,P_1\,]_{\gamma_1}, get(S, \gamma_2) = I[\,P_2\,]_{\gamma_2}$, $get(S', \gamma_1) = I[\,P_2\,]_{\gamma_1}, get(S', \gamma_2) = I[\,P_1\,]_{\gamma_2}$ *and* ξ' *results by replacing* γ_1 *with* γ_2 *and* γ_2 *with* γ_1 *into* ξ

The first condition of Definition 3.3.4 states that two systems are congruent if all of their boxes with the same label are congruent. The second says that two systems are congruent if the second can be obtained from the first by swapping two boxes.

Notation 3.3.5 *Given a well-formed system S, we refer its equivalence class as $[\![S]\!]$.*

The state space of the transition system generated by S_o is the set A where $[\![S]\!] \in A$ iff $\exists S' \in [\![S]\!]$ s.t. $S' \in ds(S_o)$. Its relation, which we refer as $\underset{\equiv}{\rightarrow}^{S_o}$ and is a subset of $A \times \mathbb{R} \cup (\{\infty\} \times \mathbb{R}) \times A$, is based on the definition of $\rightarrow_{S_o}$:

- $([\![S''']\!], r, [\![S']\!]) \in \underset{\equiv}{\rightarrow}^{S_o}$ iff
  1. $\exists S \equiv S'''$ s.t. $S \rightarrow_{r_1}^{S_o} S'_1, \ldots, S \rightarrow_{r_n}^{S_o} S'_n$ with multiplicity $m_1, \ldots, m_n$, respectively and $\{S'_1, \ldots, S'_n\} \in [\![S']\!]$
  2. $\sum_{i=0}^{n} m_i r_i = r$

3. $\nexists S''$ s.t. $S'' \in [\![S']\!]$ and $S'' \notin \{S'_1, \ldots, S'_n\}$ and $S \rightarrow_{r'}^{S_o} S''$

- $([\![S''']\!], (\infty, p), [\![S']\!]) \in \underset{\equiv}{\rightarrow}^{S_o}$ iff

1. $\exists S \equiv S'''$ s.t.
   (a) $S \rightarrow_{\infty}^{S_o} S'_1, \ldots, S \rightarrow_{\infty}^{S_o} S'_{n'}$ with multiplicity $m'_1, \ldots, m'_{n'}$, respectively and it holds that $\{S'_1, \ldots, S'_n\} \in [\![S']\!]$
   (b) $S \rightarrow_{\infty}^{S_o} S''_1, \ldots, S \rightarrow_{\infty}^{S_o} S''_{n''}$ with multiplicity $m''_1, \ldots, m''_{n''}$, respectively and it holds that $S''_1, \ldots, S''_n \notin [\![S']\!]$
2. $\nexists S''$ s.t. $S'' \in [\![S']\!]$ and $S'' \notin \{S'_1, \ldots, S'_n, S''_1, \ldots, S''_n\}$ and $S \rightarrow_{\infty}^{S_o} S''$
3. $p = \frac{\sum_{i=0}^{n'} m'_i}{\sum_{i=0}^{n'} m'_i + \sum_{i=0}^{n''} m''_i}$

Definition 3.3.6 *The transition system generated by S_o is defined as the pair $(A, \underset{\equiv}{\rightarrow}^{S_o})$ where $S \in A$ iff $\exists S' \in [\![S]\!]$ s.t. $S' \in ds(S_o)$. If $([\![S]\!], r, [\![S']\!] \in \underset{\equiv}{\rightarrow}^{S_o})$ we write $[\![S]\!] \underset{\equiv}{\rightarrow}_r^{S_o} [\![S']\!]$.*

Here we highlight that this transition system can be interpreted as the definition of a semi-Markov stochastic process. In particular it is a semi-Markov stochastic process of the kind generated by GSPNs [56] where the states are partitioned in vanishing (which perform immediate actions) and tangible (which do not). As explained in section 2.4.2, if the initial state of this stochastic process is tangible then it is associated to a CTMC.

In general in BlenX, both the transition system and its underlying CTMC, cannot be computed because they are huge or even infinite. What is done in practice is to perform simulations. At the beginning of the simulation the *simulation time* is set to zero and the system is in the initial state. The first step consists in the computation via the operational semantics of the actions which can be triggered from the initial state. If the state is tangible each action is associated to a rate which is the parameter of a negative exponential distribution. These distributions are used to stochastically determine the action to perform. The simulation trigger the selected action which causes the update of the simulation time and the movement of the system in the state computed by applying, via the operational semantics, the selected action to the initial state. In the case of a vanishing initial state each action is associated with a pair (∞, p), where p is a probability and is used to randomly decide which, among the concurrent immediate actions, is triggered to move in the next state. When the system move from a vanishing state the simulation time remains unchanged. Once in the new state, these steps are repeated until the simulation is interrupted or the system reaches a deadlock, i.e. a state which cannot perform any action.

If we think about this process in terms of transition system, the simulation is a walk among its states which is driven by the rates and the probabilities associated with arcs. We can further observe that considering the interpretation we give to immediate actions, the vanishing states visited by the system are not of any interest. Therefore a finer interpretation we can give to the simulation is a walk among the states of the CTMC associated with the stochastic process (i.e. the tangible state of the stochastic process), where each step of the walk is driven by the rates associated with the arcs of the CTMC.

Considering that we are interested only in the movement among tangible states introduces the possibility to define a quicker simulation algorithm. In particular we can make the simulation move from tangible to tangible states, without traversing all the intermediate vanishing states.

This means the simulation goes through the states of the underlying CTMC rather then the states of the transition system. To define a method which makes the simulation exclusively visit tangible states is extremely challenging thus, in practice, it can be defined a hybrid solution: in some cases the simulation jumps from tangible to tangible states, in others instead, the vanishing states are traversed. So doing the simulation moves on a reduced version of the transition system which is not the same we started with but which has the same underlying CTMC.

3.4 Using the Language: Self-Assembly

Modelling biological systems requires us to deal with a large variety of challenges. Since it is not possible to tackle all problems, we typically design a language that focuses on developing innovative solution to specific problems. The main purpose of BlenX is to allow us to describe in an intuitive way the interaction and the formation of complexes between proteins. Thanks to this capability, the language is particularly effective in the description of self-assembly processes. In this section we will illustrate the advantages of BlenX in such a scenario.

The term self-assembly indicates a process in which disordered components form an organized structure or pattern through only local interactions and without external coordination. Debates of how complex structures and functions can emerge from local interactions between simple components can be found in many different fields (e.g. nanotechnology [37], robotics [62], molecular biology [47, 115], autonomous computation [78]).

Here we concentrate our attention on inter-molecular self-assembly. We are particularly interested in, i.e. the ability of proteins to form quaternary structures), which is a crucial process to the functioning of cells.

In the following section we will present some examples of self-assembly. We start with very simple BlenX programs that build filaments and trees of boxes. After that we introduce some modifications in order to obtain programs that produce trees with some peculiar structures.

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 complex. Although this is a simplification, the programs we present give a flavor of the potential of BlenX. In all programs, the only reactions associated with finite rates are the bindings between boxes. All other actions (e.g. intra-communications, changes, inter-communications) are executed as immediate actions that atomically update the internal state of the boxes. Given that change actions are always immediate we write $\mathsf{ch}(a, T)$ in place of $\mathsf{ch}(a, T, \infty)$. In this section we are not interested in investigating the effect of associating different rates to actions, thus all the binding happens at the same rate r.

For the sake of clarity, in the following we associate bio-processes and internal processes with identifiers. Moreover, even if it is not formally allowed by the syntax, we do not write the nil term at the end of a process (e.g. we write $\mathsf{ch}(a, T)$ in place of $\mathsf{ch}(a, T).\,\mathsf{nil}$) and we write $n?-.\,P$ in place of $n?x.\,P$ if $x \notin \mathsf{fn}(P)$. We refer to $n?-$ as *empty input*. We also write $n!-$ for *empty outputs*. An output is defined to be empty if it exclusively synchronizes with empty inputs. These choices make the presentation of the programs more compact and readable.

3.4.1 Filaments

A polymerization process is one in which large molecules (polymers) are constructed from small units (monomers) that can bind together. Such a process give rise to the formation of molecules

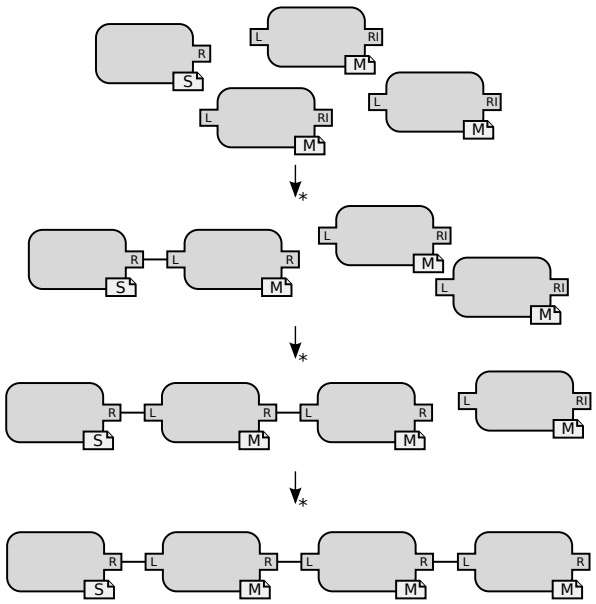
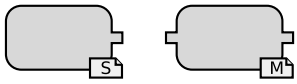


Figure 3.6 – Example of filament formation.

like DNA and actin filaments.

Let us consider a simple scenario in which a box `M` representing a monomer can bind to boxes of its own species to generate a filament of boxes. For simplicity, and for finer control over the creation of the filaments, we will also introduce a *seed* box `S`. Its role is to recruit the first monomer and start the formation of a filament. The graphical intuition of the structures of the boxes `S` and `M` is given in the following diagram:



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 behaviour of the model is depicted in Figure 3.6 (the starred arrow represents one or more steps).

Notice that a filament can grow only to the right. 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 boxes `S` and `M`:

$$\begin{aligned} \texttt{S} &\triangleq_B (\mathit{right},\, R)[\,\mathsf{nil}\,] \\ \texttt{M} &\triangleq_B (\mathit{left},\, L)(\mathit{right},\, RI)[\,\langle(\mathit{left}, \otimes)\rangle \mathsf{ch}(\mathit{right}, R)\,] \end{aligned}$$

We define the binding affinity function α_b such that it is zero for all the possible pairs of sorts with the only exception of R and L: $\alpha_b(R, L) = r$. Due to this compatibility, 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 sort of the right interface from RI (R Inactive) into R. In this way, the *right* interface of the bound `M` becomes capable of binding to the *left* interface of another free monomer (see Figure 3.6).

3.4.2 Trees

It is easy to modify the previous program to create trees. We modify the box `M` by adding a third interface, which we refer to by the identifier *branching*. Like the *right* interface, the *branching* one is activated via change actions only when `M` is bound to something on its *left* interface.

$$\begin{aligned} \texttt{M} \quad \triangleq_B \quad & (\mathit{left},\, L)(\mathit{right},\, RI)(\mathit{branching},\, BI) \\ & [\,\langle(\mathit{left}, \otimes)\rangle.\,\mathsf{ch}(\mathit{right}, R).\,\mathsf{ch}(\mathit{branching}, B)\,] \end{aligned}$$

We also add a *branching* 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. `T` is defined as:

$$\texttt{T} \quad \triangleq_B \quad (\mathit{left},\, TL)(\mathit{right},\, TRI)[\,\langle(\mathit{left}, \otimes)\rangle.\,\mathsf{ch}(\mathit{right}, TR)\,]$$

As with `M`, the internal program changes the structure of the *right* interface once the box becomes bound on the *left* interface. As final step we modify the binding capability function adding the following capabilities: $\alpha_b(B, TL) = r$ and $\alpha_b(L, TR) = r$. The first affinity enables the binding of a box `M` that is part of a filament with a free box `T`, the second one enables the binding of a box `T` that is bound on the *left* interface 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 Figure 3.7.

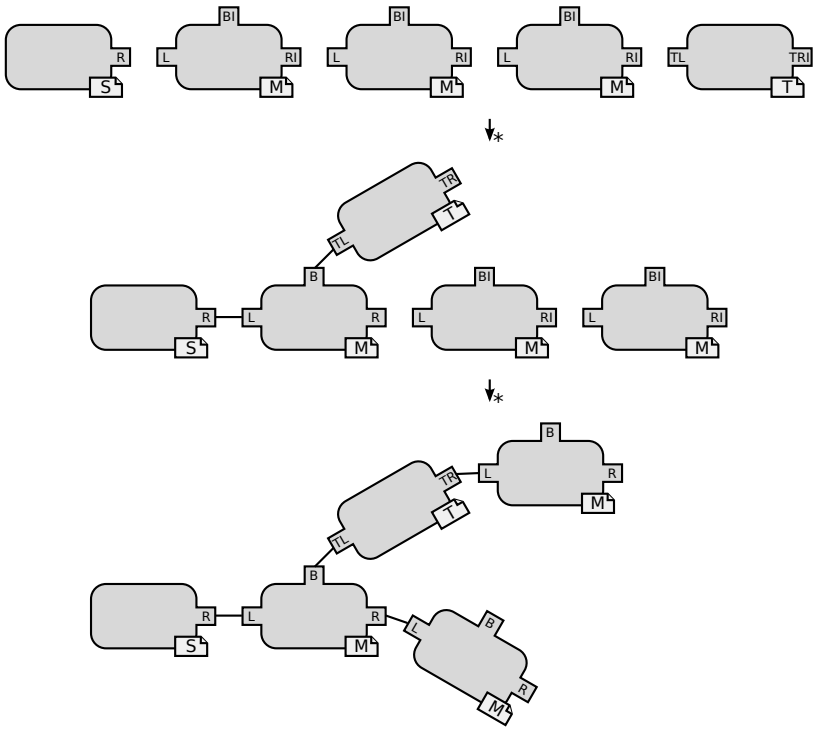


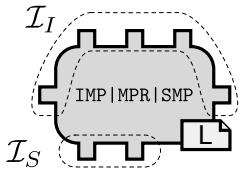
Figure 3.7 – Generation of a tree.

3.4.3 Introducing Controls over Branching Depth and Distance

So far we have seen how to program boxes in order to form large structures. Now we will introduce some controls over the formation of this structures; in particular, we will present

a program that builds trees with a maximum depth and a program that imposes a minimum distance between consecutive branches. Finally we will define a third program where we combine the two previous models to obtain trees with both the characteristics. We will define the first two programs following a schema that will make easier their merging.

This schema requires to partition the interfaces of each box into *interaction interfaces* and *state interfaces*, denoted by $\mathcal{I}_I$ and $\mathcal{I}_S$, respectively. Interaction interfaces are used by a box to interact with other boxes, while state interfaces are used to store information regarding the box state. Note that state interfaces do not have any biological interpretation, they are used because storing information in the interfaces simplifies the coding; indeed all the components of the internal program can access this information easily by managing and checking interface structures through change actions and if statements. The internal program of each box consists of three parallel processes. We call them the *interaction modifier process* (IMP), the *state modifier process* (SMP) and the *messages receiver process* (MRP). Here an illustration of the structure of a box defined with this schema:



The IMP, MRP and SMP processes are defined following a list of criteria that restrict their behaviour and the possible actions they can perform:

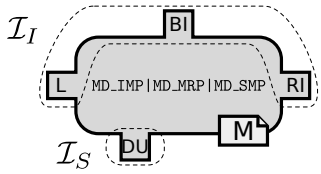
- IMP can modify and check the state of the interfaces belonging to $\mathcal{I}_I$ and only check the interfaces belonging to $\mathcal{I}_S$. However, it cannot perform inputs and outputs.
- MRP receives incoming messages from $\mathcal{I}_I$ interfaces and executes outputs over received names. We mechanically built it from the set of the interfaces $\mathcal{I}_I$. It is defined as

  $$*a_1?x.\, x!- \mid \cdots \mid *a_1?x.\, x!-$$

  where $\{a_1, \ldots, a_n\} = \mathcal{I}_I$.
- SMP can modify and check the state of the interfaces belonging to $\mathcal{I}_S$. It cannot perform inputs over interaction interfaces but it can perform outputs.

Controlling branching depth.

Each node of a tree can be associated with a branching depth level. A node has branching depth n if there are exactly $n-1$ branches from the node to the root. A tree has branching depth defined as the maximum branching depth of any of its nodes. We define a BlenX model that generates trees with branching depth less than five. The boxes involved in this model represent monomers, branches and seeds. The BlenX graphical and textual representation of monomers is:



$$\texttt{M} \triangleq_B (right,\, RI)(left,\, L)\ (branch,\, BI)(depth,\, DU)\ [\,\texttt{MD\_IMP} \mid \texttt{MD\_MRP} \mid \texttt{MD\_SMP}\,]$$

The only interface belonging to $\mathcal{I}_S$ has subject *depth* and can be associated with five different sorts: DU, $D1$, $D2$, $D3$ and $D4$. Sort DU is used when the box is free, while the others are used when the box is bound and they encode the branching depth of the node, e.g. $D1$ represents a branching depth of 1. We refer to this interface as the *depth* interface.

The *interaction modifier process* MD_IMP enables the binding on the *right* interface by changing its sort when the box binds on the left interface. Moreover the MD_IMP process changes the sort of the *branching* interface depending on the sort of the *depth* interface. In particular, the sort has to be changed in order to only allow the box to branch if the sort exposed on the *depth* interface is either $D1$, $D2$, or $D3$. Hence, MD_IMP is defined as follows:

$$\begin{aligned}\texttt{MD\_IMP} \triangleq_P\ & \langle(left, \otimes)\rangle \mathsf{ch}(right, R) \\ & \mid \langle\neg((depth, D4) \vee (depth, DU))\rangle \mathsf{ch}(branch, B)\end{aligned}$$

As previously explained we construct the *message receiver process* MD_MRP by composing in parallel one replicated process for each interface belonging to $\mathcal{I}_I$:

$$\begin{aligned}\texttt{MD\_MRP} \triangleq_P\ & *right?channel.\ channel!- \\ & \mid *\ left?channel.\ channel!- \\ & \mid *\ branch?channel.\ channel!-\end{aligned}$$

The last process composing the internal program is the *state modifier process* MD_SMP. Its role is to modify the sort associated with the *depth* interface in order to encode the number of branches from the root to the box. This task is performed through an exchange of messages between the *state modifier* processes of different boxes. The MD_SMP process is composed by two parallel processes, one receiving and using messages for updating the state interfaces, and the other sending messages to bound boxes and triggering their update:

$$\texttt{MD\_SMP} \triangleq_P \texttt{MD\_r\_SMP} \mid \texttt{MD\_s\_SMP}$$

These processes exchange information using four channel names: $d1$, $d2$, $d3$ and $d4$. MD_r_SMP is a sum of inputs over channels that we use to trigger modifications of the sort associated with *depth* interface:

$$\begin{aligned}\texttt{MD\_r\_SMP} \triangleq_P\ & d1?-.\mathsf{ch}(depth, D1) \\ & + d2?-.\mathsf{ch}(depth, D2) \\ & + d3?-.\mathsf{ch}(depth, D3) \\ & + d4?-.\mathsf{ch}(depth, D4))\end{aligned}$$

The interacting partner of MD_r_SMP is:

$$\begin{aligned}\texttt{MD\_s\_SMP} \triangleq_P\ & (\langle(depth, D1)\rangle right!d1 \\ & + \langle(depth, D2)\rangle right!d2 \\ & + \langle(depth, D3)\rangle right!d3 \\ & + \langle(depth, D4)\rangle right!d4) \\ & \mid (\langle(depth, D1)\rangle branch!d1 \\ & + \langle(depth, D2)\rangle branch!d2 \\ & + \langle(depth, D3)\rangle branch!d3) \\ & + \langle(depth, D4)\rangle branch!d4)\end{aligned}$$

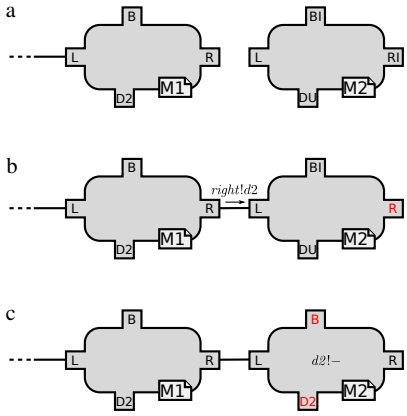


Figure 3.8 – An instance of the execution of the protocol allowing the propagation of the depth information from a leaf of a tree to a newly attached box.

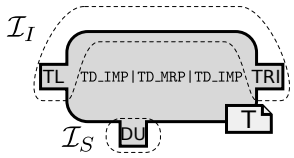
This process is composed of two parallel processes. Their task is to propagate the information regarding the depth of the box to newly attached boxes. The first process propagates this information along the *right* interface, the second one along the *branch* interface. Given that they share the same structure we only provide an explanation for the first one.

This process is a sum of outputs over the *right* interface which are guarded by conditions that check for the sort associated with the *depth* interface. These checks are mutually exclusive and thus, in a given instant, only one of the guarded outputs is active. When the box binds, the active output sends a channel name encoding the depth of the box it is sent from. The channel name is received by the `MD_MRP` process of the newly attached box which, as required by the adopted schema, performs an output over it. This output synchronizes with one of the input of the `MD_r_SMP` of the newly attached box and causes the correct update of its *depth* interface.

In Figure 3.8 we can see an instance of the described communication protocol. The involved boxes are the monomer `M1`, a leaf of a tree, and the monomer `M2`, a free box. The two boxes bind and the `MD_r_SMP` of `M1`, given that its *depth* interface is associated with the sort *D2*, sends the channel name *d2* over the right interface (Figure 3.8b). The `MD_MRP` process of `M2` receives the channel name over the *left* interface and performs an output over it. This output interact with the process `MD_r_SMP` inside the `M2` box, in particular it synchronizes with the process guarded by the input $d2?-$ (In Figure 3.8c, this intra-communication is represented by the output $d2!-$ inside the box `M2`). The execution of this input activate the change action that updates the sort of the *depth* interface to *D2* (Figure 3.8c). Note that in this protocol the `MD_IMP` process acts in parallel. Indeed in the states represented in Figure 3.8b and 3.8c the conditions $\langle(\mathit{left}, \otimes)\rangle$ and $\langle\neg((\mathit{depth}, \mathit{D4}) \vee (\mathit{depth}, \mathit{DU}))\rangle$ evaluate to true, respectively and thus the sorts of the *right* and the *branch* interfaces are updated to R and B.

In this communication protocol the immediate actions play a fundamental role, in fact they allow the atomic execution of several actions.

Now we proceed with the description of the branching box `T`:



$$\texttt{T} \triangleq_B (\mathit{right},\ \mathit{TRI})(\mathit{left},\ \mathit{TL})(\mathit{depth},\ \mathit{DU})\ [\,\texttt{TD\_IMP} \mid \texttt{TD\_MRP} \mid \texttt{TD\_SMP}\,]$$

Also in this case the interaction interfaces of the box are the same we introduced in the previous section. The interaction modifier process `TD_IMP` has only to change the sort of the *right* interface sort when the box binds on the left:

$$\texttt{TD\_IMP} \triangleq_P \langle(\mathit{left}, \otimes)\rangle \mathsf{ch}(\mathit{right}, \mathit{TR})$$

The message receiver process `TD_MRP` is defined mechanically following the schema previously described.

The state modifier process `TD_SMP`, with a structure which is similar to the `MD_SMP`, propagates the depth information to the node which will bind to the *right* interface. In this case, given that the message is received by a branching box, the depth level is increased by one. Thus, if the node receives a message encoding depth equal to one (i.e. *d1*) the *depth* interface is actually associated with depth *D2*. Note that here we do not need to consider depth *D1*, because a branch node must have a branching depth of at least two.

$$\begin{aligned}\texttt{TD\_SMP} \triangleq_P\ & \mathit{d1}?-.\,\mathsf{ch}(\mathit{depth}, \mathit{D2}) \\ & + \mathit{d2}?-.\,\mathsf{ch}(\mathit{depth}, \mathit{D3}) \\ & + \mathit{d3}?-.\,\mathsf{ch}(\mathit{depth}, \mathit{D4}))\\ & \mid \langle(\mathit{depth}, \mathit{D2})\rangle \mathit{right}!\mathit{d2} \\ & + \langle(\mathit{depth}, \mathit{D3})\rangle \mathit{right}!\mathit{d3} \\ & + \langle(\mathit{depth}, \mathit{D4})\rangle \mathit{right}!\mathit{d4})\end{aligned}$$

The seed box `S` does not have any state interface and its BlenX graphical and textual descriptions are:

$$\texttt{S} \triangleq_B (\mathit{right},\ R)\ [\,\texttt{SD\_IMP} \mid \texttt{SD\_MRP} \mid \texttt{SD\_SMP}\,]$$

The interaction capabilities of this box never change and therefore the interaction modifier process `SD_IMP` 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:

$$\texttt{SD\_SMP} \triangleq_P \mathit{right}!\mathit{d1}$$

Figure 3.9 shows an example of complex generated by the branching depth control program.

Controlling branching distance.

Now we consider the generation of trees that have at least four monomers between their branches. We store in each monomer belonging to a filament its distance from the nearest branch (the seed node is considered to be a branch). To store this information we use a distance interface that can be associated with the sorts *FU*, *F1*, *F2*, *F3* and *F4*. The sort *FU* is used when the

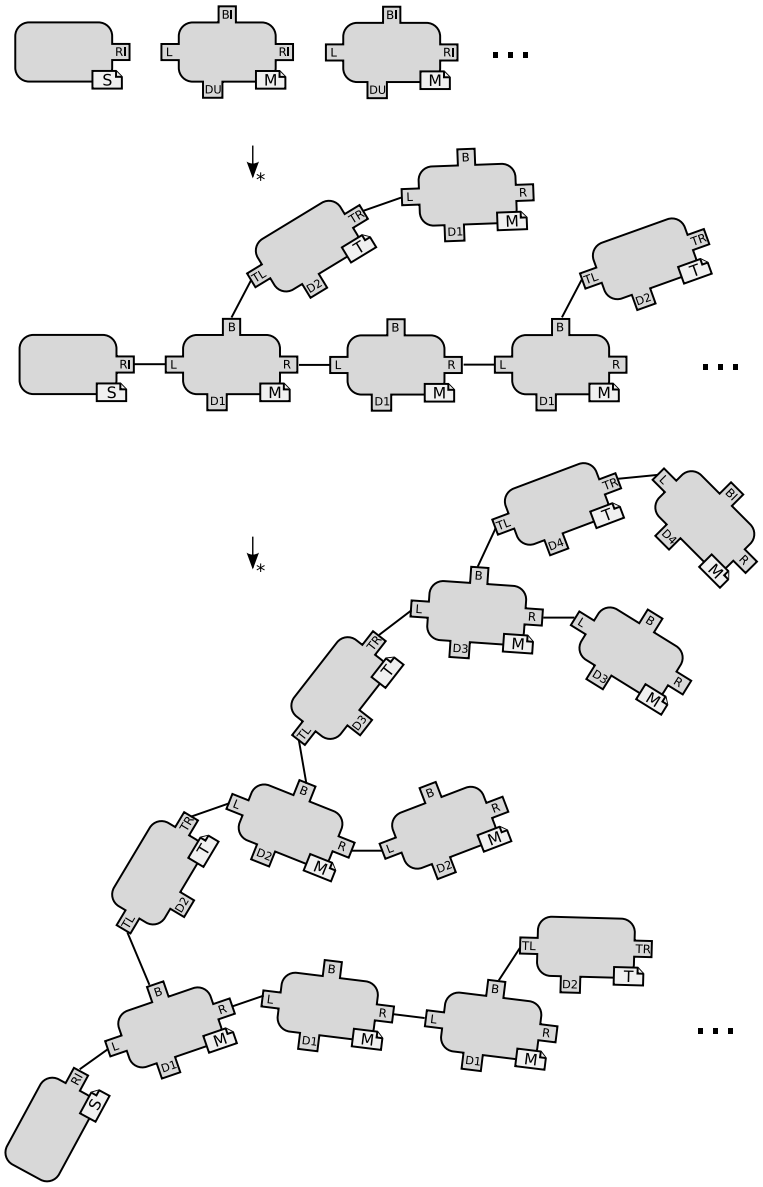
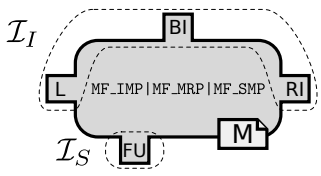


Figure 3.9 – Example of a tree generated with the branching depth control program. Notice how at level four the monomer branching interfaces are inactive. Dots indicate the presence of other boxes in the system.

monomer box is free while the others when it is bound. The number indicates the distance from the nearest branch with the exception of $F4$, which means that the distance is greater than or equal to four. The BlenX code of `M` is:



$$\texttt{M} \triangleq_B (right, RI)(left, L)\ (branch, BI)(distance, FU)\ [\,\texttt{MF\_IMP} \mid \texttt{MF\_MRP} \mid \texttt{MF\_SMP}\,]$$

The interaction modifier process MF_IMP permits the binding to the *right* interface when the box is bound on the *left* interface and permits a branch to form only if the nearest branch is at least four nodes away. Hence, if the *distance* interface has sort *F4*, the process changes the sort of the *branch* interface into B, allowing the branch formation. It then enables a change action guarded by a condition that checks for modifications to the *distance* interface. A modification can happen if a branch starts growing in the proximity of the box. In this case, the MF_SMP process changes the sort of the *distance* interface from *F4* to the sort that encodes the new distance and the MF_IMP disables the *branch* interface by changing its sort from B into BI.

$$\begin{aligned}\texttt{MF\_IMP} \triangleq_P\ & \langle (\mathit{left}, \otimes)\rangle \mathsf{ch}(\mathit{right}, R) \\ & \mid \langle (\mathit{distance}, \mathit{F4})\rangle \mathsf{ch}(\mathit{branch}, B). \\ & \quad \langle \neg(\mathit{distance}, \mathit{F4}) \wedge (\mathit{distance}, \odot)\rangle \mathsf{ch}(\mathit{branch}, BI)\end{aligned}$$

Given that the MF_MRP process has the same definition of MS_MRP we proceed with the BlenX code description of the state modifier process MF_SMP:

$$\begin{aligned}\texttt{MF\_SMP} \triangleq_P\ & \mathit{switch}!- \\ & \mid *\, \mathit{switch}?-.\, (\\ & \quad \langle (\mathit{distance}, \mathit{F1})\rangle \\ & \quad (\mathit{f1}?- + \mathit{f2}?- + \mathit{f3}?- + \mathit{f4}?-) \\ & \quad + \langle (\mathit{distance}, \mathit{F2})\rangle \\ & \quad (\mathit{f1}?-.\, \mathsf{ch}(\mathit{distance}, \mathit{F1}).\, \mathit{propagate}!\mathit{f2} \\ & \quad + \mathit{f2}?- + \mathit{f3}?- + \mathit{f4}?-) \\ & \quad + \langle (\mathit{distance}, \mathit{F3})\rangle \\ & \quad (\mathit{f1}?-.\, \mathsf{ch}(\mathit{distance}, \mathit{F1}).\, \mathit{propagate}!\mathit{f2} \\ & \quad + \mathit{f2}?-.\, \mathsf{ch}(\mathit{distance}, \mathit{F2}).\, \mathit{propagate}!\mathit{f3} \\ & \quad + \mathit{f3}?-.\, \mathit{switch}!- + \mathit{f4}?-.\, \mathit{switch}!-) \\ & \quad + \langle (\mathit{distance}, \mathit{F4}) \vee (\mathit{distance}, \mathit{FU})\rangle \\ & \quad (\mathit{f1}?-.\, \mathsf{ch}(\mathit{distance}, \mathit{F1}).\, \mathit{propagate}!\mathit{f2} \\ & \quad + \mathit{f2}?-.\, \mathsf{ch}(\mathit{distance}, \mathit{F2}).\, \mathit{propagate}!\mathit{f3} \\ & \quad + \mathit{f3}?-.\, \mathsf{ch}(\mathit{distance}, \mathit{F3}).\, \mathit{switch}!- \\ & \quad + \mathit{f4}?-.\, \mathsf{ch}(\mathit{distance}, \mathit{F4}).\, \mathit{switch}!-)) \\ & \mid \langle (\mathit{distance}, \mathit{F1})\rangle \mathit{right}!\mathit{f2} \\ & \quad + \langle (\mathit{distance}, \mathit{F2})\rangle \mathit{right}!\mathit{f3} \\ & \quad + \langle (\mathit{distance}, \mathit{F3}) \vee (\mathit{distance}, \mathit{F4})\rangle \mathit{right}!\mathit{f4} \\ & \mid *\, \mathit{propagate}?\mathit{distance}.\, (\\ & \quad (\langle (\mathit{right}, \otimes)\rangle \mathit{right}!\mathit{distance} \\ & \quad + \langle (\mathit{right}, \odot)\rangle \mathit{kill}!-) \\ & \quad \mid \\ & \quad (\langle (\mathit{left}, \otimes)\rangle \mathit{left}!\mathit{distance})) \\ & \mid *\, \mathit{kill}?-\end{aligned}$$

The first parallel component is an empty output over the *switch* channel. As soon as the program starts running, in each node, this output synchronizes with the empty input that guards the replication of the second parallel component and makes active a copy of the body of the replication. The body of the replication is a sum of processes that are guarded by conditions on the sort associated with the *distance* interface. These conditions are mutually exclusive therefore, in a given instant, only one of the processes following the conditions is active. These

processes wait for an output over one of the channel encoding distances (i.e. *f1*, *f2*, *f3* and *f4*) which is a request to change the sort associated with the *distance* interface (to *F1*, *F2*, *F3* and *F4*, respectively). Such a change is performed only if the distance encoded by the new sort is smaller than the distance encoded by the sort associated with the interface.

For the understanding of this communication protocol we exploit an instance of its execution (Figure 3.10). The involved boxes are `M1`, `M2`, `M3` and `M4`. `M1` is a free box instead `M1`, `M2` and `M3` are part of a tree structure and as it is encoded in the sort associated with their *distance* interface (that is *F4*), they are distant four or more than four nodes from the nearest branch. In the first step, shown in Figure 3.10b, the box `M1` binds to the right interface of `M2`. This action causes the `MF_IMP` process of the box `M1` to change the sort of the right interface to R, and the third parallel component of the `MF_SMP` process of the box `M2` to perform an output. Such an output is performed by the process guarded by the condition $\langle(distance, F3) \vee (distance, F4)\rangle$ which sends the channel name *f4* over the *right* interface (see Figure 3.10b).

The process `MF_MRP` of the box `M1` receives the channel name *f4* and performs an output over it. This output synchronizes with the active copy of the body of the replication guarded by the input $switch?-$. In particular it interacts with the sum guarded by the condition $\langle(distance, F4) \vee (distance, FU)\rangle$. Given that the communication is over *f4* the interacting component of the involved sum is the fourth one. This component associates the sort *F4* with the *distance* interface and then sends an output in order to start a new copy of the body of the replication guarded by the action $switch?-$. The association of the sort *F4* with the *distance* interface is recognized by the `MF_IMP` process that consequently changes the type of the *branch* interface to B (see Figure 3.10c).

At this point we introduce a new participant, the branching box `T` (in Figure 3.10d we only represent the interacting interface of this box), which binds to the *branching* interface of the box `M1`. Without considering in detail the internal process of this box, that we will describe later, we only say that, after the binding, it sends the channel name *f1* over its *left* interface. The `MF_MRP` process of the box `M2` receives the channel name and sends an empty message over it. This output, as happened in the previous step when the box `M1` synchronized a communication on the channel name *f4*, interacts with the active copy of the body of the replication guarded by the action $switch?-$. In particular the output again interacts with the sum guarded by the condition $\langle(distance, F4) \vee (distance, FU)\rangle$. However this time, given that the communication is over *f1* (instead of *f4*), the interacting component of the involved sum is the first one. This component updates the information regarding the distance from the nearest branch by associating the sort *F1* with the *distance* interface. It then performs an output on the channel *propagate* which carries as message the channel name *f2* (see Figure 3.10d).

This output activates a copy of the body of the replication guarded by $propagate?distance$ where the variable *distance* is replaced with the channel name *f2*. This newly activated process propagates the information about the distance (in this case the name channel *f2*) along both the *right* and the *left* interfaces. The process is made of two processes joint by means of the parallel operator. The first one is a sum that enables two alternative actions guarded by a condition. The first one checks if the *right* interface is bound, if it is the case it enables an output over the *right* interface that propagates the distance information, the second one checks if the *right* interface is free and in this case enables an output over the channel *kill* that interacts with the replicated input $kill?-$. In our example the *right* interface is free and thus the communication over the *kill* channel takes place. The only effect of this communication is to eliminate from the box the alternative communication over the *right* interface, in this way we eliminate an

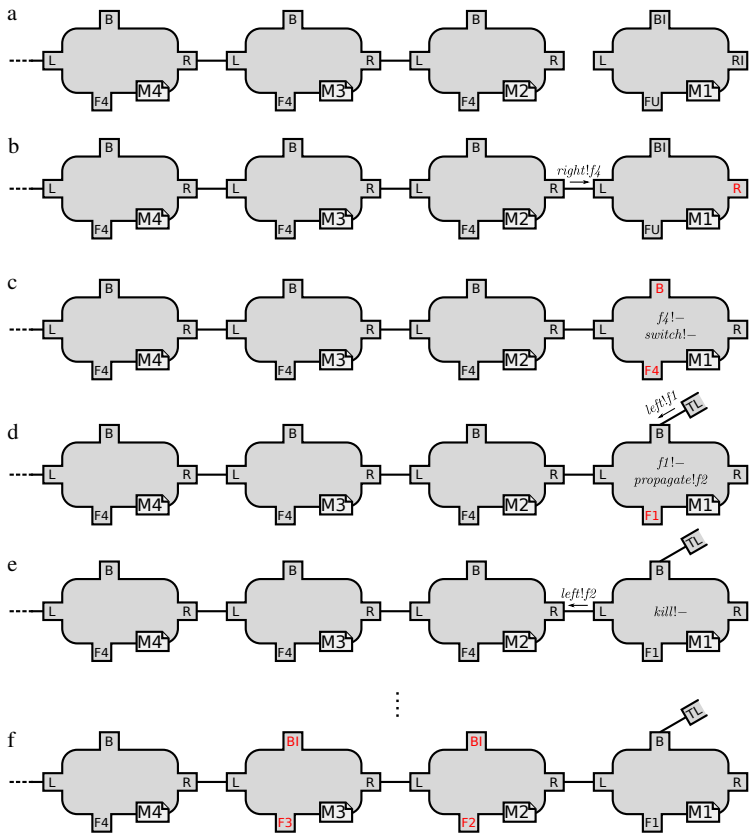


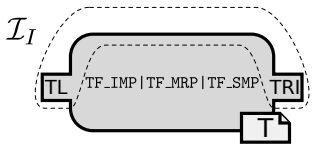
Figure 3.10 – An instance of execution of the protocol updating *distance* interfaces in order to make them encode the distance from the closest branch.

undesired pending communication. The propagation mechanism of the information along the *left* interface is simpler, indeed we know that the *left* interface is always bound and thus we can simply perform the output without the risk to generate a communication that will be pending because it cannot be immediately performed (see Figure 3.10e).

After this step (Figure 3.10 does not show the following actions) the box `M2` receives the channel name *f2* and performs a series of actions similar to the one we have just described. In this case the internal program of `M2` associates the *distance* interface with *F2* and propagates the channel name *f3*. The box that receives the channel name *f3* associates the sort of its *distance* interface with *F3* but it does not propagate the channel name *f4*. In fact this information cannot trigger any update of the *branch* interface of the surrounding boxes (given that the distance encoded by *F4* cannot be smaller than the distance encoded by another sort). Note that the `MF_IMP` process of the boxes `M2` and `M3`, when their *distance* interfaces are associated with sort *F2* and *F3*, respectively, associates the sort *BI* to the *branch* interface. In this way the boxes `M2` and `M3` are prevented to form a branch. At this point the protocol finished its job, indeed all the nodes of the tree have properly updated their *distance* and *branch* interfaces (see Figure 3.10f).

Now we introduce the definition of the branching box `T`. Here, there are no state interfaces, because the branching box does not have to store any information. Indeed once it is part of a

tree, its distance from a branch is zero given that it is the branch itself. Therefore a *distance* interface is useless and the structure of the box results to be the following:



$$\texttt{T} \triangleq_B (\mathit{right},\ \mathit{TRI})(\mathit{left},\ \mathit{TL}) \\ [\,\texttt{TF\_IMP} \mid \texttt{TF\_MRP} \mid \texttt{TF\_SMP}\,]$$

Note that the interface modifier process `TF_IMP` is the same as in the previous model (`TD_IMP`). Even though there are no state interfaces, the state modifier process `TF_SMP` is not `nil` because it has to cooperate with the state modifier processes of the other boxes in order to propagate distance information. In particular, when a box binds to the *left* or to the *right* interface, `TF_SMP` has to communicate to the new neighbour that its new distance from a branch is one:

$$\texttt{TF\_SMP} \triangleq_P \mathit{left}!\mathit{f1} \\ \mid \mathit{right}!\mathit{f1}$$

The seed box `S` has no state information, like the branching box `T`. Its structure is the following:

$$\texttt{S} \triangleq_B (\mathit{right},\ R) \\ [\,\texttt{SF\_IMP} \mid \texttt{SF\_MRP} \mid \texttt{SF\_SMP}\,]$$

Here the interaction modifier process `SF_IMP` is empty (`nil`) because the interaction capabilities of the box never change. The state modifier process `SF_SMP` has the same role of the branch box `TF_SMP` process with the difference that there is only one interface that another box can bind to:

$$\texttt{SF\_SMP} \triangleq_P \mathit{right}!\mathit{f1}$$

Figure 3.11 shows an example of a tree produced by the branching distance control program.

Merging the behaviours.

We can merge the behaviours of the two models to generate trees with a depth level less than five *and* with a branching distance of at least four monomers. To do this, we create a new seed box `S`, a new branching box `T` and a new monomer box `M`, starting from the corresponding definitions from the two previous models. We use $\texttt{S}^d$, $\texttt{T}^d$ and $\texttt{M}^d$ to denote the box definitions for the branching depth control model, and $\texttt{S}^b$, $\texttt{T}^b$ and $\texttt{M}^b$ for the branching distance control model. Each box $\texttt{B} \in \{\texttt{S}, \texttt{T}, \texttt{M}\}$ has the same interaction interfaces of $\texttt{B}^d$ and $\texttt{B}^b$ and its state interfaces are the union of the state interfaces of $\texttt{B}^d$ and $\texttt{B}^b$. The state modifier process of each box `B` is obtained by composing in parallel the state modifier processes of $\texttt{B}^d$ and $\texttt{B}^b$. Moreover, the message receiver process is the same as that of $\texttt{B}^d$ and $\texttt{B}^b$ because the two boxes have the same interaction interfaces. Finally we define a new interaction modifier process. Thanks to the adopted schema, the process of merging the two models is easy. We only need to define a new interaction modified process (`IMP`).

The box `M` of this merged model is:

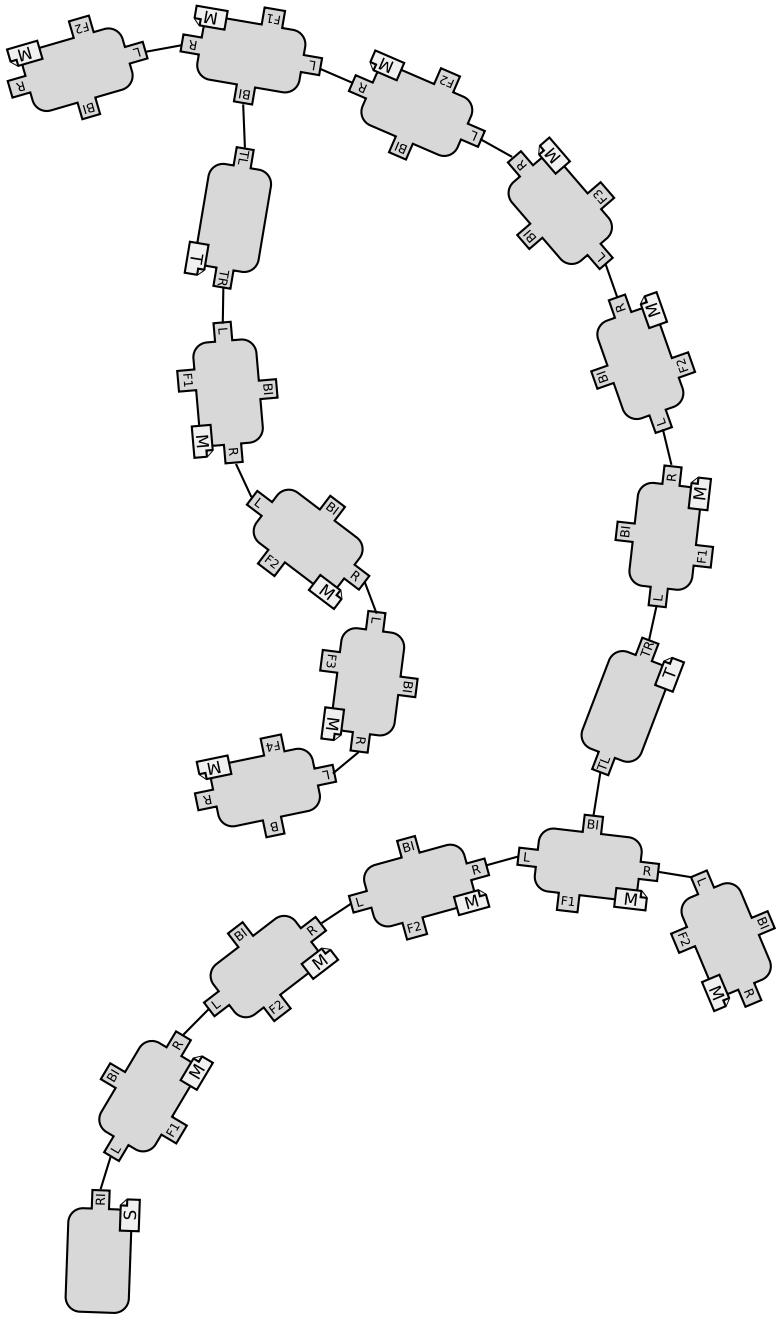
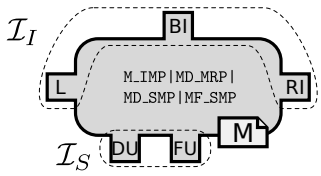


Figure 3.11 – Example of a tree generated by the branching distance control program.



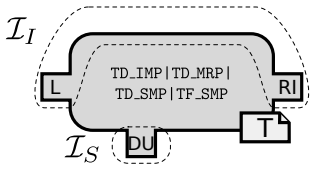
$$\texttt{M} \triangleq_B (\mathit{right}, \mathit{RI})(\mathit{left}, L)(\mathit{branch}, \mathit{BI})(\mathit{depth}, \mathit{DU})(\mathit{dist}, \mathit{FU})[\texttt{M\_IMP} \mid \texttt{MD\_MRP} \mid \texttt{MD\_SMP} \mid \texttt{MF\_SMP}]$$

We define an interaction modifier process that allows the binding on the *right* interface only if

the box is bound on the *left* interface and that allows the growth of a branch only if the box has a depth level less then five, and is at least three monomers from a branch:

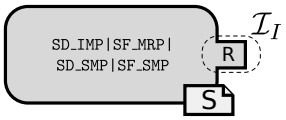
$$\begin{aligned}\texttt{MD\_IMP} \triangleq_P\ & \langle(\mathit{left},\otimes)\rangle \mathsf{ch}(\mathit{right},R)\\ & |\,\langle(\mathit{distance},F4)\wedge\neg(\mathit{depth},D4)\wedge\neg(\mathit{depth},DU)\rangle\\ & \quad \mathsf{ch}(\mathit{branch},B).\\ & \quad \langle\neg(\mathit{distance},F4)\rangle\mathsf{ch}(\mathit{branch},BI)\end{aligned}$$

Merging the behaviour of the boxes $\texttt{T}^b$ and $\texttt{T}^d$ is easier because the interaction modifier processes of the branch boxes of the models we are merging are the same ($\texttt{TD\_IMP} = \texttt{TF\_IMP}$). Thus we can use one of them:



$$\begin{aligned}\texttt{T} \triangleq_B\ & (\mathit{right},\ \mathit{TRI})(\mathit{left},\ \mathit{TL})\\ & (\mathit{depth},\ \mathit{DU})\\ & [\,\texttt{TD\_IMP}\,|\,\texttt{TD\_MRP}\,|\,\texttt{TD\_SMP}\,|\,\texttt{TF\_SMP}\,]\end{aligned}$$

Similarly, we can build the merged seed box S of this model as follows, because $\texttt{SD\_IMP} = \texttt{SF\_IMP}$:



$$\begin{aligned}\texttt{S} \triangleq_B\ & (\mathit{right},\ R)\\ & [\,\texttt{SD\_IMP}\,|\,\texttt{SD\_MRP}\,|\,\texttt{SD\_SMP}\,|\,\texttt{SF\_SMP}\,]\end{aligned}$$

Note that in these models we used sorts to encode counters. Since sorts are not numbers, no arithmetic on them is possible. Hence we have to represent each possible number with a different sort and implement addition and subtraction operations with processes that change the sorts of interfaces in the appropriate way (see for example the previous definition of TD_SMP process). It is clear that our approach results in very verbose code when dealing with wide ranges of numbers. However, since all these operations can be implemented following the same template, a possible solution is to implement them by adding macros to the BlenX that can be encoded at compile time. An alternative solution can be found in the BlenX extension presented in [97], where interface sorts can be numbers which are manipulated through arithmetic operations.

Chapter 4

Actin

In this chapter we introduce a case study to support the presentation of our analysis. The adopted example is the formation of actin filaments. Actin is a small globular protein present in almost all known eukaryotes. It is one of the most conserved protein among species because it is involved in fundamental cell processes, e.g. muscle contraction, cell motility, cell division and cytokinesis, vesicle and organelle movement, cell signaling, and the establishment and maintenance of cell junctions and cell shape. All these tasks are performed thanks to the ability of actin to polymerize in long polymerized filaments [70] whose tips are called *barbed* and *pointed*. They can interact with a multitude of molecules; for example actin can bind with the ARP2/3 complex, a seven subunit protein which, once bound, acts as a nucleation site for actin monomers, giving rise to branched filaments. This leads to the formation of tree like structures that are important for processes like cell locomotion, phagocytosis, and intracellular motility of lipid vesicles. Because of their importance in cell life, actin filaments are widely studied and modelling techniques are often employed. Here we present the approaches adopted in literature, highlighting their advantages and their limitations and finally we explain why we believe BlenX a good tool for modelling this kind of protein.

The first modelling attempts are based on ODEs. As previously explained ODE-based approaches define all the complexes involved in the system with different variables. In the case of actin this limits the approach because filaments can assume a finite but really huge number of arrangements, therefore it is necessary to introduce strong approximations [88, 19, 77] in order to reduce the number of necessary variables. However ODE-based techniques can provide useful information and abstractions, even if their application to represent the low-level dynamics underlying the formation and evolution of actin filaments is usually quite difficult and sometimes not feasible.

Better results can be obtained by adopting methods which introduce species for representing the possible conformations of the actin monomers and polymerization reactions for composing them in complexes. These kinds of models, requiring only the definition of the complexes bricks, avoid to list all the possible arrangements that actin filaments can assume. Preliminary attempts to use these approaches are in [41, 73] where the authors define ad hoc programs implementing the Gillespie algorithm to perform simulations. The limitation in this case is the absence of distinction between the model and the simulator. This makes it impossible to reuse the work done for other purposes (for example for performing simulations of different models) and moreover makes it difficult to check for the correctness of the model implementation.

In [17] the authors introduce a clear distinction among model and simulator. They define

a model of actin polymerization in stochastic π-calculus [95] and simulate it through the SPiM simulator [90]. This approach has numerous advantages. The process calculi paradigm [98] allows indeed applying the metaphor of autonomous entities that are interacting with each other in order to build complex structures. Moreover, models can be written and modified without acting on the simulator implementation and the correctness of the simulator has to be verified only once against the semantics of the considered process calculus and not against each written model. The work of [17] handles the concept of complex, but complexes generated during simulation are not explicitly listed in the output of the simulation. Actin structures can be extracted from the output using the geometric plotting, but since the information used for the plotting relies on the coordinates associated to processes, there are cases in which the interpretation is ambiguous. In these cases it is not possible, for example, to retrieve the time-course of specific actin structures.

Actin is a good example of a class of problems that are effectively modelled with BlenX. In fact the language, as the previous approach, makes a clear distinction between the simulator and the model definition and is based on the process calculi metaphor; moreover some language characteristics make it possible to overcome the aforementioned limitations. The expressivity and the ductility of approaches like BlenX is paid in terms of simulation performance. For example stochastic simulation approaches are significantly slower then deterministic one (such as ODEs); moreover, in the case of ad hoc programs defined for describing models it is possible to introduce specific optimization and tricks that are not applicable to the general case. Hence we have to deal with a trade off between expressivity and performance.

Other rule-based approaches (e.g. [30]) are also candidate for modelling actin polymerization processes, since in general can be used to model self-assembling systems [28]. In [64], for example, a rule-based model of genetic regulation (with growing DNA and RNA strands and moving ribosomes) that avoids the combinatorial state-explosion is presented.

4.1 Model Definition

As explained, modelling actin molecules is a hard challenge. In this work we do not aim at defining a detailed model, our intention is to provide an example as simple as possible which makes clear the application of the analysis presented in this thesis. This model is a simplification of those available in [1] which implement the features of the SPiM [90] models presented in [17].

Here we focus on the actin capability to polymerize and form long filaments. Therefore we can take inspiration from the model presented in Section 3.4.1, keeping in mind that there are some new features to implement:

1. The first action of actin monomers is an activation process which enables their polymerization capability;
2. Actin monomers can generate a filament without a seed molecule;
3. Filaments can grow at both ends;
4. Filaments can lose monomers from their tips.

The only box involved in this model is the box `A` which represents the actin monomer. It has two interfaces which we refer as b (barbed) and p (pointed). We omit to associate rates with interfaces because they are not used for intra-communications. In the initial state interfaces

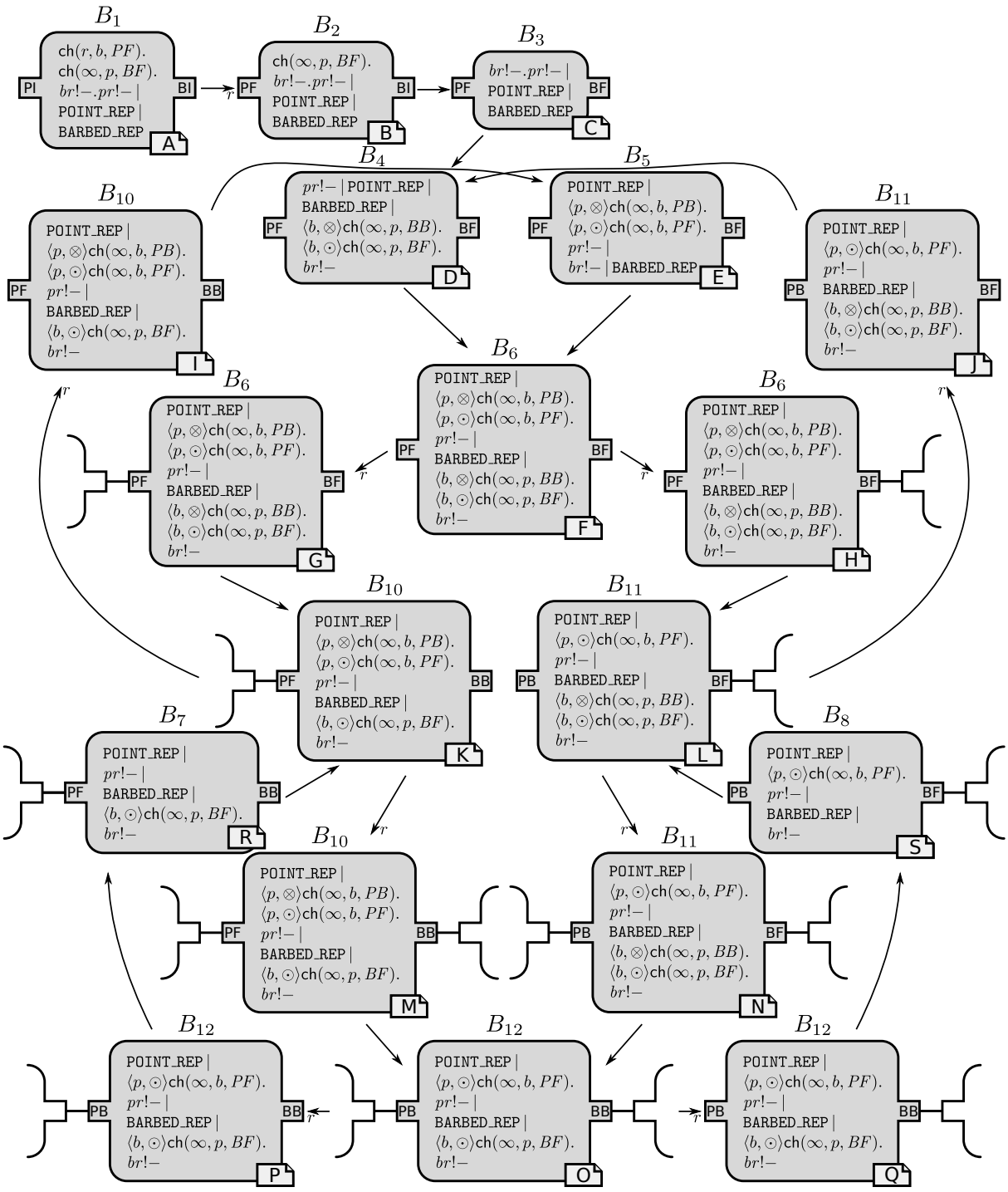


Figure 4.1 – Actin box transitions with possible binding states of the interfaces. Arrows without rate are associated with ∞.

are associated with PI and BI, respectively. For the sake of clearness sorts are associated

with meaningful names. In this case PI stays for *Pointed Inactive* and BI for *Barbed Inactive*. The capabilities of an interface depend on the binding state of the other, for this reason we associate b with sorts recalling the state of p and vice versa. Beyond PI, the barbed interface can associate with PF (*Pointed Free*), and PB (*Pointed Bound*); for the pointed interface we have the symmetric sorts BF (*Barbed Free*), and BB (*Barbed Bound*). The following list contains all the sort binding and unbinding capabilities different from zero: $\alpha_b(PF, BF)$, $\alpha_b(PF, BB)$, $\alpha_b(PB, BF)$, $\alpha_u(PF, BF)$, $\alpha_u(PF, BB)$, $\alpha_u(PB, BF)$. We associate all of them with the same rate $r \in \mathbb{R}$. This choice is obviously a simplification but keeps the model simpler and still suitable to present an analysis which does not investigate quantitative aspects. From the capability list it emerges that all the sorts can be involved in binding and unbinding actions with the exception of PI and BI.

The internal program is made of three processes joined by means of the parallel operator. The first process turns the monomer in its active state and then starts a copy of the replications defined in the other two processes. The first replication (`POINTED_REP`), depending on the binding state of the interface b, modifies the sort of the interface p; the second replication (`BARBED_REP`) changes the sort of b depending on the binding state of p.

$$\begin{aligned}
\texttt{POINT_REP} \triangleq_P\ & *pr?-. \\
& \langle p, \otimes \rangle \mathsf{ch}(\infty, b, PB). \\
& \langle p, \odot \rangle \mathsf{ch}(\infty, b, PF). \\
& pr!-
\end{aligned}$$

$$\begin{aligned}
\texttt{BARBED_REP} \triangleq_P\ & *br?-. \\
& \langle b, \otimes \rangle \mathsf{ch}(\infty, p, BB). \\
& \langle b, \odot \rangle \mathsf{ch}(\infty, p, BF). \\
& br!-
\end{aligned}$$

$$\begin{aligned}
\texttt{A} \triangleq_B\ & (p,\ BI)(b,\ PI) \\
& [\mathsf{ch}(r, b, PF).\mathsf{ch}(\infty, p, BF).br!-.pr!- \\
& \mid \texttt{POINT_REP} \\
& \mid \texttt{BARBED_REP}\ ]
\end{aligned}$$

The goal of the internal program is to keep the sort of an interface updated in such a way that it reflects the binding state of the other. Therefore, when binding or unbinding events occur, it starts a sequence of immediate actions in order to keep properly updated the sorts. The updating protocol is supported by immediate communications on channels br and pr, therefore we have $\delta(br) = \delta(pr) = \infty$.

We now refer to Figure 4.1 to understand how the state of a box `A` can possibly evolve during a simulation. Each box is associated with two labels. One is applied on the bottom-right and altogether identifies the box state with the interface binding states; the other is above the box and only refers the state of the box, this implies that the same identifier can be associated with more configurations (e.g. G, F and H are associated with the same box state B_6). The initial configuration of `A` is labelled with A. In this configuration the only active action is $\mathsf{ch}(r, b, PF)$. It starts the activation process of the monomer which, with two change actions and two communications, modifies the sort of b to BF, that of p to PF, and starts a copy of the processes guarded by the input actions $br?-$ and $pr?-$. In configuration F the monomer is free (i.e. not bound to other actin molecules), it is in active form and cannot perform any

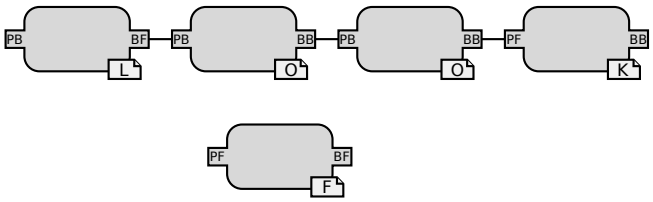


Figure 4.2 – Filament of actin boxes.

internal action; it waits for a binding which can possibly occur on both its interfaces. If the binding involves the barbed interface the monomer moves into configuration G, otherwise into configuration H. The possible evolutions from G and H are symmetric with respect to the interfaces b and p, therefore we only follow the behaviour of the monomer in configuration G. In this configuration an immediate action is active: once executed the sort associated with the barbed interface changes from BF to BB. The monomer can leave the entered configuration K for two reasons: the just formed link breaks or the pointed interface gets bound. In the first case both the interfaces are free and with a sequence of immediate actions, going through configurations I and E, the monomer goes back to F. In the second case the monomer enters the vanishing configuration M. Here the internal program recognizes the creation of the new binding and modifies the sort of b into PB moving the monomer in configuration O. Since both interfaces are bound, the internal program is blocked until one of them becomes free. The breaking of the left and the right bindings lead to symmetric evolutions with respect to the two interfaces, therefore we only follow the unbinding involving the barbed interface. In this case the monomer enters the vanishing configuration P and through two immediate actions goes back to state K, where the sort associated with the interfaces are coherent with their binding state: p remains associated with BB and b changes its sort into PF.

So far we have seen how the internal program respects the following constraint: if a monomer is in a tangible state its interface p is associated with a sort representing the binding state of b, and vice versa (with the exception of the state A where the sorts are PI and BI). Here, with the example of Figure 4.2, we show the binding and unbinding actions that this behaviour enables. Keeping in consideration the sort capabilities, the filament can add and lose monomers from both its tips: in fact boxes labelled with L and K can equally break their links or get bound to the free monomer F. Note that the filament cannot close into a ring and break in the middle because the pair of sorts (BB, PB) does not have binding and unbinding capabilities.

Configurations of Figure 4.1 can be partitioned in vanishing and tangible. Vanishing configurations, those capable of performing immediate actions, are the majority (fourteen over nineteen). They are uninformative intermediate steps necessary to properly update the sorts of the interfaces when they are involved in binding or unbinding events. Despite being uninformative these configurations must be computed and this considerably slows down the simulation of a system populated with `A` boxes. In the following chapters we show how apply our analysis in order to reduce the number of computed vanishing configurations. The system we take into consideration is $S = (\mathcal{L}(\mathtt{A}_1 \parallel \ldots \parallel \mathtt{A}_n), \emptyset, \emptyset)$, which represents n free and inactive actin monomers floating in the same solution.

Chapter 5

Transition System of the Boxes

In this chapter we focus our attention on boxes. We are interested in the computation of the *transition system of the boxes of the system* S, which we refer as $\Phi^S_{\texttt{Box}}$. The state space of $\Phi^S_{\texttt{Box}}$ is the set of boxes which appear during all the possible evolutions of S. The relation of $\Phi^S_{\texttt{Box}}$ relates two boxes, a condition C, an action a, and a rate r if the first box, in the case C is verified, can perform the action a and transforms with rate r into the second box.

In section 5.1 we define $\Phi^S_{\texttt{Box}}$ by introducing a method for its computation. The state space of $\Phi^S_{\texttt{Box}}$ is obtained by projecting the boxes from the elements of the state space of Φ^S. The relation of $\Phi^S_{\texttt{Box}}$ is computed by observing how a box transforms during the possible evolutions of S.

In section 5.2 we observe that, with the introduced method, we cannot compute $\Phi^S_{\texttt{Box}}$ if Φ^S is hard or impossible to be computed. For this reason we introduce a new procedure which, independently of Φ^S, generates an over-approximation of $\Phi^S_{\texttt{Box}}$, which we refer to as $\tilde{\Phi}^S_{\texttt{Box}}$.

In section 5.3 we investigate the cardinality of $\tilde{\Phi}^S_{\texttt{Box}}/\equiv_b$ and we ensure its finiteness by proposing some constraints to the language syntax.

The chapter concludes by applying the generation of the transition system of the boxes to the actin case study.

5.1 Transition System of the Boxes of a System

We introduce and prove some properties of the transitions involving BlenX systems. The first results are related with the notion of *sent names*. The sent names of a system S are an over-approximation of the channel names sent by boxes during the possible evolutions of S.

Definition 5.1.1 *The set of sent names of a system $S = (B, E, \xi)$ is obtained through the union of the sets $\mathsf{sn}(B) \cup \bigcup_{I[\,P\,]\blacktriangleright_r B' \in E} \mathsf{sn}(B')$, where $\mathsf{sn}(B)$ is defined in Table 5.1. We refer to the set of sent names of S as $\mathsf{sn}(S)$.*

The following lemma describes how the set of sent names of a box changes when the box is involved in a transition.

Lemma 5.1.2 *Let $I_1[\,P_1\,]_\gamma$ be a well-formed box. If $I_1[\,P_1\,]_\gamma \xrightarrow{C,a}_r I_2[\,P_2\,]_\gamma$, then one of the following holds:*

- *if $a \in \{S!m, n!m\}$ then $m \in \mathsf{sn}(I_1[\,P_1\,])$ and $\mathsf{sn}(I_2[\,P_2\,]) \subseteq \mathsf{sn}(I_1[\,P_1\,])$,*

$$
\begin{array}{lcl}
\mathsf{sn}(I[\,P\,]) & = & \mathsf{sn}(P) \setminus \mathsf{sub}(I) \\
\mathsf{sn}(\mathsf{nil}) & = & \emptyset \\
\mathsf{sn}(B_0 \parallel B_1) & = & \mathsf{sn}(B_0) \cup \mathsf{sn}(B_1) \\
\mathsf{sn}(P_0 \mid P_1) & = & \mathsf{sn}(P_0) \cup \mathsf{sn}(P_1) \\
\mathsf{sn}(M_0 + M_1) & = & \mathsf{sn}(M_0) \cup \mathsf{sn}(M_1) \\
\mathsf{sn}(\langle C \rangle M) & = & \mathsf{sn}(M) \\
\mathsf{sn}(*\pi.P) & = & \mathsf{sn}(\pi.P) \\
\mathsf{sn}(\pi.P) & = & \mathsf{sn}(\pi) \cup (\mathsf{sn}(P) \setminus \mathsf{bn}(\pi)) \\
\mathsf{sn}(\pi) & = & \begin{cases} \{m\} & \text{if } \pi = n!m \\ \emptyset & \text{otherwise} \end{cases}
\end{array}
$$

Table 5.1 – Definition of sn for bio-processes. The function goes through the structure of the bio-process collecting the free names sent by output actions.

- *if* $a \in \{S?m, n?m\}$ *then* $\mathsf{sn}(I_2[\,P_2\,]) \subseteq \mathsf{sn}(I_1[\,P_1\,]) \cup \{m\}$,
- *otherwise* $\mathsf{sn}(I_2[\,P_2\,]) \subseteq \mathsf{sn}(I_1[\,P_1\,])$.

Sketch. By induction over the length of the derivation of $I_1[\,P_1\,] \xrightarrow{C,a}_r I_2[\,P_2\,]$.

□

Exploiting the result of the previous lemma, we lift the result from boxes to systems.

Lemma 5.1.3 *Let* S_1 *be a well-formed system. If* $S_1 \xrightarrow{l}_r S_2$*, then one of the following holds:*

- *if* $l = (\gamma, T!n)$ *then* $n \in \mathsf{sn}(S_1)$ *and* $\mathsf{sn}(S_2) \subseteq \mathsf{sn}(S_1)$,
- *if* $l = (\gamma, T?n)$ *then* $\mathsf{sn}(S_2) \subseteq \mathsf{sn}(S_1) \cup \{n\}$,
- *otherwise* $\mathsf{sn}(S_2) \subseteq \mathsf{sn}(S_1)$.

Sketch. By induction on the length of the derivation of $S_1 \xrightarrow{l}_r S_2$ exploiting the result of Lemma 5.1.2.

□

This chain of results regarding the properties of sent names ends with the following corollary which states that a system, during its evolution, cannot generate new sent names.

Corollary 5.1.4 *Let* S *be a well-formed system. If* $S' \in ds(S)$*, then* $\mathsf{sn}(S') \subseteq \mathsf{sn}(S)$.

Sketch. By induction on the number of transitions for obtaining S_1 starting from S and exploiting the result of Lemma 5.1.3.

□

Hereafter we use the function *boxCh*. It identifies boxes which change state in the pair of systems passed as arguments. The function also identifies boxes only present in one of the two systems. It uses the function *get* whose behaviour is described in Definition 3.3.3.

Definition 5.1.5 *The function* $boxCh : \mathcal{S} \times \mathcal{S} \mapsto \mathcal{P}(\{\bot\} \cup \mathcal{B} \times \{\bot\} \cup \mathcal{B})$*, given a pair of systems* S *and* S'*, returns a set* A *such that* $(get(\gamma, S), get(\gamma, S'))$ *belongs to* A *iff* $get(\gamma, S) \neq get(\gamma, S')$*.*

Note that the boxes present in S and not in S' are represented by an element of the form $(I[\,P\,]_\gamma, \bot)$, and those present in S' but not in S with one of the form $(\bot, I[\,P\,]_\gamma)$.

Lemma 5.1.6 shows how boxes change state depending on the kind of transition they are involved in. Point 1 considers transitions caused by the application of an event, a binding or an unbinding. In the case of the application of an event, a box is replaced by boxes with fresh labels. Therefore no boxes change state but some of them are removed and some inserted. This implies that the elements of $boxCh(S, S')$ have form $(\bot, I[\,P\,]_\gamma)$ or $(I[\,P\,]_\gamma, \bot)$. In the case of binding or unbinding, boxes remain unchanged, therefore $boxCh(S, S') = \emptyset$.

Point 2 states that if a transition is labelled with τ and involves exactly one box then such a box performs a change action or an intra-communication with the same rate r of the transition. Moreover, this action is associated with a condition evaluating to *true* with respect to the label of the involved box and the environment of the system.

Point 3 tells that if a transition changes the state of two boxes they are involved in an inter-communication. This implies that they perform complementary input and output actions over interfaces with sorts having positive communication capability. More precisely, the communication capability is exactly r, the rate at which the transition takes place. Also in this case the actions are associated with conditions which evaluate to true with respect to the labels of the involved boxes and the environment of S.

Point 4 considers the case of transitions caused by outputs over an interface. The label of the transition stores the label γ of the box involved in the action, the sort T of the interface involved in the output and the sent channel name m. Note that m belongs to the sent names of S. Point 5 contemplates the case of transitions caused by an input over an interface.

Finally, point 6 considers transitions highlighting the capability of the box γ to perform a binding or an unbinding on the interface with sort T. No boxes change state, thus $boxCh(S, S') = \emptyset$.

Lemma 5.1.6 *Let* $S = (B, E, \xi)$ *be a well formed system. If* $S \xrightarrow{l}_r S'$*, then one of the following holds:*

1. $l = \tau$ *and* $\nexists (I[\,P\,]_\gamma, I'[\,P'\,]_\gamma) : (I[\,P\,]_\gamma, I'[\,P'\,]_\gamma) \in boxCh(S, S')$*.*

2. $l = \tau$*,* $boxCh(S, S') = \{(I[\,P\,]_\gamma, I'[\,P'\,]_\gamma)\}$ *and* $I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma$ *with* $a \in \{\tau, (T, U)\}$ *and* $\wr C \int_{\gamma,\xi} = \mathsf{true}$*.*

3. $l = \tau$*,* $boxCh(S, S') = \{(I_1[\,P_1\,]_{\gamma_1}, I'_1[\,P'_1\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I'_2[\,P'_2\,]_{\gamma_2})\}$ *and* $I_i[\,P_i\,]_{\gamma_i} \xrightarrow{C_i,a_i}_0 I'_i[\,P'_i\,]_{\gamma_i}$ *with* $\wr C_i \int_{\gamma,\xi} = \mathsf{true}$ *for* $i \in [1, 2]$ *and* $\{a_1, a_2\} = \{T!n, U?n\}$ *with* $\alpha_c(T, U) = r$ *and* $n \in \mathsf{sn}(S)$*.*

4. $l = (\gamma, T!n)$*,* $boxCh(S, S') = \{(I[\,P\,]_\gamma, I'[\,P'\,]_\gamma)\}$ *and* $I[\,P\,]_\gamma \xrightarrow{C,T!n}_r I'[\,P'\,]_\gamma$ *with* $\wr C \int_{\gamma,\xi} = \mathsf{true}$ *and* $n \in \mathsf{sn}(S)$*.*

5. $l = (\gamma, T?n)$, $boxCh(S, S') = \{(I[\,P\,]_\gamma, I'[\,P'\,]_\gamma)\}$ *and* $I[\,P\,]_\gamma \xrightarrow{C,T?n}_r I'[\,P'\,]_\gamma$ *with* $\wr C \int_{\gamma,\xi} = \mathsf{true}$.

6. $l = (\gamma, T)$ *and* $boxCh(S, S') = \emptyset$

Sketch. By induction on the length of the derivation tree of $S \xrightarrow{l}_r S'$ and exploiting the result concerning sent names of Lemma 5.1.2.

□

Here we lift the result of the previous lemma to the relation $\rightarrow$. The following corollary shows that in the case of transitions defined by this relation, if some boxes change state there are exactly one or two.

Corollary 5.1.7 *Let S be a well-formed system. If $S \rightarrow_r S'$ and $\exists \gamma_1 : \bot \neq get(\gamma_1, S) \neq get(\gamma_1, S') \neq \bot$, then the cardinality of $boxCh(S, S')$ is either one or two. More precisely it is equal to $\{(I_1[\,P_1\,]_{\gamma_1}, I'_1[\,P'_1\,]_{\gamma_1})\}$ or $\{(I_1[\,P_1\,]_{\gamma_1}, I'_1[\,P'_1\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I'_2[\,P'_2\,]_{\gamma_2})\}$.*

Proof. The proof is based on the observation that, by definition of $\rightarrow$, if $S \rightarrow_r S'$ then $S \xrightarrow{\tau}_r S'$. Thus, looking among the cases of Lemma 5.1.6, we conclude that $boxCh(S, S')$ either contains exactly one or two elements and that these elements are pairs of boxes. Observing that one of the pairs is $(I_1[\,P_1\,]_{\gamma_1}, I'_1[\,P'_1\,]_{\gamma_1})$, given that if $\exists \gamma_1 : \bot \neq get(\gamma_1, S) \neq get(\gamma_1, S') \neq \bot$ then, by definition of $boxCh$, it holds that $(I_1[\,P_1\,]_{\gamma_1}, I'_1[\,P'_1\,]_{\gamma_1}) \in boxCh(S, S')$.

□

With Corollary 5.1.8, we show that, if two systems S and S' are involved in a transition and they differ because of one box, this box is involved in a τ-transition or a change action.

Corollary 5.1.8 *Given $S = (B, E, \xi)$ a well-formed system, if $boxCh(S, S') = \{(I[\,P\,]_\gamma, I'[\,P'\,]_\gamma)\}$ and $S \rightarrow_r S'$ then $I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma$ with $a \in \{\tau, (T, U)\}$ and $\wr C \int_{\gamma,\xi} = \mathsf{true}$.*

Proof. Observing that, by definition of $\rightarrow$, if $S \rightarrow_r S'$ then $S \xrightarrow{\tau}_r S'$, we conclude by Lemma 5.1.6.

□

Corollary 5.1.9 states that if the system S transforms into S', and they differ because of two boxes, these boxes are involved in the same inter-communication.

Corollary 5.1.9 *Let $S = (B, E, \xi)$ if $boxCh(S, S') = \{(I_1[\,P_1\,]_{\gamma_1}, I'_1[\,P'_1\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I'_2[\,P'_2\,]_{\gamma_2})\}$ and $S \rightarrow_r S'$ then $I_i[\,P_i\,]_{\gamma_i} \xrightarrow{C_i,a_i}_0 I'_i[\,P'_i\,]_{\gamma_i}$ with $\wr C_i \int_{\gamma,\xi} = \mathsf{true}$ for $i \in [1, 2]$ and $\{a_1, a_2\} = \{T!n, U?n\}$ with $\alpha_c(T, U) = r$ and $n \in \mathsf{sn}(S)$.*

Proof. Observing that, by definition of $\rightarrow$, if $S \rightarrow_r S'$ then $S \xrightarrow{\tau}_r S'$, we conclude by Lemma 5.1.6.

□

We now define two relations that given a system S_o and a label γ, project all the transitions performed by box labelled with γ during the possible evolutions of S_o. The first one is $\xrightarrow[1,\mathtt{Box}]{}^{\gamma,\,S}$

and represents the transitions performed independently by the box labelled with γ, without synchronizing with other boxes. The second is $\xrightarrow[2,\texttt{Box}]{}^{\gamma,S}$ and represents those transitions that, in order to take place, requires the box labelled with γ to synchronize with another one. Each element of the first relation stores the following information: the two states $I[\,P\,]$ and $I'[\,P'\,]$ of the box involved in the transition, the binding condition C that must evaluate to true in order to perform the transition, the action $a \in \{\tau, (T,U)\}$ that causes the transition and a *simplified rate* r_s. If the transition is immediate then r_s is ∞ otherwise it is $\not\infty$.

Definition 5.1.10 *Given a well-formed system S_o, the relation $\xrightarrow[1,\texttt{Box}]{}^{\gamma,S_o} \subseteq \texttt{Box} \times \{\infty, \not\infty\} \times Cond \times Act_i \times \texttt{Box}$ is defined as the set*

$$\begin{aligned}\{(I[\,P\,], r_s, C, a, I'[\,P'\,]) \;:\; & (B,E,\xi) = S \in ds(S_o) \wedge \\ & S \rightarrow_r S' \wedge \\ & boxCh(S,S') = \{(I[\,P\,]_\gamma, I'[\,P'\,]_\gamma)\} \wedge \\ & I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma \text{ with } \wr\, C \int_{\gamma,\xi} = \mathsf{true} \wedge \\ & a \in \{\tau, (T,U)\} \wedge \\ & r_s = \infty \text{ if } r = \infty \text{ and } r_s = \not\infty \text{ if } r \in \mathbb{R}\}\end{aligned}$$

If $(I[\,P\,], r_s, C, a, I'[\,P'\,]) \in \xrightarrow[1,\texttt{Box}]{}^{\gamma,S}$ we write $I[\,P\,] \xrightarrow[1,\texttt{Box}]{C,a}{}^{\gamma,S}_{r_s} I'[\,P'\,]$.

At this point we prove that $\xrightarrow[1,\texttt{Box}]{}^{\gamma,S}$ captures all the transitions that the box labelled with γ can independently perform.

Lemma 5.1.11 *Let S_o be a well-formd system. If $S \in ds(S_0)$ and $S \rightarrow_r S'$ and $boxCh(S,S') = \{(I[\,P\,]_\gamma, I'[\,P'\,]_\gamma)\}$ then $I[\,P\,] \xrightarrow[1,\texttt{Box}]{C,a}{}^{\gamma,S_o}_{r_s} I'[\,P'\,]$ with $r_s = \infty$ if $r = \infty$ and $r_s = \not\infty$ if $r \in \mathbb{R}$.*

Sketch. We can immediately conclude by Corollary 5.1.8 and by definition of $\xrightarrow[1,\texttt{Box}]{}^{\gamma,S_o}$. □

The elements of $\xrightarrow[2,\texttt{Box}]{}^{\gamma,S}$ store the same information as those of $\xrightarrow[1,\texttt{Box}]{}^{\gamma,S}$. In this case $a \in \{T!n, T?n\}$ and the simplified rate is always "?". It means that we do not know the actual rate of the transition; it depends on the interacting partner.

Definition 5.1.12 *Given a well-formed system S_o, the relation $\xrightarrow[2,\texttt{Box}]{}^{\gamma,S_o} \subseteq \texttt{Box} \times \{?\} \times Cond \times Act_i \times \texttt{Box}$ is defined as the set*

$$\begin{aligned}\{(I_1[\,P_1\,], ?, C_1, a_1, I_1'[\,P_1'\,]) \;:\; & (B,E,\xi) = S \in ds(S_o) \wedge \\ & S \rightarrow_r S' \wedge \\ & boxCh(S,S') = \{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I_2'[\,P_2'\,]_{\gamma_2})\} \wedge \\ & \gamma = \gamma_1 \wedge \\ & I_i[\,P_i\,]_{\gamma_i} \xrightarrow{C_i,a_i}_0 I_i'[\,P_i'\,]_{\gamma_i} \text{ with } \wr\, C_i \int_{\gamma_i,\xi} = \mathsf{true} \wedge \\ & \{a_1, a_2\} = \{T!n, U?n\} \text{ with } \alpha_c(T,U) \neq 0\}\end{aligned}$$

If $(I[\,P\,], r_s, C, a, I'[\,P'\,]) \in \xrightarrow[2,\texttt{Box}]{}^{\gamma,S}$ we write $I[\,P\,] \xrightarrow[2,\texttt{Box}]{C,a}{}^{\gamma,S}_{r_s} I'[\,P'\,]$.

Thanks to Lemma 5.1.13, we prove that this relation captures all the transitions that the box labelled with γ can perform synchronizing with other boxes.

Lemma 5.1.13 *Let S_o be a well-formed system. If $S \in ds(S_0)$ and $S \rightarrow_r S'$ and $boxCh(S, S')$ is equal to the set $\{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I_2'[\,P_2'\,]_{\gamma_2})\}$ then $I_i[\,P_i\,]_{\gamma_i} \xrightarrow[2,\texttt{Box}]{C_i, a_i}{}^{\gamma_i, S_o}_{?} I'[\,P'\,]$ for $i \in [1,2]$. Moreover it holds that $\{a_1, a_2\} = \{T!n, U?n\}$ and $\alpha_c(T, U) \neq 0$.*

Sketch. We can immediately conclude by Corollary 5.1.9 and by definition of $\xrightarrow[2,\texttt{Box}]{}^{\gamma, S_o}$.

□

Through the union of $\xrightarrow[1,\texttt{Box}]{}^{\gamma, S_o}$ and $\xrightarrow[2,\texttt{Box}]{}^{\gamma, S_o}$ we obtain a relation that, given a system S_o and a label γ, projects all the transitions performed by the box labelled with γ during the possible evolutions of S_o.

Definition 5.1.14 *Given a well-formed system S_o, the relation $\xrightarrow[\texttt{Box}]{}^{\gamma, S_o} \subseteq \texttt{Box} \times \{\infty, \not\infty, ?\} \times Cond \times Act_i \times \texttt{Box}$ is defined as the union of $\xrightarrow[1,\texttt{Box}]{}^{\gamma, S_o}$ and $\xrightarrow[2,\texttt{Box}]{}^{\gamma, S_o}$. If $(I[\,P\,], r_s, C, a, I'[\,P'\,]) \in \xrightarrow[\texttt{Box}]{}^{\gamma, S}$ we write $I[\,P\,] \xrightarrow[\texttt{Box}]{C, a}{}^{\gamma, S}_{r_s} I'[\,P'\,]$.*

In the following lemma we prove that $\xrightarrow[\texttt{Box}]{}^{\gamma, S_o}$ represents all the transitions that the box labelled with γ performs during the possible evolutions of S_o.

Lemma 5.1.15 *Let S_o be a well-formed system. If $S \in ds(S_0)$, $S \rightarrow_r S'$, and $(I_1[\,P_1\,]_{\gamma_1}, I'[\,P'\,]_{\gamma}) \in boxCh(S, S')$ then one of the followings holds:*

1. *$boxCh(S, S') = \{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1})\}$ and $I_1[\,P_1\,] \xrightarrow[\texttt{Box}]{C, a}{}^{\gamma_1, S_o}_{r_s} I_1'[\,P_1'\,]$ with $r_s = \infty$ if $r = \infty$ and $r_s = \not\infty$ if $r \in \mathbb{R}$.*
2. *$boxCh(S, S') = \{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I_2'[\,P_2'\,]_{\gamma_2})\}$ and $I_i[\,P_i\,]_{\gamma_i} \xrightarrow[\texttt{Box}]{C_i, a_i}{}^{\gamma_i, S_o}_{?} I_i'[\,P_i'\,]$ for $i \in [1,2]$ with $\{a_1, a_2\} = \{T!n, U?n\}$ and $\alpha_c(T, U) \neq 0$.*

Proof. By Corollary 5.1.7 it follows that $boxCh(S, S')$ is either equal to $\{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1})\}$ or to $\{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I_2'[\,P_2'\,]_{\gamma_2})\}$. In the first case we can conclude by Lemma 5.1.11, in the second by Lemma 5.1.13.

□

Now we introduce the set collecting all the states assumable by the box labelled with γ during the possible evolutions of a system S_o.

Definition 5.1.16 *The derivatives of the box labelled with γ with respect to a system S_o is defined as*

$$ds(\gamma, S_o) = \{I[\,P\,] : \exists S \in ds(S_o) \wedge get(\gamma, S) = I[\,P\,]_{\gamma}\}$$

The set $\Phi^{\gamma, S_o}_{\texttt{Box}}$ and the relation $\xrightarrow[\texttt{Box}]{}^{\gamma, S_o}$ define a transition system describing how the box labelled with γ evolves inside the system S_o.

Definition 5.1.17 *The transition system $\Phi_{\mathtt{Box}}^{\gamma,S_o}$ generated by the box labelled with γ during the evolution of the system S_o is defined as*

$$\Phi_{\mathtt{Box}}^{\gamma,S_o} = (ds(\gamma, S_o), \xrightarrow[\mathtt{Box}]{}^{\gamma,S_o})$$

Generating the object $\Phi_{\mathtt{Box}}^{\gamma,S_o}$ for each γ appearing in S_o, we have the information to define a transition system describing how the boxes of S_o can evolve.

Definition 5.1.18 *The transition system of the boxes of the system S_o is defined as*

$$\Phi_{\mathtt{Box}}^{S_o} = \bigcup_{\gamma \in Label} \Phi_{\mathtt{Box}}^{\gamma,S_o}$$

In the following theorem we prove that the state space of $\Phi_{\mathtt{Box}}^{S_o}$ contains all the boxes which can appear in the derivatives of S_o. Moreover we show that if a box can perform a state change in the context of a derivative of S_o, it can do the same in $\Phi_{\mathtt{Box}}^{S_o}$.

Theorem 5.1.19 *Let S_o be a well-formed system. Given $\Phi_{\mathtt{Box}}^{S_o} = (A, R)$, $\forall S \in ds(S_o)$:*

1. *if $get(\gamma, S) = I[\,P\,]_\gamma$ then $I[\,P\,]_\gamma \in A$.*
2. *if $S \to_r S'$ and $(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1}) \in boxCh(S, S')$ then one of the followings holds:*
   (a) *$boxCh(S, S') = \{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1})\}$ and $(I_1[\,P_1\,], C, a, r_s, I_1'[\,P_1'\,]) \in R$ with $r_s = \infty$ if $r = \infty$ and $r_s = \not\infty$ if $r \in \mathbb{R}$.*
   (b) *$boxCh(S, S') = \{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I_2'[\,P_2'\,]_{\gamma_2})\}$, $(I_i[\,P_i\,], C, a, ?, I_i'[\,P_i'\,]) \in R$ with $\{a_1, a_2\} = \{T!n, U?n\}$ and $\alpha_c(T, U) \neq 0$.*

Proof. We immediately prove point 1 by observing that $A = \bigcup_{\gamma \in Label}\{I[\,P\,] : \exists S \in ds(S_o)$ and $get(\gamma, S) = I[\,P\,]_\gamma\}$. As far as point 2 is concerned, we refer to Lemma 5.1.15 and we get the desired result observing that $\xrightarrow[\mathtt{Box}]{}^{\gamma_i,S_o} \subseteq R$ for $i \in [1, 2]$.

□

5.2 Over-approximating the transition system of the Boxes

In this section we introduce an over-approximation of $\Phi_{\mathtt{Box}}^{S}$. Its definition requires some preliminary notions. Among them we have $\Phi_{\mathtt{Box}}$, a transition system describing the state changes BlenX boxes can potentially perform, without regard to the system S in which they are confined.

Definition 5.2.1 *The transition system of the boxes is defined as*

$$\Phi_{\mathtt{Box}} = (\mathtt{Box}, \xrightarrow[\mathtt{Box}]{})$$

where $\xrightarrow[\mathtt{Box}]{} \subseteq \mathtt{Box} \times \{\infty, \not\infty, ?\} \times Cond \times Act_i \times \mathtt{Box}$ is the largest relation satisfying the followings:

1. *If $(I[\,P\,], r_s, C, a, I'[\,P'\,]) \in \xrightarrow[\mathtt{Box}]{}$ then $I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$, $a \notin \{n!t, n?t\}$ and:*

(a) *if* $r = \infty$ *then* $r_s = \infty$.

(b) *if* $r \in \mathbb{R}$ *and* $a \notin \{T!n, T?n\}$ *then* $r_s = \not\infty$.

(c) *if* $a \in \{T!n, T?n\}$ *then* $r = ?$.

2. *Let* $I[\,P\,]$ *be a well-formed box. If* $I[\,P\,] \xrightarrow{\mathsf{true},a}_{\infty} I'[\,P'\,]$ *then* $\nexists I''[\,P''\,] : (I[\,P\,], \not\infty, C, a, I''[\,P''\,]) \in \xrightarrow[\texttt{Box}]{}$.

The definition of $\xrightarrow[\texttt{Box}]{}$ is based on the rules of Table 3.4. For each transition the elements of the relation store: the involved boxes, the associated condition C, the action causing the transition and its simplified rate. The relation $\xrightarrow[\texttt{Box}]{}$ only stores element with condition different from false. In this context we cannot require the evaluation of C to be true because we miss the information about the binding state of the box interfaces, therefore C is not always evaluable. We take into account the priority mechanism of BlenX that prevents the boxes involved in immediate actions performing transitions associated with simplified rate $\not\infty$. Observe that for each well-formed system S, the transition system $\Phi_{\texttt{Box}}$ is a valid over-approximation of $\Phi^S_{\texttt{Box}}$. We can now cut elements from $\Phi_{\texttt{Box}}$ in order to make it specific to a given system S. We obtain this result by acting on the channel names received by input actions. In the transition system $\Phi_{\texttt{Box}}$, input over box interfaces can receive any channel name. Now we introduce some notions in order to define a transition system which allows boxes to only receive names sent in the BlenX system they are part of.

The set of *proper derivatives of* $I_1[\,P_1\,]$ *with respect to the system* S is made of the box configurations we can reach through chains of transitions which start from $I_1[\,P_1\,]$. When a box performs an input action we require the received channel name to belong to $\mathsf{sn}(S)$.

Definition 5.2.2 *Let* $I_1[\,P_1\,]$ *and* $I_2[\,P_2\,]$ *be well formed boxes.* $I_2[\,P_2\,]$ *is a (one-step) proper derivative of* $I_1[\,P_1\,]$ *with respect to the system* S *if it holds that* $I_1[\,P_1\,] \xrightarrow[\texttt{Box}]{C,a}_r I_2[\,P_2\,]$ *and* $(a = T?m)$ *implies* $(m \in \mathsf{sn}(S))$. *More generally if*

$$I_1[\,P_1\,] \xrightarrow[\texttt{Box}]{C_1,a_1}_{r_1} \cdots \xrightarrow[\texttt{Box}]{C_{n-1},a_{n-1}}_{r_{n-1}} I_n[\,P_n\,]$$

and $(a_i = T?m)$ *implies* $(m \in \mathsf{sn}(S))$ *for* $i \in [1, n)$ *then* $I_n[\,P_n\,]$ *is a proper derivative of* $I_1[\,P_1\,]$ *with respect to the system* S. *Finally, the set* $pds(I[\,P\,], S)$ *of the proper derivatives of* $I[\,P\,]$ *with respect to the system* S *is defined as the set containing* $I[\,P\,]$ *and all the boxes* $I'[\,P'\,]$ *such that* $I'[\,P'\,]$ *is a proper derivative of* $I[\,P\,]$ *with respect to the system* S.

The set of *derivatives of* $I[\,P\,]$ *with respect to* S is obtained closing the set $pds(I[\,P\,], S)$ by α-equivalence.

Definition 5.2.3 *Let* $I''[\,P''\,]$ *and* $I[\,P\,]$ *be well-formed boxes.* $I''[\,P''\,]$ *is a derivative of* $I[\,P\,]$ *with respect to* S *if exists* $I'[\,P'\,] \in pds(I[\,P\,], S)$ *such that* $I'[\,P'\,] = I''[\,P''\,]$. *The set* $ds(I[\,P\,], S)$ *of the derivatives of* $I[\,P\,]$ *respect to the system* S *is defined as the set that contains all the* $I'[\,P'\,]$ *such that* $I'[\,P'\,] = I[\,P\,]$ *and the derivatives of* $I[\,P\,]$ *respect to the system* S.

Here we define the relation $\xrightarrow[\texttt{Box}]{}^{I[\,P\,],S}$ which, exploiting the set $ds(I[\,P\,], S)$, cuts from the relation $\xrightarrow[\texttt{Box}]{}$ some transitions that $I[\,P\,]$ cannot perform inside the system S.

Definition 5.2.4 *Let $I[\,P\,]$ be a well-formed box, and S a well-formed system. The relation $\xrightarrow[\texttt{Box}]{}^{I[\,P\,],S}$ is the largest subset of $\xrightarrow[\texttt{Box}]{}$ such that if the element $(I_1[\,P_1\,], r_s, C, a, I_2[\,P_2\,])$ belongs to $\xrightarrow[\texttt{Box}]{}^{I[\,P\,],S}$ then $I_1[\,P_1\,], I_2[\,P_2\,]$ belong to $ds(I[\,P\,], S)$.*

Finally we generate the transition system describing the over-approximation of the possible evolutions of a box $I[\,P\,]$ in the context of the system S.

Definition 5.2.5 *Let $I[\,P\,]$ be a box. The over-approximation of the transitions that $I[\,P\,]$ can perform with respect to the system S is the transition system*

$$\tilde{\Phi}^{I[\,P\,],S}_{\texttt{Box}} = (ds(I[\,P\,], S), \xrightarrow[\texttt{Box}]{}^{I[\,P\,],S})$$

Now we exploit $\tilde{\Phi}^{I[\,P\,],S}_{\texttt{Box}}$ in order to obtain an over-approximation of $\Phi^S_{\texttt{Box}}$. For each box of $I[\,P\,]$ appearing in the bio-process of S we compute $\tilde{\Phi}^{I[\,P\,],S}_{\texttt{Box}}$. These sets contain all the transitions that the boxes of S can perform with the exception of those caused by the application of events. In fact we build $\tilde{\Phi}^{I[\,P\,],S}_{\texttt{Box}}$ through the rules of Table 3.4 which do not contemplate application of events to boxes. In order to add the missing boxes with the related transitions, given $S = (B, \{I_1[\,P_1\,] \blacktriangleright_{r_1} B_1, \ldots, I_n[\,P_n\,] \blacktriangleright_{r_n} B_n\}, \xi)$, we extend the computation of $\tilde{\Phi}^{I[\,P\,],S}_{\texttt{Box}}$ to each box $I[\,P\,]$ appearing in $B_1, \ldots, B_n$. We now define the function *getBox*, which extracts from S the boxes whose transition system is necessary in order to compute the over-approximation of $\Phi^S_{\texttt{Box}}$.

Definition 5.2.6 *The function $getBox : \mathcal{B} \to \mathcal{P}(\mathcal{B})$, given a bio-process, returns the set of boxes which appears in its definition. Given a system $S = (B, E, \xi)$ we lift this function to systems as follows:*

$$getBox(S) = getBox(B) \cup \bigcup_{I[\,P\,] \blacktriangleright_r B' \in E} getBox(B')$$

Now we can define the over-approximation of $\Phi^S_{\texttt{Box}}$ as follows.

Definition 5.2.7 *The over-approximated transition system of the boxes of a well-formed system S is*

$$\tilde{\Phi}^S_{\texttt{Box}} = \bigcup_{I[\,P\,] \in getBox(S)} \tilde{\Phi}^{I[\,P\,],S}_{\texttt{Box}}$$

We now present some results necessary to prove that $\tilde{\Phi}^S_{\texttt{Box}}$ is a valid over-approximation of $\Phi^S_{\texttt{Box}}$. We start proving some properties regarding derivation trees generated by the BlenX semantics. In the following lemma we show that if a box performs an action with rate different from zero, then this action is neither an input nor an output (note that it can be a synchronization).

Lemma 5.2.8 *Let $I[\,P\,]$ be a well-formed box. If $I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma$ and $r \neq 0$ then $a \notin \{k!t, k?t\}$.*

Sketch. By induction on the length of the derivation tree of $I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma$ (which is obtained applying rules of Table 3.4).

□

Here we show that if a box can perform an immediate action associated with condition true then all the systems $S = (B, E, \xi)$, such that this box appears in B, are vanishing.

Lemma 5.2.9 *Let $I[\,P\,]$ be a well-formed box. If $I[\,P\,]_\gamma \xrightarrow{\mathsf{true},a}_\infty I'[\,P'\,]_\gamma$ then $\forall S \in \mathcal{S}$, if exists $\gamma' \in$ Label s.t. $get(\gamma, S) = I[\,P\,]_{\gamma'}$ then $S \rightarrow_\infty S'$ for some $S' \in \mathcal{S}$.*

Proof. By Lemma 5.2.8 $a \notin \{k!t, k?t\}$. Thus, depending on the action a, we can apply either rules r11 or rule r12 (see Table 3.6) and obtain $(I_1[\,P_1\,]_\gamma, E, \xi) \xrightarrow{\tau}_r (I_2[\,P_2\,]_\gamma, E, \xi')$. Successively we can add context to $I_1[\,P_1\,]_\gamma$ and $I_2[\,P_2\,]_\gamma$ with multiple applications of rules r18-l and r18-r (see Table 3.8). Note that we do not impose any constraints on E, ξ, γ, and the context added to $I_1[\,P_1\,]_\gamma$, therefore we can deduce an immediate τ-transition for whatever system containing the box $I_1[\,P_1\,]$, and the Lemma follows.

□

In the next lemma we show that if S transforms into S', all the boxes of S' either appear in S or are the result of a transition having as origin a box of S.

Lemma 5.2.10 *Let S be a well-formed system and γ a box label. If $S \xrightarrow{l}_r S'$ and $get(\gamma, S') = I'[\,P'\,]_\gamma$ then one of the following holds:*

1. *if $l = (\gamma', T!n)$ then*
   (a) *if $\gamma \neq \gamma'$ then $I'[\,P'\,] \in getBox(S)$*
   (b) *if $\gamma = \gamma'$ then $get(\gamma, S) = I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow{C,T!n}_0 I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$ and $n \in \mathsf{sn}(S)$.*
2. *if $l = (\gamma', T?n)$ then*
   (a) *if $\gamma \neq \gamma'$ then $I'[\,P'\,] \in getBox(S)$*
   (b) *if $\gamma = \gamma'$ then $get(\gamma, S) = I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow{C,T?n}_0 I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$.*
3. *if $l = \tau$ then*
   (a) *$I'[\,P'\,] \in getBox(S)$ or*
   (b) *$get(\gamma, S) = I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow{C,a}_{r'} I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$, $a \notin \{n?t, n!t\}$, $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{sn}(S) \wedge r' = 0)$ and $(a \notin \{T!n, T?n\} \Rightarrow r' = r)$.*
4. *if $l = (\gamma', T)$ then $I'[\,P'\,] \in getBox(S)$.*

Sketch. By induction on the length of the derivation tree of $S \xrightarrow{l}_r S'$ and exploiting the result concerning the sent names of Lemma 5.1.3.

□

Here we lift the result of the previous lemma to the multi-relation $\rightarrow$.

Lemma 5.2.11 *Let S be a well-formed system and γ a box label. If $S \rightarrow_r S'$ and $get(\gamma, S') = I'[\,P'\,]_\gamma$ then one of the following holds:*

1. $I'[\,P'\,] \in getBox(S)$.
2. $get(\gamma, S) = I[\,P\,]_\gamma$ *and* $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_{r_s} I'[\,P'\,]$ *and* $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{sn}(S))$.

Proof. By definition of $\rightarrow$, $S \rightarrow_r S'$ implies that $S \xrightarrow{\tau}_r S'$ which, by Lemma 5.2.10, implies one of the following:

1. $I'[\,P'\,] \in getBox(S)$ which proves the result;
2. $get(\gamma, S) = I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow{C,a}_{r'} I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$ and $a \notin \{n?t, n!t\}$ and $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{sn}(S) \wedge r' = 0)$ and $(a \notin \{T!n, T?n\} \Rightarrow r' = r)$.

In the second case we have two sub-cases:

$r = \infty$: by definition of $\xrightarrow[\texttt{Box}]{}$ we can conclude:

1. If $a \in \{T!n, T?n\}$ then $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_? I'[\,P'\,]$ with $n \in \mathsf{sn}(S)$.
2. If $a \notin \{T!n, T?n\}$ then $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_\infty I'[\,P'\,]$.

$r \in \mathbb{R}$: if $a \in \{T!n, T?n\}$ we can conclude as in the case $r = \infty$.

If $a \notin \{T!n, T?n\}$ we have to verify that $\nexists I''[\,P''\,] : I[\,P\,]_\gamma \xrightarrow{\mathsf{true},a'}_\infty I''[\,P''\,]_\gamma$. In fact such a transition, by definition of $\xrightarrow[\texttt{Box}]{}$, would make the existence of $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_\infty I'[\,P'\,]$ impossible.

We prove this assuming by contradiction that the transition $I[\,P\,]_\gamma \xrightarrow{\mathsf{true},a'}_\infty I''[\,P''\,]_\gamma$ exists. This, by Lemma 5.2.9, implies that $\exists S'' : S \rightarrow_\infty S''$ which is logically absurd because, by definition of $\rightarrow$, the existence of $S \rightarrow_r S'$ would be impossible. Therefore, given that $\nexists I''[\,P''\,] : I[\,P\,]_\gamma \xrightarrow{\mathsf{true},a'}_\infty I''[\,P''\,]_\gamma$, we can conclude with $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_\infty I'[\,P'\,]$.

□

Here we state that all the boxes of the derivatives of a system S are derivative of a box belonging to S.

Lemma 5.2.12 *Let S be a well-formed system and γ a box label. If $S' \in ds(S)$ and $get(\gamma, S') = I'[\,P'\,]_\gamma$ then $\exists I[\,P\,] \in getBox(S) : I'[\,P'\,] \in ds(I[\,P\,], S)$.*

Proof. By induction on the number of transitions necessary to obtain S' starting from S.

base case: in this case $S' = S$, therefore if $get(\gamma, S) = I'[\,P'\,]_\gamma$ then $I'[\,P'\,] \in getBox(S)$ by definition of the function *getBox*.

step case: in this case $S \rightarrow_{r_1} \ldots \rightarrow_{r_n} S_n \rightarrow_{r_{n+1}} S'$. If $S_n \rightarrow_{r_{n+1}} S'$ then by Lemma 5.2.11 we have two cases:

1. $I'[\,P'\,] \in \mathit{getBox}(S_n)$: by definition of getBox either $\exists \gamma' : \mathit{get}(\gamma', S_n) = I'[\,P'\,]_\gamma$ or, given $S_n = (B_n, E, \xi_n)$, that $\exists I[\,P\,] \blacktriangleright_r B \in E : I'[\,P'\,] \in \mathit{getBox}(B)$. In the first case we immediately conclude by induction hypothesis. In the second case we note that transitions between systems do not affect the set of events. This implies that S is associated with the same event set E of S' and thus we can conclude that $I'[\,P'\,]$ also belongs to $\mathit{getBox}(S)$ which lets us conclude with the desired result given that $I'[\,P'\,] \in \mathit{getBox}(S)$ and thus $I'[\,P'\,] \in \mathit{ds}(I'[\,P'\,], S)$.
2. $\mathit{get}(\gamma, S_n) = I_n[\,P_n\,]_\gamma$ and $I_n[\,P_n\,] \xrightarrow[\mathtt{Box}]{C,a}_r I'[\,P'\,]$ and $(a \in \{T!m, T?m\} \Rightarrow m \in \mathsf{sn}(S_n))$, and by Corollary 5.1.4 implies $(a \in \{T!m, T?m\} \Rightarrow m \in \mathsf{sn}(S))$: in this case, by inductive hypothesis, $I_n[\,P_n\,] \in \mathit{ds}(I[\,P\,], S)$. Considering that $I_n[\,P_n\,] \xrightarrow[\mathtt{Box}]{C,a}_r I'[\,P'\,]$ and $(a \in \{T!m, T?m\} \Rightarrow m \in \mathsf{sn}(S))$ we can immediately conclude by definition of the derivatives of $I[\,P\,]$ that $I'[\,P'\,]$ also belongs to $\mathit{ds}(I[\,P\,], S)$.

□

Lemma 5.2.13 states that if we project a box transition from $\Phi^{S_o}_{\mathtt{Box}}$, the same transition can be deduced with the rules of Table 3.4. Moreover, if the box receives or sends a name, this name belongs to $\mathsf{sn}(S_o)$.

Lemma 5.2.13 *Let S_o be a well-formed system and $I[\,P\,]$ a well-formed box. If $I[\,P\,] \xrightarrow[\mathtt{Box}]{C,a}{}^{\gamma, S_o}_{r_s} I'[\,P'\,]$ then $I[\,P\,] \xrightarrow[\mathtt{Box}]{C,a}_{r_s} I'[\,P'\,]$ and $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{sn}(S_o))$.*

Proof. By definition of $\xrightarrow[\mathtt{Box}]{}^{\gamma, S_o}$ one of the following holds:

$(I[\,P\,], r_s, C, a, I'[\,P'\,]) \in \xrightarrow[1,\mathtt{Box}]{}^{\gamma, S_o}$**:** by definition of $\xrightarrow[1,\mathtt{Box}]{}^{\gamma, S_o}$ we can easily deduce that:

$$\begin{array}{rcl} \exists S \in \mathit{ds}(S_o) & : & S \to_r S' \\ & & I[\,P\,] \xrightarrow{C,a}_r I'[\,P'\,] \\ & & \wr C \int^*_I \neq \mathsf{false} \text{ and} \\ & & a \in \{\tau, (T, U)\} \\ & & \text{if } r_s = \infty \text{ then } r = \infty, \text{ if } r_s = \not\infty \text{ then } r \in \mathbb{R} \end{array}$$

If $r_s = \infty$ we have what is necessary to conclude $I[\,P\,] \xrightarrow[\mathtt{Box}]{C,a}_\infty I'[\,P'\,]$. If $r \in \mathbb{R}$ we have to ensure that $\nexists I''[\,P''\,] : I[\,P\,]_\gamma \xrightarrow{\mathsf{true},a'}_\infty I''[\,P''\,]_\gamma$. In fact such a transition, by definition of $\xrightarrow[\mathtt{Box}]{}$, would make the existence of $I[\,P\,] \xrightarrow[\mathtt{Box}]{C,a}_{\not\infty} I'[\,P'\,]$ impossible. We prove this assuming by contradiction that the transition $I[\,P\,]_\gamma \xrightarrow{\mathsf{true},a'}_\infty I''[\,P''\,]_\gamma$ exists. This by Lemma 5.2.9 implies that $\exists S'' : S \to_\infty S''$ but this is logically absurd because, by definition of $\to$, the existence of $S \to_r S'$ would be impossible. Therefore, given that $\nexists I''[\,P''\,] : I[\,P\,]_\gamma \xrightarrow{\mathsf{true},a'}_\infty I''[\,P''\,]_\gamma$ we can conclude with $I[\,P\,] \xrightarrow[\mathtt{Box}]{C,a}_{\not\infty} I'[\,P'\,]$.

$(I[\,P\,], r_s, C, a, I'[\,P'\,]) \in \xrightarrow[2,\mathtt{Box}]{}^{\gamma, S_o}$**:** by definition of $\xrightarrow[2,\mathtt{Box}]{}^{\gamma, S_o}$, we can easily deduce that $r_s = ?$ and

$$\begin{array}{lll} \exists S \in ds(S_o) & : & S \rightarrow_r S' \\ & & boxCh(S,S') = \{(I_1[\,P_1\,]_{\gamma_1}, I_1'[\,P_1'\,]_{\gamma_1}), (I_2[\,P_2\,]_{\gamma_2}, I_2'[\,P_2'\,]_{\gamma_2})\} \\ & & (I[\,P\,]_\gamma, I'[\,P'\,]_\gamma) \in boxCh(S,S') \\ & & I[\,P\,] \xrightarrow{C,a}_0 I'[\,P'\,] \\ & & \wr C \int_I^* \neq \mathsf{false} \text{ and} \\ & & a \in \{T!n, T?n\} \end{array}$$

These observations are enough to exploit the result of Corollary 5.1.9 and obtain that $n \in \mathsf{sn}(S)$ which, given that $S \in ds(S_o)$, by Lemma 5.1.4, implies that $n \in \mathsf{sn}(S_o)$. Now we have all that is necessary to conclude, by definition of $\xrightarrow[\texttt{Box}]{}$, that $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_? I'[\,P'\,]$ with $a \in \{T!n, T?n\}$ and $n \in \mathsf{sn}(S_o)$.

□

Finally, we can show that $\tilde{\Phi}^{S_o}_{\texttt{Box}}$ is an over-approximation of $\Phi^{S_o}_{\texttt{Box}}$.

Theorem 5.2.14 *Given S_o a well-formed system*

$$\Phi^{S_o}_{\texttt{Box}} \subseteq \tilde{\Phi}^{S_o}_{\texttt{Box}}$$

Proof. We need to show the following

1. $I[\,P\,] \in \bigcup_{\gamma \in Label} ds(\gamma, S_o) \Rightarrow I[\,P\,] \in \bigcup_{I[\,P\,] \in getBox(S_o)} ds(I[\,P\,], S)$
2. $$\begin{array}{c} (I[\,P\,], r_s, C, a, I'[\,P'\,]) \in \bigcup_{\gamma \in Label} \xrightarrow[\texttt{Box}]{}^{\gamma, S_o} \Rightarrow \\ (I[\,P\,], r_s, C, a, I'[\,P'\,]) \in \bigcup_{I[\,P\,] \in getBox(S_o)} \xrightarrow[\texttt{Box}]{}^{I[\,P\,], S_o} \end{array}$$

In order to show 1 we prove that:

$$I[\,P\,] \in ds(\gamma, S_o) \Rightarrow \exists I'[\,P'\,] \in getBox(S_o) : I[\,P\,] \in ds(I'[\,P'\,], S_o)$$

This result follows by observing that if $I[\,P\,] \in ds(\gamma, S_o)$ then $\exists S \in ds(S_o) : get(\gamma, S) = I[\,P\,]_\gamma$. This, by Lemma 5.2.12, implies that $\exists I'[\,P'\,] \in getBox(S_o) : I[\,P\,] \in ds(I'[\,P'\,], S_o)$.

In order to show 2 we prove that:

$$I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_r{}^{\gamma, S_o} I'[\,P'\,] \Rightarrow \exists I''[\,P''\,] \in getBox(S_o) : I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_r{}^{I''[\,P''\,], S_o} I'[\,P'\,]$$

$I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_r{}^{\gamma, S_o} I'[\,P'\,]$, by Lemma 5.2.13, implies that $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_r I'[\,P'\,]$ and that $(a \in \{T?n!, T?n?\} \Rightarrow n \in \mathsf{sn}(S_o))$. In order to conclude we need to show that $\exists I''[\,P''\,] \in getBox(S_o) : I[\,P\,] \in ds(I''[\,P''\,], S_o)$ (note that this implies $I'[\,P'\,]$ also belongs to $ds(I''[\,P''\,], S_o)$). We get the result observing that $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_r{}^{\gamma, S_o} I'[\,P'\,]$ implies $I[\,P\,] \in ds(\gamma, S_o)$ and applying the result obtained from point 1. At this point we have all that is necessary to conclude that $\exists I''[\,P''\,] \in getBox(S_o) : I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_r{}^{I''[\,P''\,], S_o} I'[\,P'\,]$.

□

5.3 Investigating the Finiteness of the Over-approximation

The objective of this section is to understand when and how we can give a finite representation of $\tilde{\Phi}^{S_o}_{\mathtt{Box}}$. We note that $\tilde{\Phi}^{S_o}_{\mathtt{Box}}$ is finite if all the boxes which appear in S_0 deadlock after a finite number of steps. If it is not the case, a box can assume infinite configurations and consequently the system it belongs to can do the same. A box with such a property must contain at least one replication and have the capability to spawn it an infinite number of times. Here is an example:

$$\{b, T\}[\, *n?x.\, n!m.\, \mathsf{nil} \mid n!m.\, \mathsf{nil}\,]$$

This box whatever the system in which it appears, never deadlocks. In fact, when the active action $n!m$ interacts with the input operation guarding the replication, we get a new copy of the process $n!m.\, \mathsf{nil}$:

$$\{b, T\}[\, *n?x.\, n!m.\, \mathsf{nil} \mid n!m.\, \mathsf{nil} \mid \mathsf{nil}\,]$$

Therefore, after each communication over the channel name n, an additional *nil* process appears and the box assumes a new configuration. Note that even if we can generate infinite different configurations, all of them are congruent to the initial state of the box. Therefore, in order to get a finite representation of the state space of the box, we can quotient it by congruence. If we lift this operation to $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box}}$ we obtain the following transition system.

Definition 5.3.1 *Let $I[\,P\,]$ be a well-formed box and S_o a well-formed system.*

$$\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv} = (ds(I[\,P\,], S)/{\equiv_b}, \xrightarrow[\mathtt{Box},\equiv]{}^{I[\,P\,],S})$$

where

$$[\![I_1[\,P_1\,]]\!] \xrightarrow[\mathtt{Box},\equiv]{C,a}{}^{I[\,P\,],S_o}_{r_s} [\![I_2[\,P_2\,]]\!] \quad \textit{iff} \quad \begin{array}{l} I'_1[\,P'_1\,] \xrightarrow[\mathtt{Box}]{C,a}{}^{I[\,P\,],S_o}_{r_s} I'_2[\,P'_2\,] \textit{ and} \\ I_1[\,P_1\,] \equiv_b I'_1[\,P'_1\,] \textit{ and} \\ I_2[\,P_2\,] \equiv_b I'_2[\,P'_2\,] \end{array}$$

We now extend this idea to Φ^S and obtain the following:

Definition 5.3.2 *Let S be a well-formed system:*

$$\tilde{\Phi}^{S}_{\mathtt{Box},\equiv} = \bigcup_{I[\,P\,] \in getBox(S)} \tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$$

Summing up, the transition system $\tilde{\Phi}^{S}_{\mathtt{Box},\equiv}$ is a representation of $\tilde{\Phi}^{S}_{\mathtt{Box}}$ which can be finite even if the system S does not deadlock. More precisely, considering that $getBox(S)$ cannot be infinite, the object $\tilde{\Phi}^{S}_{\mathtt{Box},\equiv}$ has finite cardinality if the same holds for each $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ with $I[\,P\,] \in getBox(S)$. We are interested in defining a class of boxes which ensures the finiteness of $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$, in this way we can define systems whose $\tilde{\Phi}^{S}_{\mathtt{Box},\equiv}$ is certainly finite.

In the next lemma we show that $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ is finitely branched.

Lemma 5.3.3 *If $ds(I[\,P\,], S)/{\equiv_b}$ has finite cardinality then $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ is a finite object.*

Proof. The proof comes by observing that, under the assumption that $ds(I[\,P\,],S)/{\equiv_b}$ is finite, the relation $\xrightarrow[\mathtt{Box},\equiv]{}^{I[\,P\,],S}$ cannot have an infinite cardinality. Each element of this relation has the following form: $(\llbracket I_1[\,P_1\,]\rrbracket, C, a, r_s, \llbracket I_2[\,P_2\,]\rrbracket)$. We define $\mathcal{B}_1, \mathcal{C}, A, R, \mathcal{B}_2$ as the union of the first, second, third, fourth and fifth projection of the elements belonging to $\xrightarrow[\mathtt{Box},\equiv]{}^{I[\,P\,],S}$, respectively. Showing that these sets are finite let us deduce the finiteness of the relation.

The sets $\mathcal{B}_1$ and $\mathcal{B}_2$ are finite because they are subsets of $ds(I[\,P\,],S)/{\equiv_b}$ which, by hypothesis, has finite cardinality. The same holds for R given that it is a subset of $\{\infty, \not\infty, ?\}$. The set $\mathcal{C}$ is finite as well, in fact its elements are the composition, by means of the conjunction operator, of a finite number of conditions appearing in the congruence classes of $ds(I[\,P\,],S)/{\equiv_b}$. The set A contains elements with the following form: τ, (T,U), $T?m$ and $T!m$. This elements are finite because there is a finite set of sorts from which we can choose T and U and because m belongs to $\mathsf{sn}(S)$, which is a finite set as well. With that we can conclude the desired result. □

From the previous lemma we conclude that the finiteness of $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ only depends on that of $ds(I[\,P\,],S)/{\equiv_b}$. More precisely, considering that the set of the sets of interfaces which appear in $ds(I[\,P\,],S)/{\equiv_b}$ is finite, the finiteness of $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ depends on the number of configuration assumable by the internal program P of the box.

In Table 5.2, we propose a modified version of the semantics of Table 3.4. It describes how the internal program of a box can evolve without considering information which is not relevant to establish the finiteness of $ds(I[\,P\,],S)/{\equiv_b}$ (i.e. interface sets, label conditions and rates). In the following lemma we show the relation between the original semantics and the simplified one.

Lemma 5.3.4 *Let $I_1[\,P_1\,]$ be a well-formed box. If $I_1[\,P_1\,]_\gamma \xrightarrow{C,a}_r I_1[\,P_2\,]_\gamma$ then $P_1 \xrightarrow{a'} P_2$ and one of the followings holds:*

1. $a = a' \in \{n!m, n?m, \tau\}$
2. $a = (T,U)$ *and* $a' = \tau$
3. $a = S?m$, $(b,S) \in I$ *and* $a' = b?m$
4. $a = S!m$, $(b,S) \in I$ *and* $a' = b!m$

Sketch. By induction on the length of the derivation tree of $I[\,P\,]_\gamma \xrightarrow{C,a}_r I_1[\,P_1\,]_\gamma$. □

With Corollary 5.3.5 we lift the result of the previous lemma to the relation $\xrightarrow[\mathtt{Box}]{}$.

Corollary 5.3.5 *Let $I_1[\,P_1\,]$ be a well-formed box. If $I_1[\,P_1\,] \xrightarrow[\mathtt{Box}]{C,a}_r I_2[\,P_2\,]$ then $P_1 \xrightarrow{a'} P_2$ and one of the followings holds:*

1. $a = a' \in \{n!m, n?m, \tau\}$
2. $a = (T,U)$ *and* $a' = \tau$
3. $a = S?m$, $(b,S) \in I$ *and* $a' = b?m$

$$\text{s1}\ \frac{}{n!m.\,P \xrightarrow{n!m} P} \qquad \text{s2}\ \frac{}{n?y.\,P \xrightarrow{n?m} P\{m/y\}}$$

$$\text{s3}\ \frac{}{\mathsf{ch}(r,a,T).\,P \xrightarrow{\tau} P}$$

$$\text{s4}\ \frac{\pi.\,P \xrightarrow{a} P'}{*\pi.\,P \xrightarrow{a} *\,\pi.\,P \mid P'}$$

$$\text{s5-l}\ \frac{P_1 \xrightarrow{n!m} P_1' \qquad P_2 \xrightarrow{n?m} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'} \qquad \text{s5-r}\ \frac{P_1 \xrightarrow{n?m} P_1' \qquad P_2 \xrightarrow{n!t} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

$$\text{s6}\ \frac{M \xrightarrow{a} M'}{\langle C\rangle M \xrightarrow{a} M'}$$

$$\text{s7-l}\ \frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q} \qquad \text{s7-r}\ \frac{Q \xrightarrow{a} Q'}{P \mid Q \xrightarrow{a} P \mid Q'}$$

$$\text{s8-l}\ \frac{M \xrightarrow{a} M'}{M + N \xrightarrow{a} M'} \qquad \text{s8-r}\ \frac{N \xrightarrow{a} N'}{M + N \xrightarrow{a} N'}$$

Table 5.2 – Operational semantics for internal processes.

4. $a = S!m$, $(b, S) \in I$ *and* $a' = b!m$

Proof. By definition of $\xrightarrow[\texttt{Box}]{}$ it follows immediately that $I_1[\,P_1\,]_\gamma \xrightarrow{C,a}_r I_1[\,P_2\,]_\gamma$ which, by Corollary 5.3.4 implies the desired $P_1 \xrightarrow{a'} P_2$.

□

In order to define an over-approximation of the states which can be assumed by an internal program P_1 during the evolution of the box $I[\,P_1\,]$, we define the notion of internal derivative of $I[\,P_1\,]$ with respect to S.

Definition 5.3.6 (Internal derivative of $I[\,P_1\,]$ with respect to S) *Given $I[\,P_1\,]$ is a well-formed box, P_2 is a (one-step) internal derivative of $I[\,P_1\,]$ with respect to the system S if it holds that $P_1 \xrightarrow{a} P_2$ and $a \in \{n!m, n?m\} \Rightarrow n \in \mathsf{sub}(I) \wedge m \in \mathsf{sn}(S)$. More generally if $P_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} P_n$ and $a_i \in \{n!m, n?m\} \Rightarrow n \in \mathsf{sub}(I) \wedge m \in \mathsf{sn}(S)$ for $i \in [1, n)$, then P_n is a internal derivative of $I[\,P_1\,]$ with respect to S. We regard the set of processes comprising P_1 and the internal derivatives of $I[\,P_1\,]$ with respect to S as $ids(I[\,P_1\,],S)$.*

With the next lemma we show that $ids(I[\,P_1\,],S)$ contains all the internal programs of the box belonging to $pds(I[\,P\,], S)$.

Corollary 5.3.7 *Let S be a well-formed system and $I[\,P\,]$ a well-formed box. If $I'[\,P'\,] \in pds(I[\,P\,], S)$ then $P' \in ids(I[\,P\,], S)$.*

Sketch. By induction on the number of transitions necessary in order to reach $I'[\,P'\,]$ starting from $I[\,P\,]$. It follows immediately by Corollary 5.3.5.

□

Lemma 5.3.8 states that two equivalent boxes can perform the same transitions with the exception of input actions receiving names that clash with the subjects of one of the boxes.

Lemma 5.3.8 *Let $I_1[\,P_1\,]$ be a well-formed box and γ a box label. If $I_1[\,P_1\,]_\gamma \xrightarrow{C,a}_r I_2[\,P_2\,]_\gamma$ and $I_1'[\,P_1'\,] = I_1[\,P_1\,]$ then the following hold:*

1. *if $a \neq \{k?t\}$ then $I_1'[\,P_1'\,]_\gamma \xrightarrow{C,a}_r I_2'[\,P_2'\,]_\gamma$ with $I_2[\,P_2\,] = I_2'[\,P_2'\,]$.*
2. *if $a = \{k?t\}$ and $m \notin \mathsf{sub}(I')$ then $I_1'[\,P_1'\,]_\gamma \xrightarrow{C,a}_r I_2'[\,P_2'\,]_\gamma$ with $I_2[\,P_2\,] = I_2'[\,P_2'\,]$.*

Proof. We observe that if $I_1[\,P_1\,] = I_1'[\,P_1'\,]$, by definition of α-equivalence, the following relation between $I_1'[\,P_1'\,]$ and $I_1[\,P_1\,]$ holds: there exists a set of subjects interfaces $\{b_1, \ldots, b_n\}$ such that:

$$I_1[\,P_1\,] = \{(b_1, T_1), \ldots, (b_n, T_n)\} \cup I[\,P_1\,]$$

and

$$I_1'[\,P_1'\,] = \{(b_1', T_1), \ldots, (b_n', T_n)\} \cup I[\,P_1\{^{b_1', \ldots, b_n'}/_{b_1, \ldots, b_n}\}\,]$$

For the sake of clarity we consider the particular case where the cardinality of $\{b_1, \ldots, b_n\}$ is one. In so doing we do not lose generality because we can obtain the equality between $I_1[\,P_1\,]$ and $I_1'[\,P_1'\,]$ creating a chain of equalities for which the cardinality of $\{b_1, \ldots, b_n\}$ is one. Thus we consider the case where: $I_1[\,P_1\,] = \{(b, T)\} \cup I[\,P_1\,]$ and $I_1'[\,P_1'\,] = \{(b', T)\} \cup I[\,P_1\{^{b'}/_{b}\}\,]$. We show the thesis by induction on the length of the derivation tree of $\{(b, T)\} \cup I[\,P_1\,]_\gamma \xrightarrow{C,a}_r I_2[\,P_2\,]_\gamma$. We only consider the interesting case of rule s2 with $k = T$ and $t = m$:

$$\{(b, T)\} \cup I[\,b?y.\,P\,]_\gamma \xrightarrow{\mathsf{true}, T?m}_0 I[\,P\{^{m}/_{y}\}\,]_\gamma$$

We have to show that if $m \notin \{(b', T)\} \cup I$ then the following holds:

$$\{(b', T)\} \cup I[\,b'?y.\,(P\{^{b'}/_{b}\})\,]_\gamma \xrightarrow{\mathsf{true}, T?m}_0 I[\,(P\{^{b'}/_{b}\})\{^{m}/_{y}\}\,]_\gamma$$

Given that $\{(b', T)\} \in \{(b', T)\} \cup I$ and, by hypothesis, $m \notin \{(b', T)\} \cup I$, the side condition of s2 is satisfied and we can obtain the desired transition.

□

We lift the result of the previous lemma to the relation $\xrightarrow[\mathtt{Box}]{}$.

Corollary 5.3.9 *Let $I_1[\,P_1\,]$ be a well-formed box. If $I_1[\,P_1\,] \xrightarrow[\mathtt{Box}]{C,a}_r I_2[\,P_2\,]$ and $I_1'[\,P_1'\,] = I_1[\,P_1\,]$ then the following hold:*

1. *if $a \notin \{T!m, T?m\}$ then $I_1'[\,P_1'\,] \xrightarrow[\mathtt{Box}]{C,a}_r I_2'[\,P_2'\,]$ with $I_2[\,P_2\,] = I_2'[\,P_2'\,]$.*
2. *if $a \in \{T!m, T?m\}$ and $m \notin \mathsf{sub}(I')$ then $I_1'[\,P_1'\,] \xrightarrow[\mathtt{Box}]{C,a}_r I_2'[\,P_2'\,]$ with $I_2[\,P_2\,] = I_2'[\,P_2'\,]$.*

Proof. By definition of $\xrightarrow[\texttt{Box}]{}$ it follows that $I_1[\,P_1\,]_\gamma \xrightarrow{C,a}_r I_1[\,P_2\,]_\gamma$ which, by Lemma 5.3.8, lets us conclude with the desired result.

□

In the following lemma we prove that the set of derivatives and the set of proper derivatives of $I[\,P\,]$ with respect to S have the same elements under the assumption that the sent names of S do not clash with the subjects of I.

Lemma 5.3.10 *Let S be a well-formed system and $I[\,P\,]$ a well-formed box. If $\mathsf{sn}(S) \cap \mathsf{sub}(I) = \emptyset$ and $I'[\,P'\,] \in ds(I[\,P\,], S)$ then $\exists I''[\,P''\,] \in pds(I[\,P\,], S) : I''[\,P''\,] = I'[\,P'\,]$.*

Proof. By induction on the number of transitions necessary in order to reach $I'[\,P'\,]$ starting from a process $I'_1[\,P'_1\,] = I[\,P\,]$.

base case: it holds that $I'[\,P'\,] = I[\,P\,]$ and $I[\,P\,]$ belongs to $pds(I[\,P\,],S)$ by definition of ds.

step case: we know, by definition of $ds(I[\,P\,], S)$, that there exists $I'_1[\,P'_1\,] = I[\,P\,]$ such that:

$$I'_1[\,P'_1\,] \xrightarrow[\texttt{Box}]{C'_1,a'_1} r'_1 \ \ldots \xrightarrow[\texttt{Box}]{C'_{n'},a'_{n'}} r'_{n'}\ I'_{n'}[\,P'_{n'}\,] \xrightarrow[\texttt{Box}]{C,a} r\ I'[\,P'\,]$$

By definition of $ds(I[\,P\,], S)$, we know that $I'_{n'}[\,P'_{n'}\,] \in ds(I[\,P\,], S)$ thus, by induction hypothesis, we can say that $\exists I_n[\,P_n\,] \in pds(I[\,P\,], S) : I'_{n'}[\,P'_{n'}\,] = I_n[\,P_n\,]$. At this point we make the following observations:

1. $I'_{n'}[\,P'_{n'}\,] = I_n[\,P_n\,]$;
2. if $a \in \{T!m, T?m\}$ then $m \in \mathsf{sn}(S)$ by definition of $ds(I[\,P\,],S)$;
3. $I[\,P\,] \cap \mathsf{sn}(S) = \emptyset$ implies $I_n \cap \mathsf{sn}(S) = \emptyset$ because $\xrightarrow[\texttt{Box}]{}$ preserves the subjects of the interfaces.
4. Thanks to points 2 and 3 we can say that $m \notin \mathsf{sub}(I_n)$.

These observations let us apply the result of Corollary 5.3.9 and obtain the desired result $I_n[\,P_n\,] \xrightarrow[\texttt{Box}]{C,a}_r I_{n+1}[\,P_{n+1}\,]$ with $I_{n+1}[\,P_{n+1}\,] = I'_{n'}[\,P'_{n'}\,]$ which lets us state $\exists I''[\,P''\,] \in pds(I[\,P\,], S) : I''[\,P''\,] = I'_{n'}[\,P'_{n'}\,]$.

□

Lemma 5.3.11 states that the number of interface sets associated with the boxes of $pds(I[\,P\,], S)$ is finite.

Lemma 5.3.11 *Let S be a well-formed system and $I[\,P\,]$ a well-formed box. If we partition the boxes of the set $pds(I[\,P\,], S)$ according to the set of interfaces they are associated with, then we obtain a finite number of partitions.*

Proof. It follows immediately from two observations:

1. The set $pds(I[\,P\,], S)$ is defined on the relation $\xrightarrow[\texttt{Box}]{}$ which cannot add interfaces to the starting set of interfaces I.

2. The interfaces of I can be associated with a finite number of sorts.

□

The result of the previous lemma binds the finiteness of the set $ids(I[\,P\,], S)/\equiv_b$ to the finiteness of $pds(I[\,P\,], S)/\equiv_b$. In fact, given that the sets of interfaces associated with the boxes of $pds(I[\,P\,], S)$ is finite, the only source of infiniteness remains the internal program of the boxes.

Lemma 5.3.12 *Let S be a well-formed system and $I[\,P\,]$ a well-formed box. If $ids(I[\,P\,], S)/\equiv_p$ has finite cardinality then the same holds for the cardinality of $pds(I[\,P\,], S)/\equiv_b$.*

Proof. We show the contrapositive of the thesis: if $pds(I[\,P\,], S)/\equiv_b$ has infinite cardinality then the same holds for $ids(I[\,P\,], S)/\equiv_p$.

We observe that if $[\![I'[\,P'\,]]\!] \in pds(I[\,P\,], S)/\equiv_b$ then $\exists I''[\,P''\,] \in pds(I[\,P\,], S) : I''[\,P''\,] \equiv_b I'[\,P'\,]$. Therefore, given that the cardinality of $pds(I[\,P\,], S)/\equiv_b$ is infinite, we have $W \subseteq pds(I[\,P\,], S)$ such that W is infinite and all the boxes belong to a different congruence class. At this point, applying the result of Lemma 5.3.11, we can state that if we partition the boxes of W depending on their interfaces we obtain a finite number of partitions. Given that W is infinite this implies that there exists at least one partition with an infinite number of boxes which differs because of their internal program. All these internal programs, by Corollary 5.3.7, belong also to $ids(I[\,P\,], S)$, thus we can conclude that $ids(I[\,P\,], S)/\equiv_p$ has infinite cardinality.

□

If $\mathsf{sn}(S) \cap \mathsf{sub}(I) = \emptyset$ we can lift the result of the previous lemma to ds. Note that the hypothesis $\mathsf{fn}(S) \cap \mathsf{sub}(I) = \emptyset$ does not compromise the generality of the result, indeed, given a box $I[\,P\,]$ and a system S, there always exists a box $I'[\,P'\,] = I[\,P\,]$ such that $\mathsf{fn}(S) \cap \mathsf{sub}(I) = \emptyset$ and $ds(I[\,P\,], S) = ds(I'[\,P'\,], S)$.

Theorem 5.3.13 *Let S be a well-formed system and $I[\,P\,]$ a well-formed box. If $\mathsf{sn}(S) \cap \mathsf{sub}(I) = \emptyset$ and the cardinality of $ids(I[\,P\,], S)/\equiv_p$ is finite then the same holds for the cardinality of $ds(I[\,P\,], S)/\equiv_b$.*

Proof. By Lemma 5.3.10 $\mathsf{sn}(S) \cap \mathsf{sub}(I) = \emptyset$ implies $ds(I[\,P\,], S)/\equiv_b \;=\; pds(I[\,P\,], S)/\equiv_b$. This observation lets us immediately conclude given that, by Lemma 5.3.12, we know that if $ids(I[\,P\,], S)/\equiv_p$ has finite cardinality then the same holds for the cardinality of $pds(I[\,P\,], S)/\equiv_b$.

□

Combining the results of Lemma 5.3.3 and Theorem 5.3.13 we know that, given a well-formed system S and a well-formed box $I[\,P\,]$, if $ids(S, I[\,P\,])/\equiv_b$ is finite the same holds for $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$. Therefore we define a class of internal programs whose elements, evolving as defined by the semantics of Table 5.2, can reach a finite number of congruence classes. In Table 5.3 we present the grammar generating this class of internal programs. Parallel operator and replication only appear at the first level of depth. The set Name is partitioned into $\mathsf{Name_P}$ and $\mathsf{Name_R}$. We refer to the elements of $\mathsf{Name_P}$ and $\mathsf{Name_R}$ as n_p, x_p,*etc.* and n_r, x_r,*etc.*, respectively. The name of the first set can be used for internal communication; they can be placed in any position of the

$$\begin{array}{rcl} P & ::= & P|P \mid M \mid *\, n_r?x_p.\, M \\ M & ::= & \mathsf{nil} \mid \pi.\, M \mid M + M \mid \langle C \rangle M \mid n_r!m_p.\, \mathsf{nil} \\ \pi & ::= & n_p!m_p \mid n_p?x_p \mid \mathsf{ch}(r, n_p, S) \end{array}$$

Table 5.3 – Generating grammar of internal programs in $\mathcal{FF}$.

$$w(P) = \begin{cases} w(P_1) + w(P_2) & \text{if } P = P_1 \,|\, P_2 \text{ and } P \not\equiv_p \mathsf{nil} \\ 0 & \text{if } P \equiv_p \mathsf{nil} \\ 1 & \text{otherwise} \end{cases}$$

Table 5.4 – Function w which computes the width of an internal program.

internal program but they cannot be used in actions guarding a replication. We reserve the role of replication guard to input actions over names belonging to $\mathsf{Name_R}$. We allow outputs over channel names of $\mathsf{Name_R}$ only if they are followed by the process nil.
If P is generated by the grammar of Table 5.3 we say that P is in $\mathcal{FF}$ (*Finite Form*). We lift the notion of $\mathcal{FF}$ to boxes saying that $I[\,P\,]$ is in $\mathcal{FF}$ if P is in $\mathcal{FF}$ and $\mathsf{sub}(I) \cap \mathsf{Name_R} = \emptyset$. The condition $\mathsf{sub}(I) \cap \mathsf{Name_R} = \emptyset$ forbids the interface names being involved in actions guarding replications. The characteristic of the processes in $\mathcal{FF}$ is that they cannot increase their *width*. The width of a process is defined in Table 5.4 and is based on the concept of elementary parallel component.

Definition 5.3.14 (Elementary parallel component) *A process is said to be an elementary parallel component if it does not have the form $P \,|\, P$.*

In words, we compute the width of a process by counting how many subprocesses, not congruent to nil, it joins by means of the parallel operator. Processes in $\mathcal{FF}$ do not increase their width because of the followings properties:

1. Replication bodies do not contains the parallel operator;
2. The policy of channel name usage allows a subprocess to spawn a replication only if it becomes the nil process.

This means that when an output action spawns a replication, the width of the process remains constant (or decreases in the case of a replication having nil as body). In order to prove this property we show that $\mathcal{FF}$ of internal programs is preserved by τ-transitions.

Lemma 5.3.15 *Given P an internal program in $\mathcal{FF}$, if $P \xrightarrow{a} Q$ then the following holds:*

1. *if $a = n?m$ and $m \in \mathsf{Name_P}$ then Q is in $\mathcal{FF}$;*
2. *if $a = n!m$ then $m \in \mathsf{Name_P}$ and Q is in $\mathcal{FF}$;*
3. *if $a = \tau$ then Q is in $\mathcal{FF}$.*

Sketch. By induction on the length of the derivation tree of $P \xrightarrow{a} Q$.

□

From the previous result it immediately follows that if $I[\,P\,]$ is in $\mathcal{FF}$ and the sent names of S belong to $\mathsf{Name_P}$ then all the processes belonging to $ids(I[\,P\,], S)$ are in $\mathcal{FF}$.

Corollary 5.3.16 *Given a well formed system S and a box $I[\,P\,]$ in $\mathcal{FF}$ if $Q \in ids(I[\,P\,], S)$ with $\mathsf{sn}(S) \subseteq \mathsf{Name_P}$ then Q is in $\mathcal{FF}$.*

Sketch. By induction on the number of transition necessary to reach Q starting from P, and exploiting the result of Lemma 5.3.15.

□

In Lemma 5.3.17 we show how transitions deduced with rules of Table 5.2 modify the width of a process in $\mathcal{FF}$.

Lemma 5.3.17 *Given P a process in $\mathcal{FF}$, if $P \xrightarrow{a} Q$ then the followings hold:*

1. *if $a = n_r?m$ and $m \in \mathsf{Name_P}$ then $w(Q) \leq w(P) + 1$;*
2. *if $a = n_r!m$ then $m \in \mathsf{Name_P}$ and $w(Q) = w(P) - 1$;*
3. *if $a \in \{n_p!m, n_p?m, \tau\}$ and $m \in \mathsf{Name_P}$ then $w(Q) \leq w(P)$.*

Proof. By induction on the length of the derivation tree of $P \xrightarrow{a} Q$. We only consider the interesting cases.

base case s1: We have two cases.

If $n = n_r$ then, by definition of $\mathcal{FF}$, $P = \mathsf{nil}$. This implies that $w(\mathsf{nil}) = w(n_r!m.\,\mathsf{nil}) - 1$ and we prove the thesis.

If $n = n_p$, given that $n_p!m.\,P$ is in $\mathcal{FF}$, then $w(n!m.\,P) = 1$ and P is generated by the non-terminal M of the grammar of Table: 5.3. This implies the desired $w(P) \leq 1$.

base case s2: If $n_p?m.\,P$ is in $\mathcal{FF}$ then $w(n!m.\,P) = 1$ and P is generated by the non-terminal M. This implies that $w(P) \leq 1$ and the same holds for $P\{^m/_y\}$. At this point we have proved that $w(P\{^m/_y\}) \leq w(n?m.\,P)$ which implies the desired result.

step case s4: If $*\pi.\,P$ is in $\mathcal{FF}$, by definition of $\mathcal{FF}$, we know that $\pi.\,P = n_r?y_p.\,P$, $a = n_r?m$ and $P' = P\{^m/_{y_p}\}$. This implies that, by proceeding as in the case s2, we obtain $w(P\{^m/_y\}) \leq 1$. At this point we have what we need in order to conclude the desired $w(*\pi.\,P \mid P\{^m/_{y_p}\}) = w(*\pi.\,P) + w(P\{^m/_{y_p}\}) \leq w(*\pi.\,P) + 1$.

step case s5-l: If $P_1 \mid P_2$ is in $\mathcal{FF}$ then both P_1 and P_2 are in $\mathcal{FF}$ as well.

If $n = n_r$ then, by induction hypothesis, $w(P_1') \leq w(P_1) + 1$ and $w(P_2') = w(P_2) - 1$. This implies the desired $w(P_1' \mid P_2') = w(P_1') + w(P_2') \leq w(P_1) + 1 + w(P_2) - 1 = w(P_1 \mid P_2)$.

If $n = n_p$ then, by induction hypothesis, $w(P_1') \leq w(P_1)$ and $w(P_2') = w(P_2)$ which immediately implies the desired $w(P_1' \mid P_2') = w(P_1') + w(P_2') \leq w(P_1) + w(P_2) = w(P_1 \mid P_2)$.

$$epc(P) = \begin{cases} epc(P_1) \cup epc(P_2) & \text{if } P = P_1 \mid P_2 \\ \{P\} & \text{otherwise} \end{cases}$$

$$\begin{aligned} sp(P_1 \mid P_2) &= \{P_1 \mid P_2\} \cup sp(P_1) \cup sp(P_2) \\ sp(M_1 + M_2) &= \{M_1 + M_2\} \cup sp(M_1) \cup sp(M_2) \\ sp(\langle C \rangle M) &= \{\langle C \rangle M\} \cup sp(M) \\ sp(\pi.\, P) &= \{\pi.\, P\} \cup sp(P) \\ sp(*P) &= \{*P\} \cup sp(P) \\ sp(\mathsf{nil}) &= \{\mathsf{nil}\} \end{aligned}$$

Table 5.5 – Definition of the functions *epc* and *sp*.

□

Finally we prove that, if a process is in $\mathcal{FF}$ and can exclusively receive names belonging to $\mathsf{Name_P}$, it cannot increase its width.

Corollary 5.3.18 *Given a well formed system S and a box $I[\,P\,]$ in $\mathcal{FF}$, if $Q \in ids(I[\,P\,],S)$ and $\mathsf{sn}(S) \subseteq \mathsf{Name_P}$ then $w(Q) \leq w(P)$.*

Sketch. By induction on the number of transitions necessary to reach Q starting from P. It follows immediately exploiting the result of Lemmas 5.3.16 and 5.3.17.

□

We are interested in internal programs which cannot increase their width because they assume configurations belonging to a finite number of congruence classes. Therefore, if a box $I[\,P\,]$ is in $\mathcal{FF}$ and executes in a system where it can only receive names belonging to $\mathsf{Name_P}$, we can conclude that $ids(I[\,P\,],S)/\equiv_p$ has finite cardinality.

The underlying intuition is that the internal programs of $ids(I[\,P\,],S)$ are made of elementary parallel components belonging to a finite set. We can compute an over-approximation A of such a set by collecting all the subprograms of P and by substituting, in all the possible ways, their bound names with the free names of S. It follows that if the internal programs of $ids(I[\,P\,],S)$ do not have more than $w(P)$ elementary parallel components different from nil, and we quotient by congruence $ids(I[\,P\,],S)$, then we obtain a finite set. In fact, given that A is finite, we can build a limited number of processes by composing its elements.

In the rest of this section we formalize and show what is stated in our intuition. In the following proofs we heavily use two functions. The first one, *epc*, is used to extract the elementary parallel components of an internal program; the function *sp*, given an internal program, returns all its subprograms. Their formal definitions are reported in Table 5.5. We also introduce the set $rsp_P(N, X)$. It is made of all the internal programs obtained by replacing the names of N with the names of X inside the subprogram of P which are elementary parallel components.

Definition 5.3.19 (Over-set of reachable subprograms) *The over-set of reachable subprograms of the internal program P with respect to the sets of names N and X is defined as*

$rsp_P(N, X) = \{Q\sigma \mid Q \in sp(P)$, Q is an elementary parallel component and $cosupp(\sigma) \subseteq N$ and $supp(\sigma) \subseteq X\}$. We write $rsp_P(N)$ for $rsp_P(N, \mathsf{bn}(P))$.

In the following observations and lemmas we highlight some results necessary to prove interesting properties of *rsp*. We start with three observations which immediately follow by definition of *rsp*. The first states that if Q is a subprogram of P, then $rsp_Q(N, X) \subseteq rsp_P(N, X)$.

Observation 5.3.20 *Let N and X be two sets of names and P an internal program. If $Q \in sp(P)$ then $rsp_Q(N, X) \subseteq rsp_P(N, X)$.*

The next observation says that rsp_P is monotonic.

Observation 5.3.21 *Let N_1, N_2, X_1 and X_2 be sets of names and P an internal program. If $N_1 \subseteq N_2$ and $X_1 \subseteq X_2$ then $rsp_P(N_1, X_1) \subseteq rsp_P(N_2, X_2)$.*

We also observe that whatever the set of names N, the set $rsp_P(N)$ contains the elementary parallel component of P.

Observation 5.3.22 *Given an internal program P and a set of names N it holds that $epc(P) \subseteq rsp_P(N)$.*

In the following lemma we show that the finiteness of $rsp_P(N)$ depends on that of N.

Lemma 5.3.23 *Let N be a set of names and P an internal program. If the cardinality of N is finite then the same holds for $rsp_P(N)$.*

Proof. The dimension of P is finite, this implies that the sets $sp(P)$ and $\mathsf{bn}(P)$ have finite cardinality. Considering these observations and that the cardinality of N is finite, by definition of $rsp_P(N)$, we conclude with the desired result.

□

Lemma 5.3.24 states that τ-transitions do not generate free names and bound names.

Lemma 5.3.24 *Let P be a process. Suppose $P \xrightarrow{a} Q$:*

1. *If $a = n!m$ then $n, m \in \mathsf{fn}(P)$ and $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P)$.*
2. *If $a = n?m$ then $n \in \mathsf{fn}(P)$ and $\mathsf{fn}(Q) \subseteq (\mathsf{fn}(P) \cup \{m\})$.*
3. *If $a = \tau$ then $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P)$.*

Moreover $\mathsf{bn}(Q) \subseteq \mathsf{bn}(P)$.

Proof. By induction on the length of the derivation tree of $P \xrightarrow{a} Q$.

□

Exploiting the previous lemma we show that the free names of the internal programs belonging to $ids(I[\,P\,],S)$ are a subset of the free names of P and S.

Corollary 5.3.25 *Let $I[\,P\,]$ be a well-formed box and S a well-formed system. If $Q \in ids(I[\,P\,],S)$ then $\mathsf{fn}(Q) \in \mathsf{fn}(P) \cup \mathsf{fn}(S)$.*

Proof. By induction on the number of transitions necessary for reaching Q starting from P.

base case: In this case $P = Q$ and it follows immediately that $\mathsf{fn}(P) \subseteq \mathsf{fn}(P) \cup \mathsf{fn}(S)$.

step case: In this case $P \xrightarrow{a_1} \cdots \xrightarrow{a_n} P_n \xrightarrow{a} Q$.

If $a \neq x?m$, by Lemma 5.3.24, we know that $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P_n)$.

If $a = x?m$ we have, again by Lemma 5.3.24, that $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P_n) \cup \{m\}$. Considering that, by definition of *ids*, $m \in \mathsf{sn}(S) \subseteq \mathsf{fn}(S)$ we can say that $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P_n) \cup \mathsf{fn}(S)$. Thus in every case we have that $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P_n) \cup \mathsf{fn}(S)$. At this point, observing that by inductive hypothesis $\mathsf{fn}(P_n) \subseteq \mathsf{fn}(P) \cup \mathsf{fn}(S)$ which implies $\mathsf{fn}(P_n) \cup \mathsf{fn}(S) \subseteq \mathsf{fn}(P) \cup \mathsf{fn}(S)$, we can conclude, by transitivity of $\subseteq$, with the desired $\mathsf{fn}(Q) \subseteq \mathsf{fn}(P) \cup \mathsf{fn}(S)$.

□

In the following lemma, given two sets of names X and N, we generate a set of substitutions whose *cosupp* and *supp* are subset of X and N, respectively, and we prove that this set is closed by function composition.

Lemma 5.3.26 *Given N, X two sets of names and a set of substitutions $\Sigma = \{\sigma \mid cosupp(\sigma) \subseteq N$ and $supp(\sigma) \subseteq X\}$ if $\sigma_1, \sigma_2 \in \Sigma$ then $\sigma_1 \circ \sigma_2 \in \Sigma$.*

Proof. We note that $x \notin supp(\sigma_1) \cup supp(\sigma_2) \Rightarrow x\sigma_1\sigma_2 = x \Rightarrow x \notin supp(\sigma_1 \circ \sigma_2)$. $x \notin cosupp(\sigma_1) \cup cosupp(\sigma_2) \Rightarrow x\sigma_1\sigma_2 = x \Rightarrow x \notin cosupp(\sigma_1 \circ \sigma_2)$. The contrapositives of these implications are the following: $x \in supp(\sigma_1 \circ \sigma_2) \Rightarrow x \in supp(\sigma_1) \cup supp(\sigma_2)$ and $x \in cosupp(\sigma_1 \circ \sigma_2) \Rightarrow x \in cosupp(\sigma_1) \cup cosupp(\sigma_2)$. From these implications it follows that $supp(\sigma_1 \circ \sigma_2) \subseteq X$ and $cosupp(\sigma_1 \circ \sigma_2) \subseteq N$ which implies $\sigma_1 \circ \sigma_2 \in \Sigma$.

□

Lemma 5.3.27 states that if we obtain an internal program P by applying a substitution σ to Q, then we can obtain all the subprograms of P by applying a substitution σ' to a subprocess of Q. Moreover, *cosupp* and *supp* of σ' are included in those of σ.

Lemma 5.3.27 *Given two internal programs P and Q and a substitution σ, if $Q\sigma = P$ then $\forall P' \in sp(P)$, a subprogram $Q' \in sp(Q)$ and a substitution σ' exist such that $Q'\sigma' = P'$ with $supp(\sigma') \subseteq supp(\sigma)$ and $cosupp(\sigma') \subseteq cosupp(\sigma)$.*

Proof. We proceed by induction on the structure of the processes.

base case nil: we have that $Q\sigma = \mathsf{nil}$ iff $Q = \mathsf{nil}$. Given that $sp(\mathsf{nil}) = \{\mathsf{nil}\}$ the thesis immediately follows.

step case $x!y.P$: given that the σ function does not modify the structure of the processes we have that $Q\sigma = x!y.P$ iff $Q = z!w.P'$ for some z, w and σ s.t $z\sigma = x$ and $w\sigma = y$. This, given that $\mathsf{fn}(P') \subseteq \mathsf{fn}(z!w.P')$, implies that $Q\sigma = x!y.(P'\sigma)$. Now, observing that $P'\sigma = P$ we apply the inductive hypothesis concluding that for each element of $sp(P)$ there exists a process of $sp(P')$ such that the first results by the application of a substitution σ' to the second, with $supp(\sigma') \subseteq supp(\sigma)$ and $cosupp(\sigma') \subseteq cosupp(\sigma)$. Given that $sp(x!y.P) = \{x!y.P\} \cup sp(P)$ and $sp(z!w.P') = \{z!w.P'\} \cup sp(P')$ we here proved the thesis by hypothesis: we have that $(z!w.P')\sigma = x!y.P$.

step case $x?y$: as for the previous case it holds that $Q\sigma = x?y.P$ iff $Q = z?y.P'$ for some z, σ s.t $z\sigma = x$. Considering that $\mathsf{fn}(P') \subseteq \mathsf{fn}(z?y.P') \cup \{y\}$, we have that $Q\sigma = x?y.(P'\sigma_1)$, where $x\sigma_1 = x\sigma$ if $x \neq y$ and $x\sigma_1 = x$ otherwise. Now, observing that $P'\sigma_1 = P$ we apply the inductive hypothesis concluding that for each element of $sp(P)$ there exists a process of $sp(P')$ such that the first results by the application of a substitution σ' to the second with $supp(\sigma') \subseteq supp(\sigma_1)$ and $cosupp(\sigma') \subseteq cosupp(\sigma_1)$. Now, noting that by definition of σ_1, it holds that $supp(\sigma_1) \subseteq supp(\sigma)$ and $cosupp(\sigma_1) \subseteq cosupp(\sigma)$ and that $sp(x?y.P) = \{x?y.P\} \cup sp(P)$ and $sp(z?y.P') = \{z?w.P'\} \cup sp(P')$ we proved the thesis because, by hypothesis, we have that $(z?w.P')\sigma = x?y.P$.

We do not consider the other cases because they are simple applications of the inductive hypothesis.

□

The next five lemmas introduce properties of *rsp* which allow us to prove Lemma 5.3.33, that, given a set of names N and under the assumption that $P \xrightarrow{a} Q$, shows the relation between $rsp_P(N)$ and $rsp_Q(N)$.

Lemma 5.3.28 *Given two sets of names X and N and a process P, if $n \in N$ and $x \in X$ then $rsp_{P\{n/x\}}(N, X) \subseteq rsp_P(N, X)$.*

Proof. We define the set of substitutions $\Sigma = \{\sigma \mid cosupp(\sigma) \subseteq N \text{ and } supp(\sigma) \subseteq X\}$. Applying the result of Lemma 5.3.27 on $P\{n/x\}$ we obtain that, if $R \in sp(P\{n/x\})$, then $\exists R' \in sp(P)$ such that $R = R'$ or $R = R'\{n/x\}$. Therefore, given the set $E_{P\{n/x\}} = \{T \mid T \in sp(P\{n/x\}) \text{ and } T$ is an elementary parallel component$\}$, if $R \in E_{P\{n/x\}}$ then there exists $R' \in sp(P)$ such that $R'\sigma = R$ with $\sigma \in \{\{n/x\}, \sigma_{id}\}$. Now we group in the set W those elements of $sp(P)$ obtainable by applying $\{n/x\}$ or σ_{id} to the internal programs of $E_{P\{n/x\}}$:

$$W = \{T \in sp(P) \mid T\sigma \in E_{P\{P/Z\}} \text{ with } \sigma \in \{\{n/x\}, \sigma_{id}\}\}$$

We observe that:

1. All the elements of W are subprograms of P.
2. All the elements of W are elementary parallel components given that, if we apply to them a substitution among $\{n/x\}$ and σ_{id} we an internal program of $E_{P\{n/x\}}$, whose elements are elementary parallel components.
3. $\{\{n/x\}, \sigma_{id}\} \subseteq \Sigma$. The first because, by hypothesis, $s \in N$ and $z \in X$, and the second by definition of Σ.

These observations let us state that $W \subseteq E_P = \{T \mid T \in sp(P) \text{ and } T$ is an elementary parallel component$\}$. Therefore we have that, applying a substitution of Σ to an element of E_P, we can obtain all the processes of $E_{P\{n/x\}}$. Observing that, by definition of *rsp*, all the elements of $rsp_{P\{n/x\}}(N, X)$ are generated by applying a substitution $\sigma \in \Sigma$ to a process that belongs to $E_{P\{n/x\}}$, we can conclude that starting from an element of E_P, applying two substitutions which belong to σ, we can obtain all the elements that belong to $rsp_{P\{n/x\}}(N, X)$. This, by Lemma 5.3.26, implies that the same holds applying only one substitution that belongs to Σ

and this lets us conclude, by definition of *rsp*, that all the elements of $rsp_{P\{^n/_x\}}(N, X)$ also belong to $rsp_P(N, X)$.

□

Lemma 5.3.29 *Given an internal program P and a set of names N, if $Q \in sp(P)$ then $rsp_Q(N) \subseteq rsp_P(N)$.*

Proof. By Observation 5.3.20, $rsp_Q(N, \mathsf{bn}(P)) \subseteq rsp_P(N)$. Given that, by definition of bound names, we have that $\mathsf{bn}(Q) \subseteq \mathsf{bn}(P)$, we can also say that $rsp_Q(N) \subseteq rsp_Q(N, \mathsf{bn}(P))$. Thanks to the chain of inclusions $rsp_Q(N) \subseteq rsp_Q(N, \mathsf{bn}(P)) \subseteq rsp_P(N,)$ we can conclude with the thesis.

□

Lemma 5.3.30 *Let P and Q be two internal programs. If all the elementary parallel components $T \in sp(Q)$ belong to $rsp_P(N, X)$ then $rsp_Q(N, X) \subseteq rsp_P(N, X)$.*

Proof. We define E_P and E_Q as the elementary parallel components that belong to $sp(P)$ and $sp(Q)$, respectively. Moreover we define $\Sigma = \{\sigma \mid supp(\sigma) \subseteq X \text{ and } cosupp(\sigma) \subseteq N\}$. By definition of *rsp*, if $R \in E_Q$ belongs to $rsp_P(N, X)$ then $T \in E_P$ and $\sigma \in \Sigma$ exist such that $T\sigma = R$. After that we observe that $U \in rsp_Q(N, X)$ implies that $R \in E_Q$ and $\sigma \in \Sigma$ exist such that $R\sigma = T$. Thus we can conclude that for each $U \in rsp_Q(N, X)$ a substitution $\sigma_1\sigma_2 \in \Sigma$ and an internal program $T \in W_P$ exist such that $T\sigma_1\sigma_2 = U$. The last observation let us conclude, by Lemma 5.3.26, that a substitution $\sigma_3 \in \Sigma$ exists such that $T\sigma_3 = U$, therefore $U \in rsp_P(N, X)$. After that we proved $rsp_Q(N, X) \subseteq rsp_P(N, X)$.

□

Lemma 5.3.31 *Let P and Q be two internal programs and N_i, X_i with $i \in [1, 3]$ sets of names. If $rsp_P(N_1, X_1) \subseteq rsp_Q(N_2, X_2)$, $N_3 \subseteq N_2$ and $X_3 \subseteq X_2$ then it holds that $rsp_P(N_3, X_3) \subseteq rsp_Q(N_2, X_2)$.*

Proof. If $rsp_P(N_1, X_1) \subseteq rsp_Q(N_2, X_2)$ then, for each $R \in rsp_P(N_1, X_1)$, a substitution σ and a process T exist such that $supp(\sigma) \subseteq Z_2$, $cosupp(\sigma) \subseteq S_2$, T is an elementary parallel component and belongs to $sp(Q)$ and $T\sigma = R$. This holds in particular if R is an elementary parallel component belonging to $sp(P)$. Therefore we can apply the result of Lemma 5.3.30 and obtain that $rsp_P(N_2, X_2) \subseteq rsp_Q(N_2, X_2)$. At this point, given that by hypothesis $N_3 \subseteq N_2$ and $X_3 \subseteq X_2$, we can conclude, by Observation 5.3.21, that $rsp_P(N_3, X_3) \subseteq rsp_Q(N_2, X_2)$.

□

Lemma 5.3.32 *Given P_1 and P_2 two processes, it holds that $rsp_{P_1|P_2}(N, X) = rsp_{P_1}(N, X) \cup rsp_{P_2}(N, X)$.*

Proof. By definition of *sp* we know that $sp(P_1 \mid P_2) = sp(P_1) \cup sp(P_2)$. Moreover we observe that $P_1 \mid P_2$ is not an elementary parallel component. These observations imply the following equality: $\{Q \mid Q \in sp(P_1 \mid P_2) \text{ and is an elementary parallel component}\} = \{Q \mid Q \in sp(P_1)$

and is an elementary parallel component$\} \cup \{Q \mid Q \in sp(P_2)$ and is an elementary parallel component$\}$. The same equality is preserved closing these sets by the substitutions belonging to $\{\sigma \mid cosupp(\sigma) \in N$ and $supp(\sigma) \in X\}$. This implies by definition of rsp that $rsp_{P_1|P_2}(N, X) = rsp_{P_1}(N, X) \cup rsp_{P_2}(N, X)$.

□

Lemma 5.3.33 *Let P be an internal program and N a set of names. If $\mathsf{fn}(P) \subseteq N$ and $P \xrightarrow{a} Q$:*

1. *If $a = n?m$ then $rsp_Q(N) \subseteq rsp_P(N \cup \{m\})$.*
2. *Otherwise $rsp_Q(N) \subseteq rsp_P(N)$.*

Proof. By induction on the length of the derivation tree of $P \xrightarrow{a} Q$.

base case s1:

$$n!m.\, P \xrightarrow{n!m} P$$

By Lemma 5.3.29 we know that $rsp_{n!m.\, P}(N) \subseteq rsp_P(N)$. The same proof can be applied to rule s3.

base case s2:

$$n?x.\, P \xrightarrow{n?m} P\{^m/_x\}$$

We know by definition of bound names that $\mathsf{bn}(P\{^m/_x\}) = \mathsf{bn}(P) \subseteq \mathsf{bn}(n?x.\, P)$. This implies that $rsp_{P\{^m/_x\}}(N) \subseteq rsp_{P\{^m/_x\}}(N \cup \{m\}, \mathsf{bn}(n?x.\, P))$. Moreover, observing that $m \in N \cup \{m\}$ and $x \in \mathsf{bn}(n?x.\, P)$ we can apply the result of Lemma 5.3.28 and obtain that $rsp_{P\{^m/_x\}}(N \cup \{m\}, \mathsf{bn}(n?x.\, P)) \subseteq rsp_P(N \cup \{m\}, \mathsf{bn}(n?x.\, P))$. As the last step, given that $P \in sp(n?x.\, P)$ and by observation 5.3.20, we note that $rsp_P(N \cup \{m\}, \mathsf{bn}(n?x.\, P)) \subseteq rsp_{n?x.\, P}(N \cup \{m\}, \mathsf{bn}(n?x.\, P))$. This chain of inclusions lets us conclude with $rsp_{P\{^m/_x\}}(N) \subseteq rsp_{n?x.\, P}(N \cup \{m\}, \mathsf{bn}(n?x.\, P))$.

step case s4:

$$\frac{\pi.\, P \xrightarrow{a} P'}{*\pi.\, P \xrightarrow{a} *\,\pi.\, P \mid P'}$$

We consider the case $a = n?m$ and thus we show that $rsp_{*\pi.\, P \mid P'}(N) \subseteq rsp_{*\pi.\, P}(N \cup \{m\})$. By Lemma 5.3.32 we know that $rsp_{*\pi.\, P \mid P'}(N) = W_1 \cup W_2$ where the set $W_1 = rsp_{*\pi.\, P}(N, \mathsf{bn}(*\pi.\, P \mid P'))$ and $W_2 = rsp_{P'}(N, \mathsf{bn}(*\pi.\, P \mid P'))$.

In order to prove the thesis we show that both these sets are included in $W = rsp_{*\pi.\, P}(N \cup \{m\})$. In the case of W_1 we observe that, by Lemma 5.3.24, $\mathsf{bn}(*\pi.\, P \mid P') \subseteq \mathsf{bn}(*\pi.\, P)$ and thus we can conclude by Observation 5.3.21.

In the case of W_2 we note that, by inductive hypothesis, $rsp_{P'}(N) \subseteq rsp_{\pi.\, P}(N \cup \{m\})$. Moreover, by Lemma 5.3.29 we know that $rsp_{\pi.\, P}(N \cup \{m\}) \subseteq rsp_{*\pi.\, P}(N \cup \{m\})$ and thus, by transitivity, that $rsp_{P'}(N) \subseteq rsp_{*\pi.\, P}(N \cup \{m\})$. Given that $N \subseteq N \cup \{m\}$ and that, by Lemma 5.3.24, we have $\mathsf{bn}(*\pi.\, P \mid P') \subseteq \mathsf{bn}(*\pi.\, P)$, we can apply the result of Lemma 5.3.31 and obtain that $W_2 \subseteq W$. Hence we conclude with the desired result.

We do not consider $a \neq n?m$ because of the similarity with this case.

step case s5-l:

$$\frac{P_1 \xrightarrow{n!m} P_1' \qquad P_2 \xrightarrow{n?m} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} P_1' \mid P_2'}$$

By inductive hypothesis we have that $rsp_{P_1'}(N) \subseteq rsp_{P_1}(N)$ and $rsp_{P_2'}(N) \subseteq rsp_{P_2}(N \cup \{m\})$. Considering that $\mathsf{fn}(P_1) \subseteq N$ and that, by Lemma 5.3.24, $m \in \mathsf{fn}(P_1)$ we know that $N \cup \{m\} = N$ and thus we have $rsp_{P_2'}(N) \subseteq rsp_{P_2}(N)$.

Given that by the definition of bound names $\mathsf{bn}(P_1)$ and $\mathsf{bn}(P_2)$ are subsets of $\mathsf{bn}(P_1 \mid P_2)$, we can say, by Observation 5.3.21, that $rsp_{P_1}(N) \subseteq rsp_{P_1}(N, \mathsf{bn}(P_1 \mid P_2)) = W_{P_1}$ and that $rsp_{P_2}(N) \subseteq rsp_{P_2}(N, \mathsf{bn}(P_1 \mid P_2)) = W_{P_2}$.

At this point, by transitivity, we have that $rsp_{P_1'}(N) \subseteq W_{P_1}$ and $rsp_{P_2'}(N) \subseteq W_{P_2}$. Moreover, by Lemma 5.3.24, we have that $\mathsf{bn}(P_1' \mid P_2') \subseteq \mathsf{bn}(P_1 \mid P_2)$. Therefore we can apply the result of Lemma 5.3.31 and obtain that $W_{P_1'} = rsp_{P_1'}(N, \mathsf{bn}(P_1' \mid P_2')) \subseteq W_{P_1}$ and that $W_{P_2'} = rsp_{P_2'}(N, \mathsf{bn}(P_1' \mid P_2')) \subseteq W_{P_2'}$.

Finally, observing that by Lemma 5.3.32 it holds that $rsp_{P_1 \mid P_2}(N) = W_{P_1} \cup W_{P_2}$ and that $rsp_{P_1' \mid P_2'}(N) = W_{P_1'} \cup W_{P_2'}$, we conclude that $rsp_{P_1 \mid P_2}(N) \subseteq rsp_{P_1' \mid P_2'}(N)$.

We can similarly prove the case s5-r.

step case s7-l:

$$\frac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q}$$

We only consider the case $a = n?m$. By inductive hypothesis we have that $rsp_{P'}(N) \subseteq rsp_P(N \cup \{m\})$. Considering that $\mathsf{bn}(P) \subseteq \mathsf{bn}(P \mid Q)$, by Observation 5.3.21, $rsp_P(N \cup \{m\}) \subseteq rsp_P(N \cup \{m\}, \mathsf{bn}(P \mid Q)) = W_P$. At this point we have that:

1. By transitivity, $rsp_{P'}(N) \subseteq W_P$;
2. By Lemma 5.3.24, $\mathsf{bn}(P' \mid Q) \subseteq \mathsf{bn}(P \mid Q)$;
3. $N \subseteq N \cup \{m\}$.

Therefore we can apply the result of Lemma 5.3.31 and obtain that

$$W_{P'} = rsp_{P'}(N, \mathsf{bn}(P' \mid Q)) \subseteq W_P$$

Keeping in mind that $N \subseteq N \cup \{m\}$ and $\mathsf{bn}(P' \mid Q) \subseteq \mathsf{bn}(P \mid Q)$, thanks to Observation 5.3.21, we can say that $W_{Q_r} = rsp_Q(N, \mathsf{bn}(P' \mid Q)) \subseteq rsp_Q(N \cup \{m\}, \mathsf{bn}(P \mid Q)) = W_{Q_l}$. Now, considering that

1. $W_P' \cup W_{Q_r} \subseteq W_P \cup W_{Q_l}$;
2. by Lemma 5.3.32,
3. $rsp_{P' \mid Q}(N) = W_{P'} \cup W_{Q_r}$. $rsp_{P \mid Q}(N) = W_P \cup W_{Q_l}$

we can conclude with the desired result.

We can similarly prove the case s7-r.

step case s8-l:

$$\frac{M \xrightarrow{a} M'}{M + N \xrightarrow{a} M'}$$

We only consider the case $a = n?m$. By inductive hypothesis $rsp_{M'}(N) \subseteq rsp_M(N \cup \{m\})$. Given that $M \in sp(M + N)$, by Lemma 5.3.29, $rsp_M(N \cup \{m\}) \subseteq rsp_{M+N}(N \cup \{m\})$. At this point, by transitivity, we can conclude with the desired result.

We can similarly prove the cases s8-r and s6.

□

The following lemma, extending the previous results, shows that $rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P))$ is an over-approximation of $rsp_Q(\mathsf{fn}(S) \cup \mathsf{fn}(P))$ for all the internal programs Q of $ids(I[\,P\,],S)$.

Lemma 5.3.34 *Let $I[\,P\,]$ be a well-formed box and S a well-formed system. If $Q \in ids(I[\,P\,],S)$ then $rsp_Q(\mathsf{fn}(S) \cup \mathsf{fn}(P)) \subseteq rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P))$.*

Proof. By induction on the number of transitions necessary in order to reach Q starting from P.

base case: In this case $P = Q$ and there is nothing to be proved.

step case: In this case $P \xrightarrow{a_1} \cdots \xrightarrow{a_n} P_n \xrightarrow{a} Q$. If $a \neq x?m$, by Lemma 5.3.33, we know that $rsp_Q(\mathsf{fn}(S) \cup \mathsf{fn}(P_n)) \subseteq rsp_{P_n}(\mathsf{fn}(S) \cup \mathsf{fn}(P_n))$. If $a = n?m$, again by Lemma 5.3.33, we know that $rsp_Q(\mathsf{fn}(S) \cup \mathsf{fn}(P_n)) \subseteq rsp_{P_n}(\mathsf{fn}(S) \cup \mathsf{fn}(P_n) \cup \{m\})$. Given that by definition of $ids(I[\,P\,],S)$ if $a = n?m$ then $m \in \mathsf{sn}(S) \subseteq \mathsf{fn}(S)$, also in this case we can write $rsp_Q(\mathsf{fn}(S) \cup \mathsf{fn}(P_n)) \subseteq rsp_{P_n}(\mathsf{fn}(S) \cup \mathsf{fn}(P_n))$. At this point we observe that by inductive hypothesis $rsp_{P_n}(\mathsf{fn}(S) \cup \mathsf{fn}(P)) \subseteq rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P))$. Thus we can conclude if we show that $rsp_{P_n}(\mathsf{fn}(S) \cup \mathsf{fn}(P_n)) \subseteq rsp_{P_n}(\mathsf{fn}(S) \cup \mathsf{fn}(P))$ which lets us conclude by definition of *ids*. In order to prove it we verify that $\mathsf{fn}(S) \cup \mathsf{fn}(P_n) \subseteq \mathsf{fn}(S) \cup \mathsf{fn}(P)$. This inclusion follows by Corollary 5.3.25, which tells us that $\mathsf{fn}(P_n) \subseteq \mathsf{fn}(P) \cup \mathsf{fn}(S)$.

□

After this result we can prove that, given a system S and a box $I[\,P\,]$, all the elementary parallel components of the processes belonging to $ids(I[\,P\,],S)$ are in the set $rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P))$.

Corollary 5.3.35 *Let $I[\,P\,]$ be a well-formed box and S a well-formed system. If $Q \in ids(I[\,P\,],S)$ then $epc(Q) \subseteq rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P))$.*

Proof. By Observation 5.3.22 we know that $epc(Q) \subseteq rsp_Q(\mathsf{fn}(S) \cup \mathsf{fn}(P))$. This, by Lemma 5.3.34, lets us immediately conclude with $epc(Q) \subseteq rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P))$.

□

The next theorem states that, if less then k elementary parallel components different from nil appear inside each internal program of $ids(I[\,P\,],S)$, then $ids(I[\,P\,],S)/\equiv_b$ is finite. The key to prove this result is to observe that, from the previous corollary, it immediately follows that the set of elementary parallel components appearing in the processes of $ids(I[\,P\,],S)$ is finite.

Theorem 5.3.36 *Let $I[\,P\,]$ be a well-formed box and S a well-formed system.*

$$\exists k \in \mathbb{N} \mid \forall Q \in ids(I[\,P\,],S), w(Q) \leq k \Rightarrow ids(I[\,P\,],S)/\equiv_p \textit{ has finite cardinality.}$$

Proof. Each internal program Q is congruent to an internal program Q_{nil} obtained by eliminating from Q the elementary parallel components congruent to nil (e.g. Q$=n!m.\,\mathsf{nil} \mid \mathsf{nil}$ and $Q_{\mathsf{nil}} = n!m.\,\mathsf{nil}$). By its definition, Q_{nil} is congruent to Q, has the same width of Q and is made of the same elementary parallel components. Considering that if $Q \in ids(I,[\,P\,],S)$, by hypothesis, its width is less then k and, by Corollary 5.3.35, its elementary parallel component belongs to $rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P))$, we can conclude that the same holds for the congruent internal program Q_{nil}. Therefore, for each $Q \in ids(I,[\,P\,],S)$, it holds that Q_{nil} belongs to the set W of the internal programs whose width is less then k and whose elementary parallel components belong to $rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P)) \setminus \{\mathsf{nil}\}$. Given that, by Lemma 5.3.23, $rsp_P(\mathsf{fn}(S) \cup \mathsf{fn}(P)) \setminus \{\mathsf{nil}\}$ is finite, the set W is finite as well. From this we can conclude that $ids(I[\,P\,],S)/\equiv_p$ is finite because each element of $ids(I[\,P\,],S)$ is congruent to an element of the finite set W.

□

We can now collect the results of Theorem 5.3.13, Corollary 5.3.18 and Theorem 5.3.36 and conclude that if a box $I[\,P\,]$ is in $\mathcal{FF}$ then $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ is a finite object. The result follows because, given a system S such that $\mathsf{sn}(S) \in \mathsf{Name_P}$ and a box $I[\,P\,]$ in $\mathcal{FF}$, we can ensure that, for all the internal programs $Q \in ds(I[\,P\,],S)$, it holds that $w(Q) \leq w(P)$.

Theorem 5.3.37 *Given a well-formed box $I[\,P\,]$ and a well-formed system S, if $I[\,P\,]$ is in $\mathcal{FF}$ and $\mathsf{sn}(S) \subseteq \mathsf{Name_P}$, then $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ is a finite object.*

Proof. The transition system $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ is made of the pair $(ds(I[\,P\,],S)/\equiv_b, \xrightarrow[\mathtt{Box},\equiv]{}^{I[\,P\,],S})$. Thanks to Lemma 5.3.3 we know that if $ds(I[\,P\,],S)/\equiv_b$ is finite then the same holds for $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$. In order to prove the finiteness of $ds(I[\,P\,],S)/\equiv_b$ we prove that of $ds(I'[\,P'\,],S)/\equiv_b$ where $I'[\,P'\,]$ is α-equivalent to $I[\,P\,]$, P' is in $\mathcal{FF}$ and $\mathsf{sub}(I') \cap \mathsf{sn}(S) = \emptyset$. Given that P' is in $\mathcal{FF}$ we know, by Corollary 5.3.18, that $\exists k \in \mathbb{N} \mid \forall Q \in ids(I'[\,P'\,],S), w(Q) \leq k$, in particular $k = w(P')$. This observation lets us apply the result of Theorem 5.3.36 and conclude that $ids(I'[\,P'\,],S)/\equiv_b$ has finite cardinality. Therefore, considering that $\mathsf{sn}(S) \cap \mathsf{sub}(I') = \emptyset$, by Theorem 5.3.13, we conclude that $ds(I'[\,P'\,],S)/\equiv_b$ is finite which implies also the finiteness of $ds(I[\,P\,],S)/\equiv_b$. With that we have proved the thesis.

□

Finally we show that if all the boxes of S are in $\mathcal{FF}$ form, then $\tilde{\Phi}^{S}_{\mathtt{Box},\equiv}$ is finite as well.

Corollary 5.3.38 *Let S be a well-formed system. If the boxes of $getBox(S)$ are in $\mathcal{FF}$ then $\tilde{\Phi}^{S}_{\mathtt{Box},\equiv}$ is a finite object.*

Proof. If all the boxes of $getBox(S)$ are in $\mathcal{FF}$ it immediately follows that $\mathsf{sn}(S) \subseteq \mathsf{Name_P}$. In fact the generating grammar of the process in $\mathcal{FF}$ only allows the definition of output actions that send channel names belonging to $\mathsf{Name_P}$. After this observation we can apply the result of Theorem 5.3.37 and state that $\forall I[\,P\,] \in getBox(S)$, the transition system $\tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ has finite cardinality. Given that $\tilde{\Phi}^{S}_{\mathtt{Box},\equiv} = \bigcup_{I[\,P\,]\in getBox(S)} \tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box},\equiv}$ we can conclude with the thesis. □

5.4 Approximating the Behaviour of Actin Monomers

We apply the obtained results to the actin case study presented in Chapter 4. The system in analysis is a collection of n inactive free monomers of actin.

$$S = (\mathcal{L}(\mathtt{A}_1 \parallel \ldots \parallel \mathtt{A}_n), \emptyset, \emptyset)$$

As a first step, we verify that boxes belonging to $getBox(S)$ generate a finite state space. Here $getBox(S)$ only contains the box representing the inactive free monomer of actin. From Figure 4.1 (i.e. the picture representing the possible configurations assumable by A) we can easily observe that this box assumes a finite number of configurations. However it is worth pointing out that it is not in $\mathcal{FF}$. In fact the box respects all the constraints with the exception of one: it has an output action which fires a replication but is not followed by the nil process. The incriminated action is the output $br!-$ of the first parallel component which is followed by $pr!-$.

In order to redefine the box in $\mathcal{FF}$ and keep the same behaviour (in terms of tangible states), we should remove one of the two outputs from the first parallel component and put it in parallel with the other processes:

$$\begin{aligned} \mathtt{A} \;\triangleq_B\; & (p,\, BI)(b,\, PI) \\ & [\,\mathsf{ch}(r, b, PF).\mathsf{ch}(\infty, p, BF).br!- \mid pr!- \\ & \mid \mathtt{POINT\_REP} \\ & \mid \mathtt{BARBED\_REP}\,] \end{aligned}$$

Even if we can redefine the actin box in order to be in $\mathcal{FF}$, we proceed with the definition of Chapter 4 in order to keep the state space of the box smaller. In this way the output of the analysis is more readable.

In Figure 5.1 we have the graphical representation of $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$. It does not take care of the binding state of the interfaces, therefore an element of it can correspond to more configurations of Figure 3.1. For example the congruence class labelled with B_6 corresponds to three configurations: G, F, and H. Lacking the information regarding the binding state of the interfaces introduces approximation. For example in $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$ we make possible a transition that requires the congruence class $[\![B_4]\!]$ to be bound on its barbed interface. In particular we allow the execution of the action guarded by the condition $(b, \otimes)$ which transforms B_4 in B_7. However, as we can appreciate in Figure 4.1, the barbed interface of configuration D (the only one whose box belongs to $[\![B_4]\!]$), is free. Therefore we have introduced in $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$ a transition that actually cannot happen.

Something similar happens with congruence class B_7. In this case we execute the action guarded by the condition $(b, \odot)$, enabling a transition from B_7 to B_9. In this case, not only do

we generate an impossible transition, but we also generate a congruent class that does not exist in Figure 4.1.

However, even though $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$ is an approximation, it is safe. In fact all the boxes which appear in Figure 4.1 appear in Figure 5.1 and the same holds for the transitions.

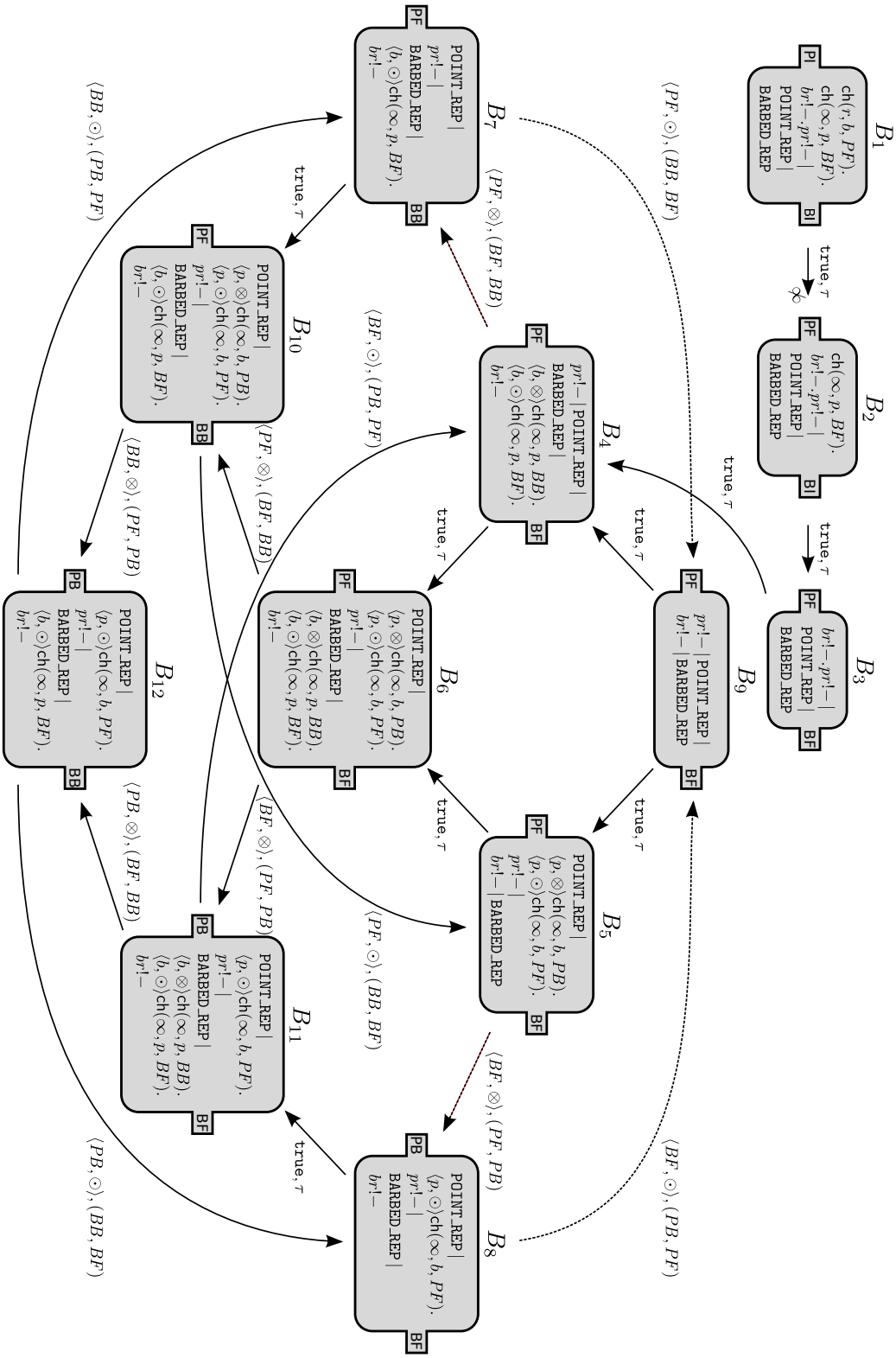


Figure 5.1 – Graphical representation of $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$.

Chapter 6

Control Flow Analysis

In Chapter 5 we introduced an over-approximation of the transition system of the boxes of a system S. In that context we restricted the names which can be received over an interface to the set of sent names of S. We showed that this is a safe over-approximation, however, in some cases, it is extremely imprecise. The objective of this chapter is to use flow logic in order to give a more precise over-approximation of the names which can be received over box interfaces. We will show that the analysis can be used in order to increase the precision of the approximated transition system of the boxes we defined in Chapter 5.

For convenience in the presentation of the analysis we partition Name into two sets: the set of the constants Const and the set of the variables Var. Constants, which range over n, m, etc., represent channel names that can be used for communication. In practice, the scope of a constant will either be one particular box — being used as the name of an interface — or the entire BlenX system — being used as a global name that can be passed between boxes. Bound names introduced by inputs will be called *variables* which range over x, y etc.

We introduce a special sort $\perp$ which is associated with those constants used as global names rather than interfaces of a box. Moreover we define the function

$$\mu : (\mathsf{Sort} \cup \{\perp\}) \to \mathcal{P}(\mathsf{Sort})$$

which, given a sort S, returns the set of all sorts $T \in \mathsf{Sort}$ such that $\alpha_c(S, T) \neq 0$. The special sort $\perp$ has no communication capability with any sort and thus $\mu(\perp) = \emptyset$. Note that the relation induced by μ is symmetric, transitive but not necessarily reflexive.

In our Flow Logic we will make use of the following abstract domains:

1. $\rho : \mathsf{Var} \to \mathcal{P}(\mathsf{Const})$ is the *abstract environment*, which maps a variable to the set of constants it might be bound to. For convenience, we extend the domain of ρ to all names, defining $\rho(n) = \{n\}$ for all $n \in \mathsf{Const}$.

2. $\kappa : \mathcal{P}(\mathsf{Const}) \times \mathsf{Const} \to \mathcal{P}(\mathsf{Const})$ is the *abstract channel environment*, which maps a pair (N, n) to the set of constants that may be communicated over the channel n inside a box $I[\,P\,]$ with $\mathsf{sub}(I) = N$. We use interface subjects instead of labels to refer boxes because new label can be created by system transitions — it is the case when an event fires. Therefore, using labels to refer boxes would make impossible to show the semantic correctness of the analysis.

We associate channel names with box references because the same channel name can appear in more than one box, but the set of names sent over it might be different. This is because the scope of communication over names that are not bound to an interface is limited to within a box. We write $\kappa(n) = \bigcup_{N \in \mathcal{P}(\mathsf{Const})} \kappa(N, n)$ for the set of names sent over n in all boxes.

3. $\tau : \mathsf{Sort} \to \mathcal{P}(\mathsf{Const})$ is the *abstract interface environment* that maps a sort to the set of names it may be associated with.

Table 6.1 gives our flow logic specification which consists of a set of analysis judgments. Specifically, there are five different judgments:

1. $\rho, \kappa, \tau \vdash_{\mathsf{S}} (B, E, \xi)$ — the BlenX system (B, E, ξ) satisfies the analysis predicates ρ, κ, and τ.

2. $\rho, \kappa, \tau \vdash_{\mathsf{E}} E$ — the event E satisfies the analysis predicates ρ, κ, and τ.

3. $\rho, \kappa, \tau \vdash_{\mathsf{B}} B$ — the bio-process B satisfies the analysis predicates ρ, κ, and τ.

4. $\rho, \kappa, \tau \vdash_{\mathsf{P}} P'$ — the process P' inside the box $I[\,P\,]$ satisfies the analysis predicates ρ, κ, and τ.

5. $\rho, \kappa, \tau \vdash_{\mathsf{A}}^{N} \pi$ — the action π inside the box $I[\,P\,]$ with $\mathsf{sub}(I) = N$ satisfies the analysis predicates ρ, κ, and τ.

It is important to remember that the analysis predicates are *global*, meaning that the same analysis predicates must constitute a valid analysis result for every component of the system. In the terminology of flow logic, we have what is known as a *verbose* specification.

Let us now detail the clauses of Table 6.1. The [SYS] clause states that the analysis predicates ρ, κ, and τ are a valid analysis result for a BlenX system (B, E, ξ), if they are valid for the bio-process B and the events E. The analysis is independent of the environment ξ, since it considers only the communication affinity between sorts when determining whether communication is possible, and not the bound state of the interface.

The [EV] clause considers the set of events in the BlenX system, which have the general form $I[\,P\,] \blacktriangleright_r B$. Such an event, if applied to a system, is responsible of the creation of the bio-process B. For this reason we require that B (which is decorated with labels in order to be compatible with clause [BOX]) satisfies the analysis predicates ρ, κ, and τ. We do not do the same for the box $I[\,P\,]$ because the event cannot be the cause of the instantiation of such a box in the system.

The only non-trivial clause of the $\rho, \kappa, \tau \vdash_{\mathsf{B}} B$ judgement is [BOX]. In addition to requiring that the analysis judgement constitute a valid solution for the process P inside the box with interface set I such that $\mathsf{sub}(I) = N$, we additionally need to record in τ the initial sort of each interface in I.

The [CON] clause checks that the capability M satisfies the analysis judgment but only if it is possible for the guard C to evaluate to true. If we are certain that C cannot evaluate to true, then the action can never execute, and so it cannot affect the state of the system. We determine this using the function $\mathsf{check}_{\rho,\tau} : \mathsf{Cond} \to \mathcal{P}(\mathsf{Bool})$ which is defined in Table 6.2. Here, $\neg^\sharp$ and $\wedge^\sharp$ are the set versions of the standard Boolean connectives, defined such that $\neg^\sharp\{\mathsf{true}, \mathsf{false}\} = \{\mathsf{true}, \mathsf{false}\}$, $\{\mathsf{true}, \mathsf{false}\} \wedge^\sharp \{\mathsf{true}, \mathsf{false}\} = \{\mathsf{true}, \mathsf{false}\}$, $\neg^\sharp \emptyset = \emptyset$ and for all $x \in$

[SYS]	$\rho,\kappa,\tau \vdash_{\mathsf{S}} (B,E,\xi)$	iff	$\rho,\kappa,\tau \vdash_{\mathsf{B}} B \wedge \rho,\kappa,\tau \vdash_{\mathsf{E}} E \wedge$ $\forall n \in \mathsf{fn}((B,E,\xi)), n \in \tau(\bot)$
[EV]	$\rho,\kappa,\tau \vdash_{\mathsf{E}} E$	iff	$\forall I[\,P\,] \blacktriangleright_r B \in E : \rho,\kappa,\tau \vdash_{\mathsf{B}} \mathcal{L}(\gamma,B)$
[PAR-B]	$\rho,\kappa,\tau \vdash_{\mathsf{B}} B_1 \parallel B_2$	iff	$\rho,\kappa,\tau \vdash_{\mathsf{B}} B_1 \wedge \rho,\kappa,\tau \vdash_{\mathsf{B}} B_2$
[BOX]	$\rho,\kappa,\tau \vdash_{\mathsf{B}} I[\,P\,]_\gamma$	iff	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{\mathsf{sub}(I)} P \wedge \forall (a,\,S)^r \in I, a \in \tau(S)$
[CON]	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} \langle C\rangle M$	iff	$\{\mathsf{true}\} \subseteq (\mathsf{check}_{\rho,\tau}(C)) \;\Rightarrow$ $\rho,\kappa,\tau \vdash_{\mathsf{A}}^{N} \pi \wedge \rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} M$
[REP]	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} *\pi.P'$	iff	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} \pi.P'$
[ACT]	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} \pi.P'$	iff	$\rho,\kappa,\tau \vdash_{\mathsf{A}}^{N} \pi \wedge \rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} P'$
[PAR]	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} P_1 \mid P_2$	iff	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} P_1 \wedge \rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} P_2$
[SUM]	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} M_1 + M_2$	iff	$\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} M_1 \wedge \rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} M_2$
[CH]	$\rho,\kappa,\tau \vdash_{\mathsf{A}}^{N} \mathsf{ch}(r,v,T)$	iff	$\rho(v) \subseteq \tau(T)$
[OUT]	$\rho,\kappa,\tau \vdash_{\mathsf{A}}^{N} v!u$	iff	$\forall n \in \rho(v) : \rho(u) \subseteq \kappa(N,n)$
[IN]	$\rho,\kappa,\tau \vdash_{\mathsf{A}}^{N} v?y$	iff	$\forall n \in \rho(v) : \kappa(N,n) \subseteq \rho(y) \wedge$ $(\forall T \in \mathsf{Sort} : \rho(v) \cap \tau(T) \neq \emptyset \;\Rightarrow$ $(\forall n \in \tau^*(\mu(T)) : \kappa(n) \subseteq \rho(y)))$

Table 6.1 – Flow logic specification: $\rho,\kappa,\tau \vdash_{\mathsf{S}} (B,E,\xi)$, $\rho,\kappa,\tau \vdash_{\mathsf{E}} E$, $\rho,\kappa,\tau \vdash_{\mathsf{B}} B$, $\rho,\kappa,\tau \vdash_{\mathsf{P}}^{N} P$, and $\rho,\kappa,\tau \vdash_{\mathsf{A}}^{N} \pi$.

$\mathcal{P}(\mathsf{Bool})$, $\{\mathsf{true}\} \wedge^\sharp x = x$, $\{\mathsf{false}\} \wedge^\sharp x = \{\mathsf{false}\}$ and $\neg^\sharp \emptyset \wedge^\sharp x = \emptyset$. Finally, $\tau^* : \mathcal{P}(\mathsf{Sort}) \to \mathcal{P}(\mathsf{Int})$ is the extension of τ onto sets of sorts: $\tau^*(\mathcal{S}) = \bigcup_{S\in\mathcal{S}} \tau(S)$.

When the condition C tests whether an interface is bound or free we return $\{\mathsf{true},\mathsf{false}\}$, since this could be evaluated to either true or false. When the condition tests whether an interface has a particular sort T, we return $\{\mathsf{true}\}$ if the interface can only ever have type T, $\{\mathsf{false}\}$ if it can never have type T, $\emptyset$ if we are dealing with a variable that cannot be associated with any name and $\{\mathsf{true},\mathsf{false}\}$ otherwise.

The remaining non trivial clauses in Table 6.1 concern actions π. Clause [CH] handles the case $\pi = \mathsf{ch}(r,v,T)$, ensuring that all the interface names that the variable v could take are associated with the sort T. The [OUT] clause records all possible values of the output u in the abstract channel environment for each value of the channel v in the box with interface set I,

$$
\begin{array}{lcl}
\mathsf{check}_{\rho,\tau}(\mathsf{true}) & = & \{\mathsf{true}\} \\
\mathsf{check}_{\rho,\tau}(\neg C) & = & \neg^\sharp(\mathsf{check}_{\rho,\tau}(C)) \\
\mathsf{check}_{\rho,\tau}(C_1 \wedge C_2) & = & \mathsf{check}_{\rho,\tau}(C_1) \wedge^\sharp \mathsf{check}_{\rho,\tau}(C_2) \\
\mathsf{check}_{\rho,\tau}((x,k)) & = & \begin{cases} \emptyset & \text{if } \rho(x) = \emptyset \\ \{\mathsf{true},\mathsf{false}\} & \text{otherwise} \end{cases} \\
\mathsf{check}_{\rho,\tau}((x,T)) & = & \begin{cases} \emptyset & \text{if } \tau^*(\mathsf{Sort}) \cap \rho(x) = \emptyset \\ \{\mathsf{true}\} & \text{if } \tau(T) \cap \rho(x) \neq \emptyset \wedge \\ & \qquad \tau^*(\mathsf{Sort} \setminus T) \cap \rho(x) = \emptyset \\ \{\mathsf{false}\} & \text{if } \tau(T) \cap \rho(x) = \emptyset \wedge \\ & \qquad \tau^*(\mathsf{Sort} \setminus T) \cap \rho(x) \neq \emptyset \\ \{\mathsf{true},\mathsf{false}\} & \text{otherwise} \end{cases}
\end{array}
$$

Table 6.2 – Definition of check.

such that $\mathsf{sub}(I) = N$.

The [IN] clause is more complex. The first line is the dual of the [OUT] clause — we record all values that can be sent on v in the box with interface set I such that $\mathsf{sub}(I) = N$ in the abstract environment of the variable y. We now consider the case when v is an interface name, and we receive a value from an output which takes place in a different box. Here we require $(\rho(v) \cap \tau(T) \neq \emptyset)$ for every sort T that v can possibly take. Finally the abstract environment of the variable y includes all the values that can be sent on all the names that can possibly have a sort that has communication affinity with T.

6.1 Semantic Correctness

Here we prove that if an analysis is valid for a system S and this system evolve to S', the same analysis is valid also for S'.

One of the intermediate results requires to show that the analysis gives the same answer for congruent systems. This causes issues in the case of systems congruent (or better equal) because of the application of α-renaming. In fact the analysis results ρ, κ and τ are strictly linked with the names which appear in the analysed systems. To lose the link with these names through α-renaming means breaking the analysis results. To solve this problem (as in [82]) we replace the concept of α-renaming with that of disciplined α-renaming and we introduce *canonical names*. Disciplined α-renaming means that when we substitute a name n we cannot use whatever name but we have to choose a name m such that $\lfloor n \rfloor = \lfloor m \rfloor$, where $\lfloor n \rfloor$ denotes the canonical name of n. This is the same of dividing names in equivalent classes and to allow substitutions only between names of the same class. We assume that there are infinitely many classes and infinitely many names for each class. We write $\lfloor S \rfloor$ for the system where all the variables and the interface names are replaced by their canonical counterparts. Similarly we extend this syntax to bio-processes, processes and interfaces. In the following we formally define the disciplined α-renaming:

$$b \notin \mathsf{fn}(P_1) \cup \mathsf{sub}(I) \Rightarrow (a,\, S)^r \cup I[\, P_1 \,] = (b,\, S)^r \cup I[\, P_1\{b/a\} \,] \wedge \lfloor a \rfloor = \lfloor b \rfloor$$

We state now some auxiliary results in order to show the semantic correctness of our analysis.

In the next lemma, we show that, if τ is a valid abstract interface environment for the set of interfaces I and the names of the condition C are constants, the results of $\wr C \int_I^*$ and $\mathsf{check}_{\rho,\tau}(\lfloor C \rfloor)$ are related.

Lemma 6.1.1 *Let I be an interface set and C a condition. If $\forall(\lfloor a \rfloor, T)^r \in \lfloor I \rfloor, \lfloor a \rfloor \in \tau(T)$ and $\mathsf{fn}(C) \subseteq \mathsf{Const}$ then we have:*

1. *if $\wr C \int_I^* = \mathsf{true}$ then $\mathsf{true} \subseteq \mathsf{check}_{\rho,\tau}(\lfloor C \rfloor)$.*
2. *if $\wr C \int_I^* = \mathsf{false}$ then $\mathsf{false} \subseteq \mathsf{check}_{\rho,\tau}(\lfloor C \rfloor)$.*
3. *if $\wr C \int_I^* \notin \{\mathsf{true}, \mathsf{false}\}$ then $\mathsf{check}_{\rho,\tau}(\lfloor C \rfloor) = \{\mathsf{true}, \mathsf{false}\}$.*

Sketch. By induction on the structure of C.

□

With Lemma 6.1.2 we point out that if a constant $\lfloor x \rfloor$ can be associated with a variable $\lfloor y \rfloor$, then, the set of values $\lfloor C\{x/y\} \rfloor$ can possibly evaluate to, is smaller than that of $\lfloor C \rfloor$.

Lemma 6.1.2 *Let $x \in \mathsf{Const}$, $y \in \mathsf{Var}$, ρ an abstract environment, τ an abstract interface environment and C a condition. If $\lfloor x \rfloor \in \rho(\lfloor y \rfloor)$ then $\mathsf{check}_{\rho,\tau}(\lfloor C\{x/y\} \rfloor) \subseteq \mathsf{check}_{\rho,\tau}(\lfloor C \rfloor)$.*

Proof. The proof is by induction on the structure of the condition C. We only consider the interesting cases.

case (b, T) : if $b \neq y$ it is straightforward.

We consider the case $b = y$. We start observing that if $\lfloor x \rfloor \in \rho(\lfloor y \rfloor)$ then $\lfloor x \rfloor \in \mathsf{Const}$ and thus $\rho(\lfloor x \rfloor) = \lfloor x \rfloor$. This implies that $\rho(\lfloor x \rfloor) \subseteq \rho(\lfloor y \rfloor)$. Therefore $\mathsf{check}_{\rho,\tau}((\lfloor y \rfloor, S))$ can evaluate to four possible values.

If $\mathsf{check}_{\rho,\tau}((\lfloor y \rfloor, T)) = \{\mathsf{true}\}$. Given that $\rho(\lfloor x \rfloor) \subseteq \rho(\lfloor y \rfloor)$ and $\tau^*(\mathsf{Sort} \setminus \{T\}) \cap \rho(\lfloor y \rfloor) = \emptyset$ we can state that $\tau^*(\mathsf{Sort} \setminus \{T\}) \cap \rho(\lfloor x \rfloor) = \emptyset$. This is enough for concluding that $\mathsf{check}_{\rho,\tau}((\lfloor x \rfloor, T))$ evaluates to $\emptyset$ or $\{\mathsf{true}\}$ and thus to ensure that $\mathsf{check}_{\rho,\tau}((\lfloor x \rfloor, T)) \subseteq \mathsf{check}_{\rho,\tau}((\lfloor y \rfloor, T))$.

If $\mathsf{check}_{\rho,\tau}((\lfloor y \rfloor, T)) = \{\mathsf{false}\}$ we can obtain that $\tau(T) \cap \rho(x) = \emptyset$ and thus $\mathsf{check}_{\rho,\tau}((\lfloor x \rfloor, T))$ evaluates to $\emptyset$ or $\{\mathsf{false}\}$. This, for the same reasons of the previous case, let us conclude with the desired result.

If $\mathsf{check}_{\rho,\tau}((\lfloor y \rfloor, T)) = \{\mathsf{true}, \mathsf{false}\}$, whatever is the evaluation of $\mathsf{check}_{\rho,\tau}((\lfloor x \rfloor, T))$, the thesis of the theorem is verified.

If $\mathsf{check}_{\rho,\tau}((\lfloor y \rfloor, T)) = \emptyset$ it is straightforward to ensure that $\tau^*(\mathsf{Sort} \setminus \{T\}) \cap \rho(\lfloor x \rfloor) = \tau(T) \cap \rho(x) = \emptyset$, and thus, as required, it follows that $\mathsf{check}_{\rho,\tau}((\lfloor x \rfloor, T)) = \emptyset$.

case $\neg C$: we can similarly handle all the possible values the condition $\neg C$ can evaluate to. For this reason we consider only the case where $\mathsf{check}_{\rho,\tau}(\neg^\sharp C) = \{\mathsf{true}\}$. Observing that $\mathsf{check}_{\rho,\tau}(\neg^\sharp C) = \neg^\sharp \mathsf{check}_{\rho,\tau}(C) = \{\mathsf{true}\}$, and thus that $\mathsf{check}_{\rho,\tau}(C) = \{\mathsf{false}\}$, we can apply the inductive hypothesis and obtain that $\mathsf{check}_{\rho,\tau}(C\{x/y\}) \in \{\{\mathsf{false}\}, \emptyset\}$. Therefore we conclude with $\neg^\sharp \mathsf{check}_{\rho,\tau}(C\{x/y\}) \in \{\{\mathsf{true}\}, \emptyset\}$.

□

Here we state that, if ρ, κ, τ is a valid solution for the process $\lfloor P \rfloor$, the same holds for $P\{x/y\}$ under the condition that $\lfloor y \rfloor$ can possibly assume the channel name $\lfloor x \rfloor$ as a value.

Lemma 6.1.3 *Let $x \in \mathsf{Const}$, $y \in \mathsf{Var}$, ρ an abstract environment, τ an abstract interface environment and C a condition. If $\lfloor x \rfloor \in \rho(\lfloor y \rfloor)$ and $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P \rfloor$ then $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P\{x/y\} \rfloor$.*

Proof. We proceed by induction on the structure of P. We only consider the most interesting cases.

case $\langle C \rangle M$: we have two cases. In the first case $\mathsf{check}_{\rho,\tau}(\lfloor C \rfloor) \subseteq \{\mathsf{false}\}$. We conclude by Lemma 6.1.2, which ensures that the evaluation of $\lfloor C\{x/y\} \rfloor$ is a subset of $\{\mathsf{false}\}$ as well. In the second case $\mathsf{check}_{\rho,\tau}(\lfloor C \rfloor) \supseteq \{\mathsf{true}\}$. If check evaluates $\lfloor C\{x/y\} \rfloor$ to $\{\mathsf{false}\}$ the result follows straightforward; otherwise $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} M\{x/y\}$ follows by inductive hypothesis.

case $v!z.\, P$: we have four cases:

case $v \neq x$ and $z \neq y$: in this case $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor v!z.\, P \rfloor$ that implies $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor v!z \rfloor$ and $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P \rfloor$. Given that $(v!z.\, P)\{x/y\} = v!z.\, P\{x/y\}$ and that, by inductive hypothesis on P, we get $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P\{x/y\} \rfloor$, we have all the elements for concluding with $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor v!z.\, P\{x/y\} \rfloor$.

case $v = y$ and $z \neq y$: in this case $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor y!z.\, P \rfloor$ that implies $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor y!z \rfloor$ and $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P \rfloor$. $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor y!z \rfloor$ implies that $\forall n \in \rho(\lfloor y \rfloor), \rho(\lfloor z \rfloor) \in \kappa(N, n)$. Considering that if $\lfloor x \rfloor \in \rho(\lfloor y \rfloor)$ then $\lfloor x \rfloor \in \mathsf{Const}$, we know that $\rho(\lfloor x \rfloor) = \lfloor x \rfloor$. These observations imply that $\rho(\lfloor x \rfloor) \subseteq \rho(\lfloor y \rfloor)$ and thus that $\forall n \in \rho(\lfloor x \rfloor), \rho(\lfloor z \rfloor) \in \kappa(N, n)$. This let us conclude that $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor x!z \rfloor$ and, given that as before, by inductive hypothesis, we have that $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P\{x/y\} \rfloor$, we obtain the desired result.

case $v \neq y$ and $z = y$: in this case $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor v!y.\, P \rfloor$ that implies $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor v!y \rfloor$ and $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P \rfloor$. $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor v!y \rfloor$ implies that $\forall n \in \rho(\lfloor v \rfloor), \rho(\lfloor y \rfloor) \subseteq \kappa(N, n)$. Given $\rho(\lfloor x \rfloor) \subseteq \rho(\lfloor y \rfloor)$, $\forall n \in \rho(\lfloor v \rfloor), \rho(\lfloor x \rfloor) \subseteq \kappa(N, n)$. Thus, exploiting the inductive hypothesis on P, we can conclude with $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor v!x.\, P\{x/y\} \rfloor$.

case $v = y$ and $z = y$: in this case $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor y!y.\, P \rfloor$ that implies $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor y!y \rfloor$ and $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P \rfloor$. $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor y!y \rfloor$ implies that $\forall n \in \rho(\lfloor y \rfloor), \rho(\lfloor y \rfloor) \subseteq \kappa(N, n)$. Given that $\rho(\lfloor x \rfloor) \subseteq \rho(\lfloor y \rfloor)$, we immediately obtain that $\forall n \in \rho(\lfloor y \rfloor), \rho(\lfloor y \rfloor) \subseteq \kappa(N, n)$ and we can conclude with the desired result as in the previous case.

case $v?z.\, P$: also in this case we have four cases:

sub-case $v \neq y$ and $z \neq y$: we handle this case as we have done for $v!z.\, P$. We only need to apply the inductive hypothesis on the process P.

sub-case $v = y$ and $z \neq y$: in this case $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor y?z.\, P \rfloor$ which implies $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor y?z \rfloor$ and $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor P \rfloor$. $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} \lfloor y?z \rfloor$ implies $\forall T : \rho(\lfloor y \rfloor) \cap \tau(T) \neq \emptyset \Rightarrow (\forall a \in \tau^*(\mu(T)), \kappa(a) \subseteq \rho(\lfloor z \rfloor))$ and $\forall w \in \rho(\lfloor y \rfloor), \kappa(N, w) \subseteq \rho(\lfloor z \rfloor)$. Considering that $\rho(\lfloor x \rfloor) \subseteq \rho(\lfloor y \rfloor)$ we know $\forall T : \rho(\lfloor x \rfloor) \cap \tau(T) \neq \emptyset \Rightarrow \rho(\lfloor y \rfloor) \cap \tau(T) \neq \emptyset$. This let us conclude $\forall T : \rho(\lfloor x \rfloor) \cap \tau(T) \neq \emptyset \Rightarrow (\forall a \in \tau^*(\mu(T)), \kappa(a) \subseteq \rho(\lfloor z \rfloor))$. From $\rho(\lfloor x \rfloor) \subseteq \rho(\lfloor y \rfloor)$ and $\forall w \in \rho(\lfloor y \rfloor), \kappa(N, w) \subseteq \rho(\lfloor z \rfloor)$, is is easy to obtain that $\forall w \in \rho(\lfloor x \rfloor), \kappa(N, w) \subseteq \rho(\lfloor z \rfloor)$.

Now, in conjunction with the application of the inductive hypothesis on P, we have all the necessary in order to state $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} \lfloor x?z.\, P\{x/y\} \rfloor$.

sub-case $v \neq y$ and $z = y$: in this case we immediately conclude by observing that $(v?y.\, P)\{x/y\} = v?y.\, P$.

sub-case $v = y$ and $z = y$: here we have that $(y?y.\, P)\{x/y\} = x?y.\, P$. In order to obtain the desired result we have to show that $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} x?y$ and that $\rho, \kappa, \tau \vdash^N_{\mathsf{P}} P$. The second statement follows immediately by hypothesis, for the first one we can proceed as we have shown $\rho, \kappa, \tau \vdash^N_{\mathsf{A}} x?z$ in the case $v = y$ and $z \neq y$.

case $\mathbf{ch}(v, S).\, P$: we consider only the case where $v = y$. Also this proof is based on the observation that $\rho(\lfloor x \rfloor) \subseteq \rho(\lfloor y \rfloor)$. In fact this observation, considering that by hypothesis it follows that $\rho(\lfloor y \rfloor) \subseteq \tau(S)$, implies $\rho(\lfloor x \rfloor) \subseteq \tau(S)$. In this way, exploiting the inductive hypothesis on P, we can conclude with the desired result.

□

With the next lemma we prove a subject reduction result limited to transitions involving boxes. The thesis of the lemma is divided in three points. They handle output, input and change actions, respectively.

Lemma 6.1.4 *Let $x \in \mathsf{Const}$, $y \in \mathsf{Var}$, ρ, τ, κ an analysis result, $I[\,P\,]_\gamma$ a well-formed box, N a set of canonical names. If $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\,P\,]_\gamma \rfloor$ and $I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$ then we have:*

1. *If $a = k!t$ then $k = n$ or $k = T_1$ with $(n, T_1)^{r_1} \in I$ and $t = m$ or $t = T_2$ with $(m, T_2)^{r_2} \in I$ and $\lfloor m \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor n \rfloor)$ and $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I'[\,P'\,]_\gamma \rfloor$.*

2. *If $a = k?t$ then $k = n$ or $k = T_1$ with $(n, T_1)^{r_1} \in I$ and $t = m$ or $t = T_2$ with $(m, T_2)^{r_2} \in I$ and we have:*

   (a) if $\lfloor m \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor n \rfloor)$ then $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I'[\,P'\,]_\gamma \rfloor$.

   (b) if $\exists U \in \mathsf{Sort} : U \in \mu(T_1)$ and $n' \in \tau(U)$ and $m \in \kappa(n')$ then $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I'[\,P'\,]_\gamma \rfloor$

3. *If $a \in \{(T, U), \tau\}$ then $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I'[\,P'\,]_\gamma \rfloor$.*

Proof. By induction on the length of the derivation of $I[\,P\,]_\gamma \xrightarrow{C,a}_r I'[\,P'\,]_\gamma$ (rules of Table 3.4). We consider only the most interesting cases.

base case r1: if $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\,n!m.\, P\,]_\gamma \rfloor$ then $\rho, \kappa, \tau \vdash^{\mathsf{sub}(\lfloor I \rfloor)}_{\mathsf{A}} \lfloor n!m \rfloor$ and $\rho, \kappa, \tau \vdash^{\mathsf{sub}(\lfloor I \rfloor)}_{\mathsf{P}} \lfloor P \rfloor$ and $\forall (\lfloor a \rfloor, S)^r \in \lfloor I \rfloor, \lfloor a \rfloor \in \tau(S)$. This is enough to conclude, indeed $\rho, \kappa, \tau \vdash^{\mathsf{sub}(\lfloor I \rfloor)}_{\mathsf{A}} \lfloor n!m \rfloor$ implies $\lfloor m \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor n \rfloor)$, and $\rho, \kappa, \tau \vdash^{\mathsf{sub}(\lfloor I \rfloor)}_{\mathsf{P}} \lfloor P \rfloor$ and $\forall (\lfloor a \rfloor, S)^r \in \lfloor I \rfloor, \lfloor a \rfloor \in \tau(S)$ imply $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\,P\,]_\gamma \rfloor$.

base case r2: if $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\,n?y.\, P\,]_\gamma \rfloor$ then $\rho, \kappa, \tau \vdash^{\mathsf{sub}(\lfloor I \rfloor)}_{\mathsf{A}} \lfloor n?y \rfloor$ and $\rho, \kappa, \tau \vdash^{\mathsf{sub}(\lfloor I \rfloor)}_{\mathsf{P}} \lfloor P \rfloor$ and $\forall (\lfloor a \rfloor, S)^r \in \lfloor I \rfloor, \lfloor a \rfloor \in \tau(S)$. If $a = k?t$ we separately consider the cases (a) and (b):

(a) $\rho, \kappa, \tau \vdash_{\mathsf{A}}^{\mathsf{sub}(\lfloor I \rfloor)} \lfloor n?y \rfloor$ implies that $\kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor n \rfloor) \subseteq \rho(\lfloor y \rfloor)$ which, given that by hypothesis $\lfloor m \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor n \rfloor)$, let us state that $m \in \rho(\lfloor y \rfloor)$. This last observation together with $\rho, \kappa, \tau \vdash_{\mathsf{P}}^{\mathsf{sub}(\lfloor I \rfloor)} \lfloor P \rfloor$ make possible to apply the result of Lemma 6.1.3 and to obtain that $\rho, \kappa, \tau \vdash_{\mathsf{P}}^{\mathsf{sub}(\lfloor I \rfloor)} \lfloor P\{^{m}/_{y}\} \rfloor$. Now we have all the necessary in order to conclude with the desired result $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\, P\{^{y}/_{m}\}\,]_{\gamma} \rfloor$.

(b) $\rho, \kappa, \tau \vdash_{\mathsf{A}}^{\mathsf{sub}(\lfloor I \rfloor)} \lfloor n?y \rfloor$ implies that if $U \in \mu(T_1)$ and $n' \in \tau(U)$ then $\kappa(\lfloor n' \rfloor) \subseteq \rho(\lfloor y \rfloor)$. Observing that $\kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor n' \rfloor) \subseteq \kappa(\lfloor n' \rfloor)$ and that, by hypothesis, $\lfloor m \rfloor \in \kappa(\lfloor n' \rfloor)$ we conclude that $\lfloor m \rfloor \in \rho(\lfloor y \rfloor)$ and therefore we can proceed as in the previous case.

base case r3: if $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor \{(a,\, U)^{r_1}\} \cup I[\, \mathsf{ch}(r, a, T).\, P\,]_{\gamma} \rfloor$ then $\forall (\lfloor b \rfloor,\, S)^{r} \in \lfloor I \rfloor, \lfloor b \rfloor \in \tau(S)$ and $\rho, \kappa, \tau \vdash_{\mathsf{A}}^{\lfloor a \rfloor \cup \mathsf{sub}(\lfloor I \rfloor)} \mathsf{ch}(r, \lfloor a \rfloor, T)$ and $\rho, \kappa, \tau \vdash_{\mathsf{P}}^{\lfloor a \rfloor \cup \mathsf{sub}(\lfloor I \rfloor)} P$. Moreover we know that if $\rho, \kappa, \tau \vdash_{\mathsf{A}}^{\lfloor a \rfloor \cup \mathsf{sub}(\lfloor I \rfloor)} \mathsf{ch}(r, \lfloor a \rfloor, T)$ then $a \in \tau(T)$ and thus $\rho, \kappa, \tau \vdash_{\mathsf{B}} \{(a,\, T)^{r_1}\} \cup I[\, P\,]_{\gamma}$.

step case r5-l: for the sake of clearness we suppose $k = n$ e $t = m$. From $\rho, \kappa, \tau \vdash_{\mathsf{B}} I[\, P_1 \mid P_2\,]_{\gamma}$ we can easily obtain that $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\, P_i\,]_{\gamma} \rfloor$ for $i \in \{1, 2\}$. This lets us apply the inductive hypothesis to $I[\, P_1\,]_{\gamma} \xrightarrow{C_1, n!m}_{0} I'[\, P_1'\,]_{\gamma}$ and obtain that $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I'[\, P_1'\,]_{\gamma} \rfloor$ and $\lfloor m \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor n \rfloor)$. The last observation makes possible to apply the inductive hypothesis to $I[\, P_2\,]_{\gamma} \xrightarrow{C_1, n?m}_{0} I'[\, P_2'\,]_{\gamma}$ and obtain that $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I'[\, P_2'\,]_{\gamma} \rfloor$. From $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I'[\, P_i'\,]_{\gamma} \rfloor$ we have that $\rho, \kappa, \tau \vdash_{\mathsf{P}}^{\mathsf{sub}(I')} \lfloor P_i' \rfloor$ per $i \in [1, 2]$ and $\forall (a,\, S)^{r} \in I', a \in \tau(S)$. These let us conclude with the desired result $\rho, \kappa, \tau \vdash_{\mathsf{B}} I'[\, P_1' \mid P_2'\,]_{\gamma}$.

step case r6: by hypothesis we know that $\wr C \int_I^* \wedge C' \neq \mathsf{false}$, which implies $\wr C \int_I^* \neq \mathsf{false}$. Considering that if $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\, \langle C \rangle M\,]_{\gamma} \rfloor$ then $\forall (\lfloor a \rfloor,\, S)^{r} \in \lfloor I \rfloor, \lfloor a \rfloor \in \tau(S)$, by Lemma 6.1.1, $\wr C \int_I^* \neq \mathsf{false}$ implies that $\{\mathsf{true}\} \subseteq \mathsf{check}_{\rho,\tau}(\lfloor C \rfloor)$. This implies that if $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\, \langle C \rangle M\,]_{\gamma} \rfloor$ then $\rho, \kappa, \tau \vdash_{\mathsf{P}}^{\mathsf{sub}(\lfloor I \rfloor)} \lfloor M \rfloor$. Hence, we have the necessary to state that $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor I[\, M\,]_{\gamma} \rfloor$ and to conclude by applying the inductive hypothesis to $I[\, M\,]_{\gamma} \xrightarrow{C', a}_{r} I'[\, M'\,]_{\gamma}$.

□

Now we extend the result of the previous lemma to transitions involving systems. Again, we separately consider transitions generated by different actions. The three points listed in the thesis handle transitions fired by input and output actions over an interface, and by internal actions, respectively.

Lemma 6.1.5 *Let S be a well-formed system and ρ, κ, τ an analysis result. If $S \xrightarrow{l}_{r} S'$ and $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor S \rfloor$ then we have:*

1. *If $l = (\gamma, T!n)$ then $get(\gamma, S) = I[\, P\,]_{\gamma}$ and $(b,\, T)^{r_1} \in I$ and $\lfloor n \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), b)$ and $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor S' \rfloor$.*
2. *If $l = (\gamma, T?n)$ then $get(\gamma, S) = I[\, P\,]_{\gamma}$ and $(b,\, T)^{r_1} \in I$ and (if $\exists U \in \mathsf{Sort} : U \in \mu(T)$ and $m \in \tau(U)$ and $n \in \kappa(m)$ then $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor S' \rfloor$).*
3. *If $l \in \{(\gamma, T), \tau\}$ then $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor S' \rfloor$.*

Proof. By induction on the length of the derivation of $S \xrightarrow{l}_{r} S'$. We consider only the interesting cases.

step case r16-l: if $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor (B_1 \parallel B_2, E, \xi) \rfloor$ then $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor B_i \rfloor$ for $i \in [1,2]$ and $\rho, \kappa, \tau \vdash_{\mathsf{E}} \lfloor E \rfloor$ and $\forall n \in \mathsf{fn}((B_1 \parallel B_2, E, \xi)) \lfloor n \rfloor \in \kappa(\bot)$. Considering that $\mathsf{fn}((B_i, E, \xi)) \subseteq \mathsf{fn}((B_1 \parallel B_2, E, \xi))$ we know that $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor (B_i, E, \xi) \rfloor$ for $i \in [1,2]$. This let us apply the inductive hypothesis to $(B_1, E, \xi) \xrightarrow{(\gamma_1, T!n)}_0 (B_1', E, \xi)$ and obtain that $get(\gamma_1, (B_1, E, \xi)) = I_1[\, P_1 \,]_{\gamma_1}$ and $\exists b_1 : (b_1, T_1)^{r_1} \in I_1$ and $\lfloor n \rfloor \in \kappa(\mathsf{sub}(\lfloor I_1 \rfloor), b_1)$ and $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor (B_1', E, \xi) \rfloor$. Moreover, given that $\alpha_c(T_1, T_2) \neq 0$, we know that $T_1 \in \mu(T_2)$. These observations, together with the application of the inductive hypothesis to $(B_2, E, \xi) \xrightarrow{(\gamma_2, T?n)}_0 (B_2', E, \xi)$, let us conclude that $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor (B_2', E, \xi) \rfloor$. At this point we have that $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor B_i' \rfloor$ for $i \in [1,2]$ and $\rho, \kappa, \tau \vdash_{\mathsf{E}} \lfloor E \rfloor$ and, considering that by Corollary 5.1.4 we have that $\mathsf{fn}((B_1' \parallel B_2', E, \xi)) \subseteq \mathsf{fn}((B_1 \parallel B_2, E, \xi))$ and thus that $\forall n \in \mathsf{fn}((B_1' \parallel B_2', E, \xi)), \lfloor n \rfloor \in \kappa(\bot)$, we can conclude that $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor (B_1' \parallel B_2', E, \xi) \rfloor$.

base case r17: if $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor (I[\, P \,]_\gamma, E, \xi) \rfloor$ then $\rho, \kappa, \tau \vdash_{\mathsf{E}} \lfloor E \rfloor$ which implies $\forall I[\, P \,] \blacktriangleright_{r'} B \in E$, $\rho, \kappa, \tau \vdash_{\mathsf{B}} \lfloor \mathcal{L}(\gamma, B) \rfloor$. These observations, considering that, by Corollary 5.1.4, we know that $\mathsf{fn}((I[\, P \,]_\gamma, E, \xi)) \subseteq \mathsf{fn}((\mathcal{L}(\gamma, B)\gamma, E, \xi))$, let us conclude with the desired result $\rho, \kappa, \tau \vdash_{\mathsf{S}} \lfloor (\mathcal{L}(\gamma, B), E, \xi) \rfloor$.

□

Finally we prove the semantic correctness of our analysis, showing that, if ρ, κ, τ is a valid analysis result for S_o, the same holds for its derivatives.

Theorem 6.1.6 *Let S_o be a well-formed system and ρ, κ, τ an analysis result. If $S \in ds(S_o)$ and $\rho, \kappa, \tau \vdash_{\mathsf{S}} S_o$ then $\rho, \kappa, \tau \vdash_{\mathsf{S}} S$.*

Proof. By induction on the number of transitions which are necessary in order to reach S starting from S_o.

base case: we have that $S = S_o$ and we can immediately conclude observing that, by definition of canonical names, this implies that $\lfloor S \rfloor = \lfloor S_o \rfloor$.

step case: we have that $S_0 \rightarrow_{r_1} \ldots \rightarrow_{r_n} S_n \rightarrow_r S$. By inductive hypothesis $\rho, \kappa, \tau \vdash_{\mathsf{S}} S_n$. We conclude observing that that if $S_n \rightarrow_r S$ then $S_n \xrightarrow{\tau}_r S$ and thus, by Lemma 6.1.5, $\rho, \kappa, \tau \vdash_{\mathsf{S}} S$.

□

6.1.1 Moore Family Result

Here we show that the set of solutions which satisfies a system S is a Moore family. This implies that there always exists a least solution for S. As first step we prove a result about conditions and the intersection of couples of abstract environments and abstract interface environments.

Lemma 6.1.7 *If $\rho = \rho_1 \sqcap \rho_2$ and $\tau = \tau_1 \sqcap \tau_2$ then $\mathsf{check}_{\rho,\tau}(C) \subseteq \mathsf{check}_{\rho_1,\tau_1}(C) \cap \mathsf{check}_{\rho_2,\tau_2}(C)$.*

Proof. We proceed by induction on the structure of C. We only consider the case $C = (x,\, T)^r$. The proof goes through all the possible values $\mathsf{check}_{\rho_1,\tau_1}(C)$ and $\mathsf{check}_{\rho_2,\tau_2}(C)$ can evaluate to.

case $\mathsf{check}_{\rho_1,\tau_1}(C) = \emptyset$**:** we have that $\tau_1^*(\mathsf{Sort}) \cap \rho_1(x) = \emptyset$ which implies $\tau^*(\mathsf{Sort}) \cap \rho(x) = \emptyset$. This is enough for concluding with the desired result given that $\mathsf{check}_{\rho,\tau}(C) = \emptyset$.

case $\mathsf{check}_{\rho_i,\tau_i}(C) = \{\mathsf{true}\}$ **for** $i \in \{1,2\}$**:** this implies that $\tau_i(T) \cap \rho_i(x) \neq \emptyset$ and $\tau_i^*(\mathsf{Sort}) \cap \rho_i(x) = \emptyset$ with $i \in \{1,2\}$. From this we get that $\tau^*(\mathsf{Sort}) \cap \rho(x) = \emptyset$ which is enough for concluding that $\mathsf{check}_{\rho,\tau}(C)$ evaluates to $\{\mathsf{true}\}$ or $\emptyset$, which are subsets of $\{\mathsf{true}\}$.

case $\mathsf{check}_{\rho_i,\tau_i}(C) = \{\mathsf{false}\}$ **for** $i \in \{1,2\}$**:** this case is similar to the previous because $\tau(T) \cap \rho(x) = \emptyset$. This implies that $\mathsf{check}_{\rho,\tau}(C)$ can evaluate to $\emptyset$ or $\{\mathsf{false}\}$ and thus it is a subset of $\{\mathsf{false}\}$.

case $\mathsf{check}_{\rho_1,\tau_1}(C) = \{\mathsf{true}\}$ **and** $\mathsf{check}_{\rho_2,\tau_2}(C) = \{\mathsf{false}\}$**:** we have to verify that $\mathsf{check}_{\rho,\tau}(C) = \emptyset$. We observe that $\tau_1(T) \cap \rho_1(x) \neq \emptyset$ and $\tau_2(T) \cap \rho_2(x) = \emptyset$ which implies $\tau(T) \cap \rho(x) = \emptyset$. Moreover, we have that $\tau_1^*(\mathsf{Sort} \setminus \{T\}) \cap \rho_1(x) = \emptyset$ and $\tau_2^*(\mathsf{Sort} \setminus \{T\}) \cap \rho_2(x) \neq \emptyset$ that implies $\tau^*(\mathsf{Sort} \setminus \{T\}) \cap \rho(x) = \emptyset$. Now, $\tau(T) \cap \rho(x) = \emptyset$ and $\tau^*(\mathsf{Sort} \setminus \{T\}) \cap \rho(x) = \emptyset$ imply $\tau^*(\mathsf{Sort}) \cap \rho(x) = \emptyset$ and thus we can conclude with the desired result $\mathsf{check}_{\rho,\tau}(C) = \emptyset$.

case $\mathsf{check}_{\rho_1,\tau_1}(C) = \{\mathsf{true}, \mathsf{false}\}$**:** in this case $\tau^*(\mathsf{Sort} \setminus \{T\}) \cap \rho(x) = \emptyset$ given that $\tau_2^*(\mathsf{Sort} \setminus \{T\}) \cap \rho_2(x) = \emptyset$, instead $\tau(T) \cap \rho(x)$ can be either empty or not. Anyway we are safe in both cases given that $\mathsf{check}_{\rho,\tau}(C)$ evaluates to $\emptyset$ and $\{\mathsf{true}\}$ respectively.

case $\mathsf{check}_{\rho_1,\tau_1}(C) = \{\mathsf{true}, \mathsf{false}\}$ **and** $\mathsf{check}_{\rho_2,\tau_2}(C) = \{\mathsf{true}\}$**:** In this case $\tau^*(\mathsf{Sort} \setminus \{T\}) \cap \rho(x) = \emptyset$ given that $\tau_2^*(\mathsf{Sort} \setminus \{T\}) \cap \rho_2(x) = \emptyset$, instead $\tau(T) \cap \rho(x)$ can be either empty or not. Anyway we are safe in both cases given that $\mathsf{check}_{\rho,\tau}(C)$ evaluates to $\emptyset$ and $\{\mathsf{true}\}$, respectively.

case $\mathsf{check}_{\rho_1,\tau_1}(C) = \{\mathsf{true}, \mathsf{false}\}$ **and** $\mathsf{check}_{\rho_2,\tau_2}(C) = \{\mathsf{false}\}$**:** here, with respect to the previous case, we obtain the specular situation where $\tau(T) \cap \rho(x)$ is empty and we cannot say anything about $\tau^*(\mathsf{Sort} \setminus \{T\}) \cap \rho(x)$. However, also in this case, we obtain the desired result given that the result of $\mathsf{check}_{\rho,\tau}(C)$ is either $\emptyset$ or $\{\mathsf{false}\}$.

case $\mathsf{check}_{\rho_i,\tau_i}(C) = \{\mathsf{true}, \mathsf{false}\}$ **for** $i \in \{1,2\}$**:** we can immediately conclude because we have $\mathsf{check}_{\rho_1,\tau_1}(C) \cap \mathsf{check}_{\rho_2,\tau_2}(C) = \{\mathsf{true}, \mathsf{false}\}$.

□

Here we extend the result of the previous lemma to abstract environments and abstract interface environments which result from the intersection of a generic number of elements.

Corollary 6.1.8 *Let C be a condition, ρ_i an abstract environment and τ_i an abstract interface environment for $i \in [1,n]$. If $\rho = \bigsqcap_{i\in[1,n]} \rho_i$ and $\tau = \bigsqcap_{i\in[1,n]} \tau_i$ then:*

$$\mathsf{check}_{\rho,\tau}(C) \subseteq \bigcap_{i\in[1,n]} \mathsf{check}_{\rho_i,\tau_i}(C)$$

Sketch. It follows by Lemma 6.1.7.

□

Finally, we prove that the set of valid analysis results of the system S is a Moore Family.

Theorem 6.1.9 *Given a system S the set $\{(\rho,\kappa,\tau) : \rho,\kappa,\tau \vdash_{\mathsf{S}} S\}$ is a Moore Family.*

Proof. Let us consider an index set I and assume that $\forall i \in I : \rho_i,\kappa_i,\tau_i \vdash_{\mathsf{S}} S$ so that, given $\rho = \bigcap_{i\in I}\rho_i$, $\kappa = \bigcap_{i\in I}\kappa_i$ and $\tau = \bigcap_{i\in I}\tau_i$ we shall prove that $\rho,\kappa,\tau \vdash_{\mathsf{S}} S$. In order to get the result we should prove this result for all the judgments. However we observe that for $\vdash_{\mathsf{S}}$ and $\vdash_{\mathsf{B}}$ the result is immediate and thus we focus our attention on $\vdash_{\mathsf{P}}$ and $\vdash_{\mathsf{A}}$.

In order to get the result for $\vdash_{\mathsf{A}}$ we should handle all the possible values which π can assume. However we observe that the result follows from these observations:

1. $\forall i \in I : \rho_i(v) \subseteq \tau_i(T) \Rightarrow \rho(v) \subseteq \tau(T)$
2. $\forall i \in I : \rho_i(v) \subseteq \kappa_i(A,n) \Rightarrow \rho(u) \subseteq \kappa(A,n)$
3. $\forall i \in I : \kappa_i(A,n) \subseteq \rho_i(y) \Rightarrow \kappa(A,n) \subseteq \rho(y)$
4. $\forall i \in I : \rho_i(v) \cap \tau_i(T) \supseteq V \Rightarrow \rho(v) \cap \tau(T) \subseteq V$

To get the result for $\vdash_{\mathsf{P}}$ we proceed by induction on the structure of the processes. Most of the cases can be trivially proved by mere application of the induction hypothesis therefore we consider only the most interesting case $\langle C\rangle M$. Here we have two sub-cases.

sub-case $\{\mathsf{true}\} \subseteq \mathsf{check}_{\rho,\tau}(C)$**:** in this case, by Corollary 6.1.8, we know that $\forall i \in I, \{\mathsf{true}\} \subseteq \mathsf{check}_{\rho_i,\tau_i}(C)$, which implies $\rho_i,\kappa_i,\tau_i \vdash^N_{\mathsf{A}} \pi$ and $\rho_i,\kappa_i,\tau_i \vdash^N_{\mathsf{P}} P$. This, exploiting the result for the judgment $\vdash_{\mathsf{A}}$ and applying the inductive hypothesis on $\rho_i,\kappa_i,\tau_i \vdash^N_{\mathsf{P}} P$, let us conclude with the desired result.

sub-case $\mathsf{check}_{\rho,\tau}(C) \subseteq \{\mathsf{false}\}$**:** here we can conclude considering that $\mathsf{check}_{\rho,\tau}(C) \subseteq \{\mathsf{false}\}$ is enough in order to conclude $\rho,\kappa,\tau \vdash^N_{\mathsf{P}} \langle C\rangle M$.

□

6.2 Exploiting the Analysis

In order to exploit the result of the analysis we redefine the proper derivatives (Definition 5.2.2) of a box $I[\,P\,]$ with respect to the system S. The new definition replaces the implication $a = T?m \Rightarrow m \in \mathsf{sn}(S)$ with $a = T?m \Rightarrow \lfloor m \rfloor \in \bigcup_{m'\in\tau^*(\mu(S))}\kappa(m'))$. In doing so we take into account that, in order to receive a name m over an interface which is associated with the sort T, such a name must be sent by an interface which is possibly associated with a sort having non-zero binding capability with T.

Definition 6.2.1 *$I_2[\,P_2\,]$ is a (one-step) proper derivative of $I_1[\,P_1\,]$ with respect to the system S if it holds that $I_1[\,P_1\,] \xrightarrow[\mathtt{Box}]{C,a}_r I_2[\,P_2\,]$ and $a = T?m \Rightarrow \lfloor m \rfloor \in \bigcup_{m'\in\tau^*(\mu(S))}\kappa(m'))$ where ρ,κ,τ is the smallest analysis result such that $\rho,\kappa,\tau \vdash_{\mathsf{S}} S$. More generally if*

$$I_1[\,P_1\,] \xrightarrow[\mathtt{Box}]{C_1,a_1}_{r_1} \cdots \xrightarrow[\mathtt{Box}]{C_{n-1},a_{n-1}}_{r_{n-1}} I_n[\,P_n\,]$$

and $a_i = T?m \Rightarrow \lfloor m \rfloor \in \bigcup_{m'\in\tau^*(\mu(S))}\rho(m')$ for $i \in [1,n)$ then $I_n[\,P_n\,]$ is a proper derivative of $I_1[\,P_1\,]$ respect to the system S. Finally, the set $pds(I[\,P\,],S)$ of the proper derivatives of $I[\,P\,]$

with respect to the system S, is defined as the set containing $I[\,P\,]$ and all the boxes $I'[\,P'\,]$ such that $I'[\,P'\,]$ is a proper derivative of $I[\,P\,]$ with respect to the system S.

In order to verify that Theorem 5.2.14 is still valid we show a modified version of the Lemma 5.2.10 in Lemma 6.2.3. This result, through Lemmas 5.2.11, 5.2.12, and 5.2.13, propagates to Theorem 5.2.14. In this way we guarantee that the relation $\Phi^{S_o}_{\mathtt{Box}} \subseteq \tilde{\Phi}^{S_o}_{\mathtt{Box}}$ is still valid, even with the modified definition of proper derivatives. After showing an intermediate result we prove Lemma 6.2.3.

Lemma 6.2.2 *Let $I[\,P\,]_\gamma$ be a well-formed box and ρ, τ, κ an analysis result. If exists the box transition $I[\,P\,]_\gamma \xrightarrow{C,T!n}_r I[\,P\,]_\gamma$ and $\rho, \tau, \kappa \vdash_{\mathsf{B}} I[\,P\,]_\gamma$ then $(b, T)^{r_1} \in I$ and $\lfloor b \rfloor \in \tau(T)$ and $\lfloor n \rfloor \in \kappa(\lfloor b \rfloor)$.*

Proof. By induction on the length of the derivation of $I[\,P\,]_\gamma \xrightarrow{C,T!n}_r I[\,P\,]_\gamma$. We only consider the interesting case of rule r1. We have that $\rho, \tau, \kappa \vdash_{\mathsf{B}} I[\,n!m.\,P\,]_\gamma$ and that $(n, T)^{r_1} \in I$. This immediately implies that $\lfloor n \rfloor \in \tau(T)$ and that $\lfloor m \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor b \rfloor)$ and thus, observing that $\lfloor m \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor b \rfloor)$ implies $\lfloor m \rfloor \in \kappa(\lfloor b \rfloor)$, we concluded with the desired result.

□

Lemma 6.2.3 *Let S be a well-formed system and ρ, τ, κ an analysis result. If $S \xrightarrow{l}_r S'$ and $get(\gamma, S') = I'[\,P'\,]_\gamma$ and $\rho, \kappa, \tau \vdash_{\mathsf{S}} S$ then one of the following holds:*

1. *if $l = (\gamma', T!n)$ then*
   (a) *if $\gamma \neq \gamma'$ then $I'[\,P'\,] \in getBox(S)$,*
   (b) *otherwise $get(\gamma, S) = I[\,P\,]_\gamma$ and $\rho, \tau, \kappa \vdash_{\mathsf{B}} I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow{C,T!n}_0 I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$ and $(b, T)^{r_1} \in I$ and $\lfloor n \rfloor \in \kappa(\lfloor b \rfloor)$ and $\lfloor b \rfloor \in \tau(T)$.*
2. *if $l = (\gamma', T?n)$ then*
   (a) *if $\gamma \neq \gamma'$ then $I'[\,P'\,] \in getBox(S)$,*
   (b) *otherwise $get(\gamma, S) = I[\,P\,]_\gamma$ and $\rho, \tau, \kappa \vdash_{\mathsf{B}} I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow{C,T?n}_0 I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$.*
3. *if $l = \tau$ then*
   (a) *$I'[\,P'\,] \in getBox(S)$ or*
   (b) *$get(\gamma, S) = I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow{C,a}_{r'} I'[\,P'\,]_\gamma$ with $C \neq \mathsf{false}$,*
   $a \notin \{n?t, n!t\}$,
   $(a = T!n \Rightarrow (b, T)^{r_1} \in I \wedge \lfloor n \rfloor \in \kappa(\mathsf{sub}(\lfloor I \rfloor), \lfloor b \rfloor) \wedge \lfloor b \rfloor \in \tau(T) \wedge r' = 0)$,
   $(a = T?n \Rightarrow \lfloor n \rfloor \in \bigcup_{m \in \tau^*(\mu(T))} \kappa(m) \wedge r' = 0)$ and
   $(a \notin \{T!n, T?n\} \Rightarrow r' = r)$
4. *if $l = (\gamma', T)$ then $I'[\,P'\,] \in getBox(S)$.*

Proof. By induction on the length of the derivation of $S \xrightarrow{l}_r S'$. We consider only the interesting cases.

base case r9: if $\rho, \tau, \kappa \vdash_{\mathsf{S}} (I[\, n!m.\, P\,]_\gamma, E, \xi)$ then $\rho, \tau, \kappa \vdash_{\mathsf{B}} I[\, n!m.\, P\,]_\gamma$. We immediately conclude by Lemma 6.2.2.

base case r16-l: from $\rho, \tau, \kappa \vdash_{\mathsf{S}} (B_1 \parallel B_2, E, \xi)$ we obtain that $\rho, \tau, \kappa \vdash_{\mathsf{S}} (B_i, E, \xi)$ for $i \in [1, 2]$. We now observe that if $get(\gamma, (B_1' \parallel B_2', E, \xi)) = I'[\, P'\,]_\gamma$ then it holds that either $(get(\gamma, (B_1', E, \xi)) = I'[\, P'\,]_\gamma$ and $get(\gamma, (B_2', E, \xi)) = \bot)$ or $(get(\gamma, (B_1', E, \xi)) = \bot$ and $get(\gamma, (B_2', E, \xi)) = I'[\, P'\,]_\gamma)$.

In the case $get(\gamma, (B_1', E, \xi)) = I'[\, P'\,]_\gamma$, given that $\rho, \tau, \kappa \vdash_{\mathsf{S}} (B_1, E, \xi)$, we apply the inductive hypothesis to $\rho, \tau, \kappa \vdash_{\mathsf{S}} (B_1, E, \xi) \xrightarrow{(\gamma_1, T_1!n)}_r (B_1', E, \xi)$ and we obtain that one of the following holds:

1. if $\gamma \neq \gamma_1'$ then $I'[\, P'\,] \in getBox((B_1, E, \xi))$ which implies $I'[\, P'\,] \in getBox((B_1 \parallel B_2, E, \xi))$ and we are done.
2. otherwise we have that $get(\gamma, (B_1, E, \xi)) = I[\, P\,]_\gamma$ and $I[\, P\,]_\gamma \xrightarrow{C, T_1!n}_r I'[\, P'\,]_\gamma$ with $\rho, \tau, \kappa \vdash_{\mathsf{B}} I[\, P\,]_\gamma$ and thus we can apply the result of Lemma 6.2.2 and obtain that $(b, T_1)^{r_1} \in I$ and $\lfloor b \rfloor \in \tau(T)$ and $\lfloor n \rfloor \in \kappa(\lfloor b \rfloor)$. If $get(\gamma, (B_1, E, \xi)) = I[\, P\,]_\gamma$ then $get(\gamma, (B_1 \parallel B_2, E, \xi)) = I[\, P\,]_\gamma$ and we obtained the desired result.

In the case $get(\gamma, (B_2', E, \xi)) = I'[\, P'\,]_\gamma$, given that $\rho, \tau, \kappa \vdash_{\mathsf{S}} (B_1, E, \xi)$, we apply the inductive hypothesis to $\rho, \tau, \kappa \vdash_{\mathsf{S}} (B_2, E, \xi) \xrightarrow{(\gamma_2, T_2?n)}_r (B_2', E, \xi)$ and similarly to the previous case we obtain that:

1. if $\gamma \neq \gamma_2$ then $I'[\, P'\,] \in getBox((B_1 \parallel B_2, E, \xi))$,
2. otherwise $get(\gamma, (B_1, E, \xi)) = I[\, P\,]_\gamma$ and $I[\, P\,]_\gamma \xrightarrow{C, T_2?n}_0 I'[\, P'\,]_\gamma$.

In this case if $\gamma = \gamma_2$, in oder to conclude, we have to verify that $\lfloor n \rfloor \in \bigcup_{m \in \tau^*(\mu(T_2))} \kappa(m)$. This follows by observing that we know (as we have seen in the previous case if $\gamma = \gamma_1$), that $\exists b : \lfloor n \rfloor \in \kappa(b)$ with $b \in \tau(T_1)$ and $T_1 \in \mu(T_2)$ (given that $\alpha_c(T_1, T_2) \neq 0$). This immediately implies that $\lfloor n \rfloor \in \bigcup_{m \in \tau^*(\mu(T_2))} \kappa(m)$ therefore we can conclude with the desired result.

□

Summing up, in this chapter, we refined the over-approximation of $\Phi^{S_o}_{\mathtt{Box}}$ proposed in Chapter 5. We obtained the result by modifying the notion of proper derivatives of a box. Proper derivatives introduce approximation because their generation requires to guess the channel names which boxes can receive. In Chapter 5 this set of channels is over-approximated to the sent names of S_o; this approximation is safe but, in most of the cases, extremely imprecise. Here we defined a control flow analysis which, for each interface, compute an over-approximation of the sorts it can be associated with and of the channels it can send. This information allows us to improve the precision in guessing the channel names receivable by interfaces of a specific box.

Here we do not apply this analysis to the actin case study because, in the actin model, boxes do not send names over interfaces, therefore, what introduced in this chapter cannot improve the results obtained in Section 5.4.

Chapter 7

Encoding to κ-calculus

In this chapter we define an encoding of BlenX into κ-calculus. This allows us to exploit the results obtained in [32] (see Section 2.3.4) in the context of BlenX.

The representation of a BlenX system in κ-calculus encodes each box with an agent. Interfaces of a box have a one to one mapping with the interfaces of the dual agent, and bindings among agents occur as specified by the environment of the BlenX system. The information to generate the rules driving the evolution of the κ-calculus expression encoding the system S, is gathered from the transition system $\tilde{\Phi}^S_{\mathtt{Box},\equiv}$. Here, the finiteness of $\tilde{\Phi}^S_{\mathtt{Box},\equiv}$, which we proved in Chapter 5.3.38 for system whose boxes are in $\mathcal{FF}$, becomes crucial in order to ensure the generation of a finite number of κ-calculus rules.

In this chapter we enrich the κ-calculus language by associating priorities with rules:

$$E_l \rightarrow_p E_r$$

where p can be either ∞ (high priority) or $\not\infty$ (low priority). We only use priorities to show the properties of our encoding; they do not play any role in the semantics of the language. For the sake of readability we modify the syntax of κ-calculus transitions. Originally it was

$$E \xrightarrow[\kappa]{r} E'$$

where r is the rule applied in order to obtain the transition. Now it becomes

$$E \xrightarrow[\kappa]{r}_p E'$$

where p is the priority associated with r.

7.1 Encoding BlenX to κ-calculus

The encoding makes use of canonical names introduced in Chapter 6. We perform the encoding using two functions. The first one is *S2*κ. Given a BlenX system it returns a triple made of a set of agent names, a function defining the interfaces associated with each agent name, and a set of κ-calculus rules. The second function is *B2A* which, given a bio-process and an environment, generates a κ-calculus mixture.

Suppose S to be a BlenX system and $S2\kappa(S) = (\mathtt{Agent}, \Sigma, R)$. The names of `Agent` are sets of canonical names. If a set of names is in `Agent` then it is the subject set of a box of S. Formally:

$$
\begin{array}{rcl}
B2A(B_1 \parallel B_2, \xi) & = & B2A(B_1, \xi), B2A(B_2, \xi) \\
B2A(I[\,P\,]_\gamma, \xi) & = & \mathsf{sub}(\lfloor I \rfloor)\left(z_{[\![I[\,P\,]]\!]}, \lfloor b_1 \rfloor^{j_1}, \ldots, \lfloor b_n \rfloor^{j_n}\right) \text{ where} \\
 & & \lfloor b_i \rfloor < \lfloor b_{i+1} \rfloor \text{ for } i \in [1, n), \\
 & & \{(\lfloor b_1 \rfloor, T_1), \ldots, (\lfloor b_n \rfloor, T_n)\} = \lfloor I \rfloor, \\
 & & \text{for } i \in [1, n]\ j_i = \begin{cases} \ln(L) & \text{if } \exists T', \gamma' \text{ s.t } L = \{(T_i, \gamma), (T', \gamma')\} \in \xi \\ \epsilon_\lambda & \text{otherwise} \end{cases}
\end{array}
$$

Table 7.1 – Function for translating a bio-process into a list of agents.

$$\mathtt{Agent} = \{\mathsf{sub}(\lfloor I \rfloor) : I[\,P\,] \in getBox(S)\}$$

For each agent name the function Σ is defined as follows:

$$
\begin{array}{c}
\Sigma(\mathsf{sub}(\lfloor I \rfloor)) = \{z, \lfloor b_1 \rfloor, \ldots, \lfloor b_n \rfloor\} \text{ where } \lfloor b_1 \rfloor, \ldots, \lfloor b_n \rfloor \text{ are distinct names,} \\
\{\lfloor b_1 \rfloor, \ldots, \lfloor b_n \rfloor\} = \mathsf{sub}(\lfloor I \rfloor) \text{ and } \forall n \in \mathsf{Name}, \lfloor n \rfloor \neq z
\end{array}
$$

Therefore, each agent has a set of interfaces mapping to those of the box it corresponds to, plus a special interface named z. The special interface z stores in its internal state the equivalence class of the box represented by its agent. The other interfaces have the same binding state of the BlenX interfaces they map to. Note that the sort associated with the BlenX interfaces is stored in the internal state of z. What we intuitively explained can be seen in practice in the definition of *B2A* (see Table 7.1). This function, taking as argument a bio-process B and an environment ξ, returns a list whose agents map the boxes of B. It also links the agents in the same way of the boxes they encode (i.e. as specified by ξ). The binding states of the agent interfaces are generated through ln, a bijective function which, given a link belonging to ξ, returns a natural number. Exploiting this function we associate bound interfaces with the same binding state and use a reserved integer number of each link. We introduce an ordering between canonical names. In this way, given a set of interfaces, we generate a unique interface sequence.

The following observation formalizes an interesting property of *B2A* which directly follows by its definition: we can deduce the links of (B, E, ξ) by looking at the agents of $[\![B2A(B, \xi)]\!]$, and vice versa.

Observation 7.1.1 *Given the system (B, E, ξ), it holds that $get(\gamma_i, B) = I_i[\,P_i\,]_{\gamma_i}$, $(b_i, T_i)^{r_i} \in I_i$ for $i \in \{1, 2\}$ and $\{(\gamma_1, T_1), (\gamma_2, T_2)\} \in \xi$ iff in $B2A(B, \xi)$ the agents $\mathsf{sub}(\lfloor I_1 \rfloor)(\sigma_1)$ and $\mathsf{sub}(\lfloor I_2 \rfloor)(\sigma_2)$ occur with the following characteristics: an interface named $\lfloor b_i \rfloor$ with binding state $l \in \mathbb{N}$ and the interface $z_{[\![I_i[\,P_i\,]]\!]}$ occur in σ_i for $i \in \{1, 2\}$. By definition of $\equiv_\kappa$ we can lift this property to $[\![B2A(B, \xi)]\!]$.*

The last element of the triple generated by $S2\kappa$ is the set of rules R. We denote the set R resulting by the application of the function $S2\kappa$ to S as R^S. The definition of R^S is based on $\tilde{\Phi}^S_{\mathtt{Box},\equiv} = (A, U)$. Rules are extracted from the relations whose union generates U (see Definition 5.3.2). These relations are those we refer to as $\xrightarrow[\mathtt{Box},\equiv]{}^{I[\,P\,],S}$ with $I[\,P\,] \in getBox(S)$. Before formalizing the definition of R^S we consider an example in order to give the intuition

which lies behind its generation. Suppose that we want to express

$$[\![I_1[\, P_1\,]]\!] \xrightarrow[\mathtt{Box},\equiv]{C,\tau} {}^{I[P],S}_{\not\sim} [\![I_2[\, P_2\,]]\!]$$

with a κ-calculus rule. Given that $a = \tau$, we know that the boxes belonging to $[\![I_1[\, P_1\,]]\!]$ can perform an internal communication. Internal communications are independent from interactions with other boxes, therefore we generate a κ-calculus rule involving one agent. The agent name is a set of canonical names and is obtained from the interface set of the boxes involved in the transition: $\mathsf{sub}(\lfloor I_1 \rfloor) = \mathsf{sub}(\lfloor I_2 \rfloor) = \mathsf{sub}(\lfloor I \rfloor)$.

For the sake of simplicity, firstly, we consider $C = \mathsf{true}$. This implies that the rule we are defining will not require any binding constraint. As last step we define the internal state of the interface z, both in the lhs and in the rhs of the rule. If an agent matches this rule we want it to belong to the equivalence class $[\![I_1[\, P_1\,]]\!]$, therefore in the lhs we require that z has internal state $[\![I_1[\, P_1\,]]\!]$. After the application of the rule the agent must represent a box belonging to $[\![I_2[\, P_2\,]]\!]$, hence the internal state of z in the rhs is $[\![I_2[\, P_2\,]]\!]$. The generated rule looks as follows:

$$\mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\, P_1\,]]\!]}\right) \to_{\not\sim} \mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_2[\, P_2\,]]\!]}\right)$$

Now we consider a more complex condition C. Suppose $C = (T, \otimes)$ with $(b, T) \in I_1$. The binding constraint can be expressed as follows:

$$\mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\, P_1\,]]\!]}, \lfloor b \rfloor^-\right) \to_p \mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_2[\, P_2\,]]\!]}, \lfloor b \rfloor^-\right)$$

Not all the conditions can be so easily expressed. This is the case for conditions expressing disjunction: e.g. $\neg((T_1, \otimes) \wedge (T_2, \otimes))$ which is equivalent to $(T_1, \odot) \vee (T_2, \odot)$. A single κ-calculus rule cannot express $(T_1, \odot) \vee (T_2, \odot)$, in fact a list of constraints expressed in terms of binding states can express conjunction but not disjunction. We need to introduce more rules which consider all the cases in which the condition is satisfied. In our example, assuming $(b_1, T_1), (b_2, T_2) \in I_1$, we need three rules which handle separately the cases $(T_1, \odot) \wedge (T_2, \odot)$, $(T_1, \odot) \wedge (T_2, \otimes)$ and $(T_1, \otimes) \wedge (T_2, \odot)$:

$$\begin{array}{rcl}
\mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\, P_1\,]]\!]}, \lfloor b_1 \rfloor, \lfloor b_2 \rfloor\right) & \to_p & \mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_2[\, P_2\,]]\!]}, \lfloor b_1 \rfloor, \lfloor b_2 \rfloor\right) \\
\mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\, P_1\,]]\!]}, \lfloor b_1 \rfloor, \lfloor b_2 \rfloor^-\right) & \to_p & \mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_2[\, P_2\,]]\!]}, \lfloor b_1 \rfloor, \lfloor b_2 \rfloor^-\right) \\
\mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\, P_1\,]]\!]}, \lfloor b_1 \rfloor^-, \lfloor b_2 \rfloor\right) & \to_p & \mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_2[\, P_2\,]]\!]}, \lfloor b_1 \rfloor^-, \lfloor b_2 \rfloor\right)
\end{array}$$

Thus, when $(T_1, \odot) \vee (T_2, \odot)$ is true, we know that one of these rules is active, in fact it holds that:

$$\begin{array}{c}
\wr \neg((T_1, \otimes) \wedge (T_2, \otimes)) \int_{\gamma,\xi} \\
\Leftrightarrow \\
\wr (T_1, \odot) \wedge (T_2, \odot) \int_{\gamma,\xi} \vee \wr\, (T_1, \odot) \wedge (T_2, \otimes) \int_{\gamma,\xi} \vee \wr\, \neg((T_1, \otimes) \wedge (T_2, \odot)) \int_{\gamma,\xi}
\end{array}$$

Thus we need to define a function which, given a partially evaluated condition C, generates a set of sets of elementary conditions:

$$\{\{(T_1, k_{1,1}), \ldots, (T_n, k_{1,n})\}, \ldots, \{(T_1, k_{m,1}), \ldots, (T_n, k_{m,n})\}\}$$

$$
\begin{array}{rcl}
icg(\mathsf{true}) & = & \{\emptyset\} \\
icg((T,k)) & = & \{(T,k)\} \text{ where } (T,\,k)^r \in I \\
icg(\neg C) & = & \{A \setminus icg(C)\} \\
 & & \text{where } A = \{\{(b_1,k_1),\ldots,(b_n,k_n)\} \mid \\
 & & ((b_1,k_1),\ldots,(b_n,k_n)) \in \{(b_1,\odot),(b_1,\otimes)\} \times \cdots \times \{(b_n,\odot),(b_n,\otimes)\} \\
 & & \text{and } \{(b_1,k_1),\ldots,(b_n,k_n))\} \in icg(C) \\
\{icg(C_1 \wedge C_2) & = & \{A_1 \cup A_2 \mid \quad A_1 \in icg(C_1) \wedge A_2 \in icg(C_2) \wedge \\
 & & \qquad\qquad \forall (T,k) \in A_1 \cup A_2, \neg(T,k) \notin A_1 \cup A_2\}
\end{array}
$$

Table 7.2 – Definition of the *icg* function. In the case $icg(\neg C)$, if $icg(C)$ is either $\emptyset$ or $\{\emptyset\}$, we consider that $A = \{\emptyset\}$.

such that

$$\wr C \int_{\gamma,\xi} \Leftrightarrow (\wr(T_1,k_{1,1}) \wedge \cdots \wedge (T_n,k_{1,n}) \int_{\gamma,\xi} \vee \cdots \vee (\wr(T_1,k_{m,1}) \wedge \cdots \wedge (T_n,k_{m,n}) \int_{\gamma,\xi})$$

Formally, we give a method to generate a disjunctive normal form formula which is equivalent to the input condition. This task is performed by the function *interface configuration generator* (*icg*) whose definition is in Table 7.2. Its property is formalized by Lemma 7.1.2.

Lemma 7.1.2 *If* $icg(C) = \{\{(T_1,k_{1,1}),\ldots,(T_n,k_{1,n})\},\ldots,\{(T_1,k_{m,1}),\ldots,(T_n,k_{m,n})\}\}$ *then*

$$\wr C \int_{\gamma,\xi} \Leftrightarrow (\wr(T_1,k_{1,1}) \wedge \cdots \wedge (T_n,k_{1,n}) \int_{\gamma,\xi} \vee \cdots \vee (\wr(T_1,k_{m,1}) \wedge \cdots \wedge (T_n,k_{m,n}) \int_{\gamma,\xi})$$

Sketch. By induction on the structure of the partially evaluated condition C.

□

In our example the result of *icg* to $\neg((T_1,\otimes) \wedge (T_2,\otimes))$ is:

$$\{\{(T_1,\odot),(T_2,\odot)\},\{(T_1,\odot),(T_2,\otimes)\},\{(T_1,\otimes),(T_2,\odot)\}\}$$

Note that, in general, the formulas generated by the *icg* function have the following properties: the literals of each clause involve the same interfaces, and these interfaces appear exactly once (in this way we avoid having clauses in which both (T,k) and $\neg(T,k)$) appear.

In Table 7.3, we give the clauses which formally define the high priority rule set R^S_∞. [INT] generates rules from transitions which involve a single box performing an internal action (a change action or an internal communication). We make use of the function *it* (see Table 7.4). Given a sequence of κ-calculus interfaces σ and a set of BlenX interfaces I, it generates a set of BlenX literals that, joined by means of the conjunction operator, express the same binding constraints of σ. Note that the function is defined only if σ is an ordered sequence. In doing so we make *it* a bijective function and we avoid having several rules which express the same transition with interface sequences which only differ in the interface ordering.

The clause [BIND] generates rules which involve two agents with a free interface each. As a result of the reaction the free interfaces bind together. Note that they map BlenX interfaces with sorts having ∞ as binding capability. The clause [UNBIND] is very similar but it handles rules which break a link between two agents.

[INT]
$$\mathsf{sub}(\lfloor I' \rfloor)\left(z_{[\![I'[\,P'\,]]\!]}, \sigma\right) \to_\infty \mathsf{sub}(\lfloor I' \rfloor)\left(z_{[\![I''[\,P''\,]]\!]}, \sigma\right) \in R_\infty$$
$$\text{iff}$$
$$\exists I[\,P\,] \in getBox(S) \text{ s.t. } [\![I'[\,P'\,]]\!] \xrightarrow[\text{Box},\equiv]{C,a}_\infty I[\,P\,], S\ [\![I''[\,P''\,]]\!] \text{ with } a \in \{\tau, (T,U)\} \wedge it(\sigma, \lfloor I' \rfloor) \in icg(C)$$

[BIND]
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \lfloor b_1 \rfloor\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \lfloor b_2 \rfloor\right) \to_\infty$$
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \lfloor b_1 \rfloor^1\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \lfloor b_2 \rfloor^1\right) \in R_\infty$$
$$\text{iff}$$
$$\exists I_i[\,P_i\,] \in getBox(S) \text{ s.t. } [\![I'_i[\,P'_i\,]]\!] \in ds(I_i[\,P_i\,], S)/{\equiv_b} \wedge (\lfloor b_i \rfloor, T_i)^{r_i} \in \lfloor I_i \rfloor \text{ with } i \in [1,2] \wedge \alpha_b(T_1, T_2) = \infty$$

[UNB]
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \lfloor b_1 \rfloor^1\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \lfloor b_2 \rfloor^1\right) \to_\infty$$
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \lfloor b_1 \rfloor\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \lfloor b_2 \rfloor\right) \in R_\infty$$
$$\text{iff}$$
$$\exists I_i[\,P_i\,] \in getBox(S) \text{ s.t. } [\![I'_i[\,P'_i\,]]\!] \in ds(I_i[\,P_i\,], S)/{\equiv_b} \wedge (\lfloor b_i \rfloor, T_i)^{r_i} \in \lfloor I_i \rfloor \text{ with } i \in [1,2] \wedge \alpha_u(T_1, T_2) = \infty$$

[UCOM]
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \sigma_1\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \sigma_2\right) \to_\infty$$
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I''_1[\,P''_1\,]]\!]}, \sigma_1\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I''_2[\,P''_2\,]]\!]}, \sigma_2\right) \in R_\infty$$
$$\text{iff}$$
$$\exists I_i[\,P_i\,] \in getBox(S) \text{ s.t. } [\![I'_i[\,P'_i\,]]\!] \xrightarrow[\text{Box},\equiv]{C_i,a_i} ?\ I_i[\,P_i\,], S\ [\![I''_i[\,P''_i\,]]\!]$$
$$(\lfloor m_i \rfloor, T_i)^{r_i} \in \lfloor I_i \rfloor \wedge it(\sigma_i, \lfloor I'_i \rfloor) \in icg(C_i \wedge (T_i, \odot)) \text{ with } i \in [1,2] \wedge$$
$$a_1 = T_1!n \wedge a_2 = T_2?n \wedge \alpha_c(T_1, T_2) = \infty \wedge \alpha_b(T_1, T_2) = \alpha_u(T_1, T_2) = 0$$

[BCOM]
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \lfloor m_1 \rfloor^1, \sigma_1\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \lfloor m_2 \rfloor^1, \sigma_2\right) \to_\infty$$
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I''_1[\,P''_1\,]]\!]}, \lfloor m_1 \rfloor^1, \sigma_1\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I''_2[\,P''_2\,]]\!]}, \lfloor m_2 \rfloor^1, \sigma_2\right) \in R_\infty$$
$$\text{iff}$$
$$\exists I_i[\,P_i\,] \in getBox(S) \text{ s.t. } [\![I'_i[\,P'_i\,]]\!] \xrightarrow[\text{Box},\equiv]{C_i,a_i} ?\ I_i[\,P_i\,], S\ [\![I''_i[\,P''_i\,]]\!] \wedge$$
$$(\lfloor m_i \rfloor, T_i)^{r_i} \in \lfloor I'_i \rfloor \wedge (it((\sigma_i), \lfloor I'_i \rfloor) \cup \{(T_i, \otimes)\}) \in icg(C_i \wedge (T_i, \otimes)) \wedge \text{ with } i \in [1,2] \wedge$$
$$a_1 = T_1!n \wedge a_2 = T_2?n \wedge \alpha_c(T_1, T_2) = \infty$$

[EV]
$$\mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\,P_1\,]]\!]}, \sigma_1\right), \emptyset, \ldots, \emptyset \to_\infty$$
$$\mathsf{sub}(\lfloor I_2 \rfloor)\left(z_{[\![I_2[\,P_2\,]]\!]}, \sigma_2\right), \ldots, \mathsf{sub}(\lfloor I_n \rfloor)\left(z_{[\![I_n[\,P_n\,]]\!]}, \sigma_n\right) \in R_\infty$$
$$\text{iff}$$
$$\exists I[\,P\,] \in getBox(S) \text{ s.t. } I'_1[\,P'_1\,] \blacktriangleright_\infty I'_2[\,P'_2\,] \parallel \cdots \parallel I'_n[\,P'_n\,] \in E \wedge \exists [\![I_1[\,P_1\,]]\!] \in ds(I[\,P\,], S)/{\equiv_b} \wedge$$
$$I'_i[\,P'_i\,] \equiv_b I_i[\,P_i\,] \wedge \sigma_i = \lfloor b_{1,i} \rfloor, \ldots, \lfloor b_{n_i,i} \rfloor \wedge \Sigma(\mathsf{sub}(\lfloor I_i \rfloor)) = \{\lfloor b_{1,i} \rfloor, \ldots, \lfloor b_{n_i,i} \rfloor\} \text{ for } i \in [1,n]$$

Table 7.3 – Definition of the set R^S_∞ with respect to the system $S = (B, E, \xi)$.

The clauses [UCOM] and [BCOM] generate rules mapping the capability of the boxes to perform inter-communication. The first handles the case of inter-communications taking place over free interfaces, the second of those taking place through bound interfaces. The agents involved in the rule maps to boxes which can perform symmetric input and output over interfaces with sorts having communication capability equal to ∞. In the case of [UCOM] the communication can happen only if sorts have null binding and unbinding capabilities, as it is the case in BlenX (see Rule 16-l and 16-r in Table 3.8).

The last clause is [EV] which defines rules starting from the event set. In particular an event

$$
\begin{array}{rcl}
it((b_1{}^{\lambda_1}_{\iota_1}, \ldots, b_n{}^{\lambda_n}_{\iota_n}, I) & = & \begin{cases} \{(T_1,k_1),\ldots,(T_n,k_n)\} & \text{with } k_i = \odot \text{ if } \lambda_i = \epsilon_\lambda \wedge \\ & k_i = \otimes \text{ otherwise} \\ & \text{and } (b_i,\, T_i)^r \in I \wedge b_1 < \ldots < b_n \\ \bot & \text{if } \exists i \in [1,n) : b_i \geq b_{i+1} \end{cases} \\
it(\varepsilon, I) & = & \emptyset
\end{array}
$$

Table 7.4 – Definition of the *it* function. Note that in this table the interfaces have two subscripts. The integer number is part of the interface name, the symbol ι_i with $i \in \mathbb{N}$ is instead the internal state of the interface (which in practice is always ϵ_ι).

$I[\,P\,] \blacktriangleright_\infty B$ leads to the generation of a rule if, inside the transition system of the boxes of the system S, a box congruent to $I[\,P\,]$ exists. The rules generated, when applied, remove from the target expression the agent mapping the box congruent to $I[\,P\,]$ and introduce the agents mapping the bio-process B. Note that we use ghost agents in order to obtain the same number of agents in the lhs and in the rhs (as required in order to have a well formed rule).

For our convenience, we partition the set R^S_∞ dividing the rules depending on their generation clause. The partitions are intuitively named $R^{S,\mathrm{INT}}_\infty$, $R^{S,\mathrm{BIND}}_\infty$, $R^{S,\mathrm{UNB}}_\infty$, $R^{S,\mathrm{UCOM}}_\infty$, $R^{S,\mathrm{BCOM}}_\infty$, and $R^{S,\mathrm{EV}}_\infty$. The set $R^S_{\not\infty}$ is defined with the same clauses of Table 7.3 where the occurrences of ∞ are replaced with $\not\infty$. Also in this case we introduce the partition in the same way of R^S_∞. We base the definition of R^S, the set of rules generated by $S2\kappa(S)$, on the sets R^S_∞ and $R^S_{\not\infty}$:

$$R^S = R^S_\infty \cup \{E_1 \to_{\not\infty} E_1' \in R^S_{\not\infty} \mid \nexists E_3 \in [\![E_1]\!], \nexists E_2 \to_\infty E_2' \in R^S_\infty : E_3 \vDash_\kappa E_2\}$$

In words, R^S results by joining R^S_∞ and the set obtained removing from $R^S_{\not\infty}$ the rules we can ensure to be concurrent with a high priority rule. The concurrency between rules is checked exploiting the $\vDash_\kappa$ relation; if the lhs of a low priority rule has a congruent expression which matches the lhs of a high priority rule, then all the expressions matching the lhs of the low priority rule also match the high priority one. This implies that when the first rule can be applied the same holds for the second.

We now introduce some intermediate results necessary to prove a soundness and a completeness result of the encoding. The following lemma states that if the encoding of a system S performs a step, it transforms into an expression which is congruent to the encoding of a one-step derivative of S. The lemma exclusively holds if we consider high priority rules and derivatives obtained through immediate actions.

Lemma 7.1.3 *If* $S = (B, E, \xi)$ *and* $S2\kappa(S) = (\mathtt{Agent}, \Sigma, R^S)$ *and* $B2A(B,\xi) \xrightarrow[\kappa]{r}_\infty E$ *with* $r \in R^S$ *then*

$$\exists (B', E, \xi') : (B, E, \xi) \to_\infty (B', E, \xi') \text{ with } B2A(B', \xi') \equiv_\kappa E$$

Proof. We only consider the cases where $r \in R^{S,\mathrm{INT}}_\infty \cup R^{S,\mathrm{BIND}}_\infty$.

If r belongs to $R^{S,\mathrm{INT}}_\infty$, it has the following form:

$$\mathsf{sub}(\lfloor I' \rfloor)\left(z_{[\![I'[\,P'\,]]\!]}, \sigma\right) \to_\infty \mathsf{sub}(\lfloor I' \rfloor)\left(z_{[\![I''[\,P''\,]]\!]}, \sigma\right)$$

and this implies that:

$$\exists I[\,P\,] \in getBox(S) \text{ s.t. } [\![I'[\,P'\,]]\!] \xrightarrow[\texttt{Box},\equiv]{C,a}{}^{I[\,P\,],\,S}_{\infty} [\![I''[\,P''\,]]\!]$$
$$\text{with } a \in \{\tau, (T,U)\} \wedge it(\sigma, \lfloor I' \rfloor) \in icg(C)$$

Given that the rule has been applied to $B2A(B,\xi)$, this mixture has an agent $A\,(\sigma')$ which satisfies the lhs of r. By definition of $B2A$ this means that in B there exists a box $I_1[\,P_1\,]_\gamma$ such that $I_1[\,P_1\,] \equiv_b I'[\,P'\,]$ which implies $\lfloor I_1 \rfloor = \lfloor I' \rfloor$. Moreover $\sigma = \lfloor b_1 \rfloor^{j_1}, \ldots, \lfloor b_n \rfloor^{j_n}$ from which we deduce $it(\sigma, \lfloor I_1 \rfloor) = \{(T_1,k_1), \ldots, (T_n,k_n)\}$ where, for $i \in [1,n]$, $k_i = \odot$ if $j_i = \epsilon_\lambda$ and, otherwise, $k_i = \otimes$. It also holds that b_i occurs free in σ' if $j_i = \epsilon_\lambda$ and bound otherwise. These observations, by definition of $B2A$, imply that, for $i \in [1,n]$, if $j_i = \epsilon_\lambda$ then $\exists T, \gamma' : \{\{(T_l,\gamma),(T,\gamma')\} \in \xi\}$, if $j_i \neq \epsilon_\lambda$ then $\exists T, \gamma' : \{\{(T_l,\gamma),(T,\gamma')\} \in \xi\}$. This lets us conclude that $\wr(T_1,k_1) \wedge \cdots \wedge (T_n,k_n)\int_{\gamma,\xi} = \mathsf{true}$ which, by Lemma 7.1.2, implies $\wr C\int_{\gamma,\xi} = \mathsf{true}$. Now, given that $[\![I'[\,P'\,]]\!] \xrightarrow[\texttt{Box},\equiv]{C,a}{}^{I[\,P\,],\,S}_{\infty} [\![I''[\,P''\,]]\!]$ we know that $I_1[\,P_1\,]\gamma \xrightarrow{C,a}_\infty I_2[\,P_2\,]_\gamma$ with $I_2[\,P_2\,] \equiv_b I''[\,P''\,]$. From this, given that $\wr C\int_{\gamma,\xi} = \mathsf{true}$, applying rules r11 (if $a = (T,U)$) or r12 (if $a = \tau$) of Table 3.6 and adding context by applying rules r18-l and r18-r of Table 3.8 we obtain $(B,E,\xi) \rightarrow_\infty (B',E,\xi')$. Note that we can compute B' by replacing $I_1[\,P_1\,]_\gamma$ with $I_2[\,P_2\,]_\gamma$ in B while ξ' is equal to ξ if $a = \tau$ and to $\xi\{{}^{(\gamma,T)}\!/\!_{(\gamma,U)}\}$ if $a = (T,U)$. It remains to show that $B2A(B',\xi') \equiv_\kappa E$; we only consider $(a = \tau)$ where $\xi' = \xi$. We observe that if we modify $B2A(B,\xi)$ by replacing the internal state of the z interface of $A\,(\sigma')$ with $I''[\,P''\,]$, we obtain an expression $E' \equiv_\kappa E$. We also observe that B' is equal to B with the exception of the replacement of $I_1[\,P_1\,]_\gamma$ with $I_2[\,P_2\,]_\gamma$ which is congruent to $I''[\,P''\,]$. This, by definition of $B2A$, lets us state that $B2A(B',\xi) \equiv_\kappa E'$ which, by transitivity of $\equiv_\kappa$, implies $B2A(B',\xi) \equiv_\kappa E$.

If r belongs to $R^{S,\mathrm{INT}}_\infty$ it has the following form:

$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \lfloor b_1 \rfloor\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \lfloor b_2 \rfloor\right) \rightarrow_\infty$$
$$\mathsf{sub}(\lfloor I'_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \lfloor b_1 \rfloor^1\right), \mathsf{sub}(\lfloor I'_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \lfloor b_2 \rfloor^1\right)$$

In this case the mixture $B2A(B,\xi)$ contains two agents a_1 and a_2 matching the first and the second agent of the lhs of the rule. This, by definition of $B2A$, implies that in B we have two boxes $I''_1[\,P''_1\,]_{\gamma_1}$ and $I''_2[\,P''_2\,]_{\gamma_2}$ such that $I''_i[\,P''_i\,] \equiv_b I'_i[\,P'_i\,]$ and $(\lfloor b_i \rfloor, T_i)^{r_i} \in \lfloor I'_i \rfloor$ and $(\gamma_i, T_i) \notin \xi$ for $i \in \{1,2\}$ and $\alpha_b(T_1,T_2) = \infty$. These observations let us state that applying the rules r13, r14, r18-l and r18-r we obtain $(B,E,\xi) \rightarrow_\infty (B,E,\xi \cup \{\{(\gamma_1,T_1),(\gamma_2,T_2)\})\}$. It remains to verify that $(B,E,\xi \cup \{\{(\gamma_1,T_1),(\gamma_2,T_2)\})\} \equiv_\kappa E$. We note that if we manipulate $B2A(B,\xi)$ by substituting the binding state of the interfaces $\lfloor b_1 \rfloor$ and $\lfloor b_2 \rfloor$ of the agents a_1 and a_2 from ϵ_λ to $ln(\{(T_1,\gamma_1),(T_2,\gamma_2)\})$, we obtain an expression $E' \equiv_\kappa E$. At this point, by definition of $B2A$, we note that $B2A(B,\xi \cup \{\{(\gamma_1,T_1),(\gamma_2,T_2)\}\}) = E'$ which implies $B2A(B,\xi \cup \{\{(\gamma_1,T_1),(\gamma_2,T_2)\}\}) \equiv_\kappa E$.

□

In the next lemma we prove that all the rules belonging to R^S and applicable to S' derivative of S, also belong to $R^{S'}$.

Lemma 7.1.4 *Given $S = (B,E,\xi)$, $S' = (B',E,\xi')$, $S2\kappa(S) = (\mathtt{Agent},\Sigma,R^S)$, $S2\kappa(S') = (\mathtt{Agent}',\Sigma',R^{S'})$, $r = E_1 \rightarrow_p E_2 \in R^S$ and $B2A(B',\xi') \vDash_\kappa E_1$ then $r \in R^{S'}$.*

Proof. We only prove the result for $r \in R_{\infty}^{S,\mathrm{INT}}$. In this case r has the following form:

$$\mathsf{sub}(\lfloor I' \rfloor)\left(z_{[\![I'[\,P'\,]]\!]}, \sigma\right) \rightarrow_{\infty} \mathsf{sub}(\lfloor I' \rfloor)\left(z_{[\![I''[\,P''\,]]\!]}, \sigma\right)$$

and it holds that

$$\exists I[\,P\,] \in \mathit{getBox}(S) \text{ s.t. } [\![I'[\,P'\,]]\!] \xrightarrow[\mathtt{Box},\equiv]{C,a}{}^{I[\,P\,],\,S}_{\infty} [\![I''[\,P''\,]]\!]$$
$$\text{with } a \in \{\tau, (T,U)\} \wedge \mathit{it}(\sigma, \lfloor I' \rfloor) \in \mathit{icg}(C)$$

If $B2A(B', \xi') \vDash_{\kappa} E_1$ then, by definition of $B2A$, in B' we have a box $I_1[\,P_1\,]_{\gamma}$ such that $I_1[\,P_1\,] \equiv_b I'[\,P'\,]$. This implies that that:

$$I_1[\,P_1\,] \in \mathit{getBox}(S') \text{ and } [\![I'[\,P'\,]]\!] \xrightarrow[\mathtt{Box},\equiv]{C,a}{}^{I_1[\,P_1\,],\,S}_{\infty} [\![I''[\,P''\,]]\!]$$
$$\text{with } a \in \{\tau, (T,U)\} \wedge \mathit{it}(\sigma, \lfloor I' \rfloor) \in \mathit{icg}(C)$$

which lets us conclude that $r \in R^{S'}$.

□

We now have the lemmas necessary to show a completeness result regarding high priority rules. We cannot extend it to all the rules because κ-calculus semantics ignores priorities. Therefore we can apply a rule with low priority even in the case it is concurrent with a high priority one. To modify κ-calculus in order to prevent this behaviour is trivial, however it prevents the possibility to exploit the result of [33]. Therefore, for the moment we keep the original κ-calculus semantics. In the future, in order to improve the precision of our analysis, we will extend the work of [33] to deal with prioritization.

Theorem 7.1.5 *If $S = (B, E, \xi) \in ds(S_o)$, $S2\kappa(S_o) = (\mathtt{Agent}_o, \Sigma_o, R^{S_o})$, $B2A(B, \xi) \xrightarrow[\kappa]{r_o}_{\infty} E'$ and $r_o \in R^{S_o}$ then $\exists (B', E, \xi') : (B, E, \xi) \rightarrow_{\infty} (B', E, \xi')$ with $B2A(B', \xi') \equiv_{\kappa} E'$.*

Proof. By Lemma 7.1.3 we know that, given $S2\kappa(S) = (\mathtt{Agent}, \Sigma, R^S)$, if $B2A(B, \xi) \xrightarrow[\kappa]{r}_{\infty} E'$ with $r \in R^S$ then $\exists (B', E, \xi') : (B, E, \xi) \rightarrow_{\infty} (B', E, \xi')$ with $B2A(B', \xi') \equiv_{\kappa} E$. In order to conclude we prove that $r_0 \in R^{S_o}$. We observe that if $r_o = E_1 \rightarrow_{\infty} E_2$ then $B2A(B, \xi) \vDash_{\kappa} E_1$, moreover S is associated with the same set of events E of all its derivatives and in particular of S'. These observations, by Lemma 7.1.4, imply that $r_0 \in R^{S_o}$ and thus we can conclude with the desired result.

□

Lemma 7.1.6 states that the inclusion of the transition systems of two boxes with respect to two different systems depends on the inclusion of their state spaces.

Lemma 7.1.6 *If $ds(I'[\,P'\,], S') \subseteq ds(I[\,P\,], S)$ then $\tilde{\Phi}^{I'[\,P'\,],S'}_{\mathtt{Box}} \subseteq \tilde{\Phi}^{I[\,P\,],S}_{\mathtt{Box}}$.*

Proof. In order to get the result we show that $\xrightarrow[\mathtt{Box}]{}^{I'[\,P'\,],S'} \subseteq \xrightarrow[\mathtt{Box}]{}^{I[\,P\,],S}$. By definition $\xrightarrow[\mathtt{Box}]{}^{I'[\,P'\,],S'}$ is the biggest subset of $\xrightarrow[\mathtt{Box}]{}$ such that, if it contains the element $(I_1[\,P_1\,], r_s, C, a, I_2[\,P_2\,])$, then $I_1[\,P_1\,], I_2[\,P_2\,]$ belong to $ds(I'[\,P'\,], S')$. Instead $\xrightarrow[\mathtt{Box}]{}^{I[\,P\,],S}$ is the biggest subset of $\xrightarrow[\mathtt{Box}]{}$ such that, if it contains $(I_1[\,P_1\,], r_s, C, a, I_2[\,P_2\,])$, then $I_1[\,P_1\,], I_2[\,P_2\,]$ belong to $ds(I[\,P\,], S)$. From

these definitions it follows immediately that, if $ds(I'[\,P'\,], S') \subseteq ds(I[\,P\,], S)$, then $\xrightarrow[\texttt{Box}]{}^{I'[\,P'\,],S'} \subseteq \xrightarrow[\texttt{Box}]{}^{I[\,P\,],S}$.

□

With the following lemma we show the relation between the state space of boxes which are source and destination of the same transition.

Lemma 7.1.7 *If* $I_1[\,P_1\,] \xrightarrow[\texttt{Box}]{C,a}_{r_s} I_2[\,P_2\,]$ *and* $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{fn}(S_1))$ *and* $\mathsf{fn}(S_2) \subseteq \mathsf{fn}(S_1)$ *then* $ds(I_2[\,P_2\,], S_2) \subseteq ds(I_1[\,P_1\,], S_1)$.

Proof. By Corollary 5.3.9 we know that if $I_1'[\,P_1'\,] = I_1[\,P_1\,]$ and $\mathsf{fn}(S) \cap \mathsf{sub}(I_1') = \emptyset$ then $I_1'[\,P_1'\,] \xrightarrow[\texttt{Box}]{C,a}_{r_s} I_2'[\,P_2'\,]$ with $I_2'[\,P_2'\,] = I_2[\,P_2\,]$. Given that $\mathsf{fn}(S_1) \cap \mathsf{sub}(I_1') = \emptyset$ and $\mathsf{sub}(I_2') = \mathsf{sub}(I_1')$ and $\mathsf{fn}(S_2) \subseteq \mathsf{fn}(S_1)$ we know that $\mathsf{fn}(S_2) \cap \mathsf{sub}(I_2') = \emptyset$. We have now the hypothesis to apply the result of Lemma 5.3.10 which let us state that $pds(I_i'[\,P_i'\,], S_i) = ds(I_i'[\,P_i'\,], S_i)$ for $i \in \{1, 2\}$. Given that $I_i'[\,P_i'\,] = I_i[\,P_i\,]$, by definition of ds, we know that $ds(I_i'[\,P_i'\,], S_i) = ds(I_i[\,P_i\,], S_i)$ for $i \in \{1, 2\}$. Therefore, in order to achieve the desired result, we can prove that $pds(I_2'[\,P_2'\,]) \subseteq pds(I_1'[\,P_1'\,])$ in place of $ds(I_2[\,P_2\,], S_2) \subseteq ds(I_1[\,P_1\,], S_1)$. We can obtain this observing that if $I[\,P\,] \in pds(I_2'[\,P_2'\,], S_2)$ then $I_2'[\,P_2'\,] \xrightarrow[\texttt{Box}]{C_2,a_2}_{r_2} \ldots \xrightarrow[\texttt{Box}]{C_n,a_n}_{r_n} I[\,P\,]$ where $(a_i \in \{T!n, T?n\} \Rightarrow n \in \mathsf{fn}(S_2))$ for $i \in [2, n]$. This, given that $\mathsf{fn}(S_2) \subseteq \mathsf{fn}(S_1)$ and $I_1'[\,P_1'\,] \xrightarrow[\texttt{Box}]{C,a}_{r_s} I_2'[\,P_2'\,]$ where $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{fn}(S_1))$, lets us state that $I_1'[\,P_1'\,] \xrightarrow[\texttt{Box}]{C,a}_{r_s} I_2'[\,P_2'\,] \xrightarrow[\texttt{Box}]{C_2,a_2}_{r_2} \ldots \xrightarrow[\texttt{Box}]{C_n,a_n}_{r_n} I[\,P\,]$ where $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{fn}(S_1))$ which implies $I[\,P\,] \in pds(I_1'[\,P_1'\,], S_1)$. Therefore we have proved that $I[\,P\,] \in pds(I_2'[\,P_2'\,], S_2) \Rightarrow I[\,P\,] \in pds(I_1'[\,P_1'\,], S_1)$ which implies $pds(I_2'[\,P_2'\,], S_2) \subseteq pds(I_1'[\,P_1'\,], S_1)$ and thus $ds(I_2[\,P_2\,], S_2) \subseteq ds(I_1[\,P_1\,], S_1)$.

□

In the next lemma we show that if two systems are source and destination of the same transition, the transition system of the boxes of the second is included in that of the first.

Lemma 7.1.8 *If* $S \rightarrow_r S'$ *then* $\tilde{\Phi}^{S'}_{\texttt{Box}} \subseteq \tilde{\Phi}^{S}_{\texttt{Box}}$.

Proof. By definition $\tilde{\Phi}^{S}_{\texttt{Box}} = \bigcup_{I[\,P\,] \in getBox(S)} \tilde{\Phi}^{I[\,P\,],S}_{\texttt{Box}}$ and $\tilde{\Phi}^{S'}_{\texttt{Box}} = \bigcup_{I[\,P\,] \in getBox(S')} \tilde{\Phi}^{I[\,P\,],S'}_{\texttt{Box}}$. This implies that, in order to obtain the result, we can verify that:

$$\forall I'[\,P'\,] \in getBox(S'), \exists I[\,P\,] \in getBox(S) : \tilde{\Phi}^{I'[\,P'\,],S'}_{\texttt{Box}} \subseteq \tilde{\Phi}^{I[\,P\,],S}_{\texttt{Box}}$$

Thanks to the result of Lemma 7.1.6 this can be shown by proving that

$$\forall I'[\,P'\,] \in getBox(S'), \exists I[\,P\,] \in getBox(S) : ds(I'[\,P'\,], S') \subseteq ds(I[\,P\,], S)$$

We observe that, by Lemma 5.2.11 we know that, between the boxes which appear in S and those which appear in S' there is the following relation: if $get(\gamma, S') = I'[\,P'\,]_\gamma$ then one of the following holds:

1. $I'[\,P'\,] \in getBox(S)$.

2. $get(\gamma, S) = I[\,P\,]_\gamma$ and $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_{r_s} I'[\,P'\,]$ and $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{fn}(S))$.

In the first case we conclude immediately with the desired result. In the second case we observe that, by Lemma 5.1.3, $\mathsf{fn}(S') \subseteq \mathsf{fn}(S)$ and that $I[\,P\,] \xrightarrow[\texttt{Box}]{C,a}_{r_s} I'[\,P'\,]$ where $(a \in \{T!n, T?n\} \Rightarrow n \in \mathsf{fn}(S))$; therefore, by Lemma 7.1.7 we state that $ds(I'[\,P'\,], S') \subseteq ds(I[\,P\,], S)$. The last observation, by Lemma 7.1.6, lets us conclude with the desired result.

□

We now extend the previous result to all the derivatives of a system.

Corollary 7.1.9 *If $S \in ds(S_o)$ then $\tilde{\Phi}^S_{\texttt{Box}} \subseteq \tilde{\Phi}^{S_o}_{\texttt{Box}}$*

Sketch. It follows immediately by induction on the number of transitions which are necessary in order to reach S starting from S_o. The proof is based on the result of Lemma 7.1.8.

□

In the next lemma we show that the set of rules generated by applying the function $S2\kappa$ to a system, includes all the sets generated by $S2\kappa$ applied to the derivatives of the same system.

Lemma 7.1.10 *Given a well-formed system S_o, if $S \in ds(S_o)$ then $R^S \subseteq R^{S_o}$.*

Proof. By Corollary 7.1.9 we know that if $S \in ds(S_o)$ then $\tilde{\Phi}^S_{\texttt{Box}} \subseteq \tilde{\Phi}^{S_o}_{\texttt{Box}}$. This implies that if $S \in ds(S_o)$ then $\tilde{\Phi}^S_{\texttt{Box},\equiv} \subseteq \tilde{\Phi}^{S_o}_{\texttt{Box},\equiv}$. After this observation we consider one by one the clauses of Table 7.3 and we show that if $r \in R^S$ then $r \in R^{S_o}$.

In the cases [INT], [UCOM] and [BCOM] the result immediately follows observing that, thanks to the inclusion $\tilde{\Phi}^S_{\texttt{Box},\equiv} \subseteq \tilde{\Phi}^{S_o}_{\texttt{Box},\equiv}$, we know that if $[\![I_1[\,P_1\,]]\!] \xrightarrow[\texttt{Box},\equiv]{C,a}{}^{I[\,P\,],\,S}_{\infty} [\![I_2[\,P_2\,]]\!]$ with $I[\,P\,] \in getBox(S)$ then $[\![I_1[\,P_1\,]]\!] \xrightarrow[\texttt{Box},\equiv]{C,a}{}^{I_o[\,P_o\,],\,S_o}_{\infty} [\![I_2[\,P_2\,]]\!]$ with $I_o[\,P_o\,] \in getBox(S_o)$.

In the case [BIND] and [UNB] the desired result follows immediately observing that if $\tilde{\Phi}^S_{\texttt{Box},\equiv} \subseteq \tilde{\Phi}^{S_o}_{\texttt{Box},\equiv}$ then it holds that $I_1[\,P_1\,] \in ds(I_o[\,P_o\,], S_o)$ with $I_o[\,P_o\,] \in getBox(S_o)$ implies $I_1[\,P_1\,] \in ds(I[\,P\,], S)$ with $I[\,P\,] \in getBox(S)$.

In the case [EV] we can conclude with the same observations done for the cases [BIND] and [UNB] plus the fact that if $S \in ds(S_o)$ then S and S_o are associated with the same set of events.

□

Lemma 7.1.11 states that, if a system performs a τ-action, the same can do its encoding in κ-calculus and with the same priority.

Lemma 7.1.11 *Given $S = (B, E, \xi)$, $S' = (B', E, \xi')$ and $S2\kappa(S) = (\texttt{Agent}, \Sigma, R^S)$, if $S \xrightarrow{l}_r S'$ then one of the following holds:*

1. *$l = (\gamma, T!n)$ and $get(\gamma, S) = I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow[\texttt{Box}]{C,T!n}_0 I'[\,P'\,]_\gamma$ with $get(\gamma, S') = I'[\,P'\,]_\gamma$ and $\gamma' \neq \gamma \Rightarrow get(\gamma, S) = get(\gamma, S')$ and $\wr C \int_{\gamma,\xi} = \mathsf{true}$ and $n \in \mathsf{fn}(S)$.*

2. *$l = (\gamma, T?n)$ and $get(\gamma, S) = I[\,P\,]_\gamma$ and $I[\,P\,]_\gamma \xrightarrow[\texttt{Box}]{C,T?n}_0 I'[\,P'\,]_\gamma$ with $get(\gamma, S') = I'[\,P'\,]_\gamma$ and $\gamma' \neq \gamma \Rightarrow get(\gamma, S) = get(\gamma, S')$ and $\wr C \int_{\gamma,\xi} = \mathsf{true}$.*

3. $l = (\gamma, T)$ *and* $get(\gamma, S) = I[\,P\,]_\gamma$ *and* $\exists b : (b,\, T)^{r''} \in I$.

4. $l = \tau$ *and* $B2A(B, \xi) \xrightarrow[\kappa]{r'}_p E' \equiv_\kappa B2A(B, \xi)$ *with* $r \neq \infty \Rightarrow p = \not\infty \wedge r' \in R^S_{\not\infty}$ *and* $r = \infty \Rightarrow p = \infty \wedge r' \in R^S_\infty$.

Proof. By induction on the length of the derivation of $S \xrightarrow{l}_r S'$ (rules of Tables 3.6 and 3.8). We consider only some representative cases.

base case r9 :

$$\frac{I[\,P\,]_\gamma \xrightarrow{C,T!n}_0 I[\,P'\,]_\gamma}{(I[\,P\,]_\gamma, E, \xi) \xrightarrow{(\gamma,T!n)}_0 (I[\,P'\,]_\gamma, E, \xi)} \quad \text{provided } \wr C \int_{\gamma,\xi} = \mathsf{true}$$

From the premises of the rule $I[\,P\,]_\gamma \xrightarrow{C,T!n}_0 I[\,P'\,]_\gamma$ and from the side condition $\wr C \int_{\gamma,\xi} = \mathsf{true}$. Moreover, by Lemma 5.1.3, if $l = (\gamma, T!n)$ then $n \in (I[\,P\,]_\gamma, E, \xi)$. To conclude we observe that if $\gamma' \neq \gamma$ then it holds that $get(\gamma', (I[\,P\,]_\gamma, E, \xi)) = \bot = get(\gamma', (I[\,P'\,]_\gamma, E, \xi))$.

step case r14 :

$$\frac{(B_1, E, \xi) \xrightarrow{(\gamma_1,T_1)}_0 (B_1, E, \xi) \qquad (B_2, E, \xi) \xrightarrow{(\gamma_2,T_2)}_0 (B_2, E, \xi)}{(B_1 \parallel B_2, E, \xi) \xrightarrow{\tau}_{\alpha_b(T_1,T_2)} (B_1 \parallel B_2, E, \xi \cup \{\{(\gamma_1, T_1), (\gamma_2, T_2)\}\})}$$
$$\text{provided: } \alpha_b(T_1, T_2) > 0 \wedge \forall L \in \xi : \{(\gamma_1, T_1), (\gamma_2, T_2)\} \cap L = \emptyset$$

By inductive hypothesis $get(\gamma_i, (B_1, E, \xi)) = I_i[\,P_i\,]_{\gamma_i}$ and $\exists b_i : (b_i,\, T_i)^{r_i} \in I_i$. Moreover, from the side condition of the rule $\alpha_b(T_1, T_2) \neq 0$ and $\forall L \in \xi : \{(\gamma_1, T_1), (\gamma_2, T_2)\} \cap L = \emptyset$.

From these observations $I_i[\,P_i\,] \in getBox((B_1, E, \xi))$, $[\![I_i[\,P_i\,]]\!] \in ds(I_i[\,P_i\,], S)$, $(b_i,\, T_i)^{r_i} \in I_i$, for $i \in \{1, 2\}$ and $\alpha_b(T_1, T_2) \neq 0$. Because of the [BIND] rule of Table 7.3, this implies that r'

$$\mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\,P_1\,]]\!]}, \lfloor b_1 \rfloor\right), \mathsf{sub}(\lfloor I_2 \rfloor)\left(z_{[\![I_2[\,P_2\,]]\!]}, \lfloor b_2 \rfloor\right) \rightarrow_\infty$$
$$\mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I'_1[\,P'_1\,]]\!]}, \lfloor b_1 \rfloor^1\right), \mathsf{sub}(\lfloor I_2 \rfloor)\left(z_{[\![I'_2[\,P'_2\,]]\!]}, \lfloor b_2 \rfloor^1\right)$$

belongs to R^S_p where $p = \infty$ if $r = \infty$ and $p = \not\infty$ if $r \neq \infty$.

Given that $get(\gamma_i, (B_1, E, \xi)) = I_i[\,P_i\,]_{\gamma_i}$ for $i \in \{1, 2\}$, we observe that, by definition of $B2A$, the expression $B2A(B_1 \parallel B_2, \xi)$ has the following form:

$$A\,(\sigma), \ldots, \mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\,P_1\,]]\!]}, \sigma_1\right), \ldots, \mathsf{sub}(\lfloor I_2 \rfloor)\left(z_{[\![I_2[\,P_2\,]]\!]}, \sigma_2\right), \ldots, A'\,(\sigma') = E$$

where, given that $\forall L \in \xi : \{(\gamma_1, T_1), (\gamma_2, T_2)\} \cap L = \emptyset$ and $(b_i,\, T_i)^{r_i} \in I_i$, the interface $\lfloor b_i \rfloor^{\epsilon_\lambda}$ for $i \in \{1, 2\}$ appears in σ_i. It is easy to observe that such an expression can be involved in the transition $E \xrightarrow[\kappa]{p}_{r'} E'$ with E' which has the following form:

$$A\,(\sigma), \ldots, \mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\,P_1\,]]\!]}, \sigma'_1\right), \ldots, \mathsf{sub}(\lfloor I_2 \rfloor)\left(z_{[\![I_2[\,P_2\,]]\!]}, \sigma'_2\right), \ldots, A'\,(\sigma')$$

where all the interfaces of σ_i' are the same of those of σ_i with the exception of $\lfloor b_i \rfloor^{\epsilon_\lambda}$, whose binding state changes from ϵ_λ to $n \in \mathbb{N}$. Now we observe that $B2A(B_1 \parallel B_2, \xi \cup \{\{(\gamma_1, T_1), (\gamma_2, T_2)\}\})$ has the following form:

$$E \xrightarrow[\kappa]{p}_{r'} A(\sigma), \ldots, \mathsf{sub}(\lfloor I_1 \rfloor)\left(z_{[\![I_1[\,P_1\,]]\!]}, \sigma_1''\right), \ldots, \mathsf{sub}(\lfloor I_2 \rfloor)\left(z_{[\![I_2[\,P_2\,]]\!]}, \sigma_2''\right), \ldots, A'(\sigma')$$

where σ_i'' is equal to σ_i' with the exception of the binding state of the interface $\lfloor b_i \rfloor$ which, in place of being n, is $ln(\{(T_l, \gamma), (T', \gamma')\})$. Therefore, by definition of $\equiv_\kappa$, $E' \equiv_\kappa B2A(B_1 \parallel B_2, \xi \cup \{\{(\gamma_1, T_1), (\gamma_2, T_2)\}\})$ and thus we get the desired result.

□

We now prove some transitivity properties regarding the $\vDash_\kappa$ operator.

Lemma 7.1.12 *The operator $\vDash_\kappa$ is transitive over interface sequences.*

Proof. By induction on the length of the derivation tree of $\sigma \vDash_\kappa \sigma'$.

base case :

$$x_\iota^\lambda \vDash_\kappa x_\iota^\lambda$$

We trivially conclude because the involved sequences are equal.

base case :

$$x_\iota^i \vDash_\kappa x_\iota^-$$

Here we observe that if $\sigma \vDash_\kappa x_\iota^i$ then $\sigma = x_\iota^i, \sigma'$. From this we get immediately the desired result observing that:

$$\frac{x_\iota^i \vDash_\kappa x_\iota^- \qquad \sigma' \vDash_\kappa \varepsilon}{x_\iota^i, \sigma' \vDash_\kappa x_\iota^-}$$

base case :

$$\sigma \vDash_\kappa \varepsilon$$

For all $\sigma' \vDash_\kappa \sigma$ we conclude immediately observing that $\sigma' \vDash_\kappa \varepsilon$.

step case :

$$\frac{s \vDash_\kappa s_l \qquad \sigma \vDash_\kappa \sigma_l}{s, \sigma \vDash_\kappa s_l, \sigma_l}$$

Let us suppose to have a sequence σ' such that $\sigma' \vDash_\kappa s, \sigma$. In order to get this match it must hold that $\sigma' = s', \sigma''$, $s' \vDash_\kappa s$ and $\sigma'' \vDash_\kappa \sigma$.

By inductive hypothesis we have that $s' \vDash_\kappa s_l$ and that $\sigma'' \vDash_\kappa \sigma_l$ which lets us conclude with:

$$\frac{s' \vDash_\kappa s_l \qquad \sigma'' \vDash_\kappa \sigma_l}{\sigma' = s', \sigma'' \vDash_\kappa s_l, \sigma_l}$$

□

Lemma 7.1.13 *The operator $\vDash_\kappa$ is transitive over expressions.*

Sketch. By induction on the length of the derivation tree of $E \vDash_\kappa E'$ and exploiting the result of Lemma 7.1.12. The proof is similar to that of Lemma 7.1.12.

□

Lemma 7.1.14 *If $E_1 \vDash_\kappa E_2$ and $E_2' \equiv_\kappa E_2$ then $\exists E_1' \equiv_\kappa E_1 : E_1' \vDash_\kappa E_2'$.*

Proof. By induction on the number of times the transitivity has been applied.

base case :

$$E, A\left(\sigma, s, s', \sigma'\right), E' \equiv_\kappa E, A\left(\sigma, s', s, \sigma'\right), E'$$

If $E_1 \vDash_\kappa E, A\left(\sigma, s, s', \sigma'\right), E'$ then $E_1 = E_3, A\left(\sigma_3, s_3, s_3', \sigma_3', \sigma_3''\right), E_3'$ with $E = a_1, \ldots, a_n$ and $E_3 = a_{3,1}, \ldots, a_{3,n}$ and $a_i \vDash_\kappa a_3, i$ for $i \in [1, n]$ and $E_3' \vDash_\kappa E'$ and $A\left(\sigma_3, s_3, s_3', \sigma_3', \sigma_3''\right) \vDash_\kappa A\left(\sigma, s, s', \sigma'\right)$ which implies $\sigma_3 \vDash_\kappa \sigma$, $s_3 \vDash_\kappa s$, $s_3' \vDash_\kappa s'$ and $\sigma_3', \sigma_3'' \vDash_\kappa \sigma'$. That is enough to state that $E_1' = E_3, A\left(\sigma_3, s_3', s_3, \sigma_3', \sigma_3''\right), E_3' \vDash_\kappa E, A\left(\sigma, s', s, \sigma'\right), E'$. After that we can conclude by observing that $E_1' \equiv_\kappa E_1$.

base case :

$$E, a, a', E' \equiv_\kappa E, a', a, E'$$

If $E_1 \vDash_\kappa E, a, a', E'$ then $E_1 = E_3, a_3, a_3', E_3'$ with $E = a_1, \ldots, a_n$ and $E_3 = a_{3,1}, \ldots, a_{3,n}$ and $a_i \vDash_\kappa a_{3,i}$ for $i \in [1, n]$, $a_3 \vDash_\kappa a'$, $a_3' \vDash_\kappa a_3$ and $E_3' \vDash_\kappa E'$. These matchings imply that $E_1' = E_3, a_3', a_3, E_3' \vDash_\kappa E, a', a, E'$. Observing that $E_1' \equiv_\kappa E_1$, we can conclude with the desired result.

base case :

$$E \equiv_\kappa E, \emptyset$$

If $E_1 \vDash_\kappa E$ we know that $E_1 = E_3, E_3'$ where $E = a_1, \ldots, a_n$ and $E_3 = a_{3,1}, \ldots, a_{3,n}$ and $a_i \vDash_\kappa a_{3,i}$ for $i \in [1, n]$. This implies that $E_1' = E_3, \emptyset, E_3' \vDash_\kappa E, \emptyset$. We can conclude observing that $E_1' \equiv_\kappa E_1$.

base case :

$$i, j \in \mathbb{N} \wedge i \text{ does not occur in } E \Rightarrow E \equiv_\kappa E[i/j]$$

$E_1 \vDash_\kappa E$ implies that $E_1 = E_3, E_3'$ where $E = a_1, \ldots, a_n$ and $E_3 = a_{3,1}, \ldots, a_{3,n}$ and $a_i \vDash_\kappa a_{3,i}$ for $i \in [1, n]$. Now we create an $E_1'' = E_1[j/l]$ where l does not occur in E_1. Then we perform a second substitution to E_1'' and we obtain $E_1' = E_1[i/j]$. We have $E_1' = E_3[j/l][i/j], E_3'[j/l][i/j]$ and thus $E_1' \vDash_\kappa E[j/l][i/j]$. We can conclude observing that $E[j/l][i/j] = E[i/j]$ (given that j does not occur in E) and that $E_1' \equiv_\kappa E_1$.

base case

$$i \in \mathbb{N} \wedge i \text{ occurs once in } E \Rightarrow E[\epsilon_\lambda/i] \equiv_\kappa E$$

Similar to the previous case.

step case :

$$E \equiv_\kappa E'' \equiv_\kappa E' \Rightarrow E \equiv_\kappa E'$$

Given that $E_1 \vDash_\kappa E$, we can apply the inductive hypothesis to $E \equiv_\kappa E''$ and obtain that exists $E_1'' \equiv_\kappa E_1 : E_1'' \vDash_\kappa E''$. Given that $E_1'' \vDash_\kappa E''$ we can apply the inductive hypothesis to $E'' \equiv_\kappa E'$ and obtain that exists $E_1' \equiv_\kappa E_1'' : E_1' \vDash_\kappa E'$. After that we can conclude by observing that $E_1' \equiv_\kappa E_1$ by transitivity of $\equiv_\kappa$.

□

Corollary 7.1.15 *If $E_1' \in [\![E_1]\!], E_1' \vDash_\kappa E_2$ and $E \vDash_\kappa E_1$ then $\exists E' \in [\![E]\!] : E' \vDash_\kappa E_2$.*

Proof. If $E \vDash_\kappa E_1$ and $E_1' \in [\![E_1]\!]$, by Lemma 7.1.14, it exists $E' \in [\![E]\!] : E' \vDash_\kappa E_1'$. This let us conclude by transitivity of the $\vDash_\kappa$ operator (see Lemma 7.1.13), that $E' \vDash_\kappa E_2$.

□

Finally we prove the soundness of our encoding.

Theorem 7.1.16 *If $S = (B, E, \xi) \in ds(S_o)$ and $S2\kappa(S_o) = (\mathtt{Agent}_o, \Sigma_o, R^{S_o})$ then*

$$S \rightarrow_r S' \Rightarrow B2A(B, \xi) \xrightarrow[\kappa]{r'}_p E' \equiv_\kappa B2A(B', \xi')$$

where $S' = (B', E, \xi')$ and $r \in \mathbb{R} \Rightarrow p = \not\infty$ and $r = \infty \Rightarrow p = \infty$ and $r' \in R^{S_o}$.

Proof. By definition of $\rightarrow$, if $S \rightarrow_r S'$ then $S \xrightarrow{\tau}_r S'$. This, by Lemma 7.1.11, implies that $B2A(B, \xi) \xrightarrow[\kappa]{r''}_p E \equiv_\kappa B2A(B', \xi')$ and

1. if $r = \infty$ then $p = \infty$ and $r'' \in R^S_\infty$;
2. if $r \in \mathbb{R}$ then $p = \not\infty$, and $r'' \in R^S_{\not\infty}$.

By Lemma 7.1.10, we know that $r'' \in R^S$ implies $r'' \in R^{S_o}$. If $r'' \in R^S_\infty$ we get the desired result because this implies that $r'' \in R^S$ and thus $r'' \in R^{S_o}$; instead if $r'' \in R^S_{\not\infty}$ we have to verify that, given $r'' = E_1 \rightarrow_{\not\infty} E_1'$, it does not exist $E_3 \in [\![E_1]\!], r_2 = E_2 \rightarrow_\infty E_2' \in R^S_\infty$ such that $E_3 \vDash_\kappa E_2$. In order to get this result we suppose by contradiction that E_3 and $E_2 \rightarrow_\infty E_2'$ exist. Given that $E_3 \in [\![E_1]\!]$, $E_3 \vDash_\kappa E_2$ and $B2A(B, \xi) \vDash_\kappa E_1$ we can apply the result of Corollary 7.1.15 and state that $\exists E'' \in [\![B2A(B, \xi)]\!] : E'' \vDash_\kappa E_2$. This implies that $E_2 \rightarrow_\infty E_2'$ can be applied to $B2A(B, \xi)$ and thus we can obtain the transition $B2A(B, \xi) \xrightarrow[\kappa]{r_2}_\infty E''[E_2']$. However, by Theorem 7.1.5, this implies that $\exists S''$ such that $S \rightarrow_\infty S''$ but this is absurd because such a transition would make the existence of $S \rightarrow_r S'$ impossible. Therefore we conclude that E_3 and r_2 do not exist and thus that, even in the case $r'' \in R^S_{\not\infty}$, r'' belongs to R^S and thus to R^{S_o}.

□

At this point given $S2\kappa((B_o, E, \xi_o)) = (\mathtt{Agent}_o, \Sigma_o, R^{S_o})$, by Theorem 7.1.5, we can state that if $(B, E, \xi) \in ds((B_o, E, \xi_o))$ then $B2A(B, \xi) \xrightarrow[\kappa]{r_1}_{p_1} \ldots \xrightarrow[\kappa]{r_n}_{p_n} E'$ with $E' \equiv_\kappa B2A(B, \xi)$ and $r_1, \ldots, r_n \in R^{S_o}$. Thus, exploiting Observation 7.1.1, in order to compute an over-approximation of the complexes which appear during the possible evolutions of (B_o, E, ξ_o), we can apply the abstract interpretation method of [33] to the mixture $B2A(B_o, \xi_o)$ and the rules of R^{S_o}.

$$
\begin{array}{rcl}
k_1 &=& A\left(z_{[\![B_2]\!]}\right) \to_\infty A\left(z_{[\![B_3]\!]}\right) \\
k_2 &=& A\left(z_{[\![B_3]\!]}\right) \to_\infty A\left(z_{[\![B_4]\!]}\right) \\
k_3 &=& A\left(z_{[\![B_4]\!]}\right) \to_\infty A\left(z_{[\![B_6]\!]}\right) \\
k_4 &=& A\left(z_{[\![B_4]\!]}, \lfloor b \rfloor^-\right) \to_\infty A\left(z_{[\![B_7]\!]}, \lfloor b \rfloor^-\right) \\
k_5 &=& A\left(z_{[\![B_5]\!]}\right) \to_\infty A\left(z_{[\![B_6]\!]}\right) \\
k_6 &=& A\left(z_{[\![B_5]\!]}, \lfloor p \rfloor^-\right) \to_\infty A\left(z_{[\![B_8]\!]}, \lfloor p \rfloor^-\right) \\
k_7 &=& A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor^-\right) \to_\infty A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor^-\right) \\
k_8 &=& A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor^-\right) \to_\infty A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor^-\right) \\
k_9 &=& A\left(z_{[\![B_7]\!]}, \lfloor b \rfloor\right) \to_\infty A\left(z_{[\![B_9]\!]}, \lfloor b \rfloor\right) \\
k_{10} &=& A\left(z_{[\![B_7]\!]}\right) \to_\infty A\left(z_{[\![B_{10}]\!]}\right)
\end{array}
\qquad
\begin{array}{rcl}
k_{11} &=& A\left(z_{[\![B_8]\!]}, \lfloor p \rfloor\right) \to_\infty A\left(z_{[\![B_9]\!]}, \lfloor p \rfloor\right) \\
k_{12} &=& A\left(z_{[\![B_8]\!]}\right) \to_\infty A\left(z_{[\![B_{11}]\!]}\right) \\
k_{13} &=& A\left(z_{[\![B_9]\!]}\right) \to_\infty A\left(z_{[\![B_4]\!]}\right) \\
k_{14} &=& A\left(z_{[\![B_9]\!]}\right) \to_\infty A\left(z_{[\![B_5]\!]}\right) \\
k_{15} &=& A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor\right) \to_\infty A\left(z_{[\![B_5]\!]}, \lfloor b \rfloor\right) \\
k_{16} &=& A\left(z_{[\![B_{10}]\!]}, \lfloor p \rfloor^-\right) \to_\infty A\left(z_{[\![B_{12}]\!]}, \lfloor p \rfloor^-\right) \\
k_{17} &=& A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor\right) \to_\infty A\left(z_{[\![B_4]\!]}, \lfloor p \rfloor\right) \\
k_{18} &=& A\left(z_{[\![B_{11}]\!]}, \lfloor b \rfloor^-\right) \to_\infty A\left(z_{[\![B_{12}]\!]}, \lfloor b \rfloor^-\right) \\
k_{19} &=& A\left(z_{[\![B_{12}]\!]}, \lfloor p \rfloor\right) \to_\infty A\left(z_{[\![B_7]\!]}, \lfloor p \rfloor\right) \\
k_{20} &=& A\left(z_{[\![B_{12}]\!]}, \lfloor b \rfloor\right) \to_\infty A\left(z_{[\![B_8]\!]}, \lfloor b \rfloor\right)
\end{array}
$$

Table 7.5 – Rules of R^S_∞

7.2 Applying the Encoding to the Actin Case Study

The application of the encoding to the actin case study, and in particular to the system $S = (\mathcal{L}(\mathtt{A}_1 \parallel \ldots \parallel \mathtt{A}_n), \emptyset, \emptyset)$, is based on the transition system $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$ we computed in section 5.4 and whose representation appears in Figure 5.1.

In order to perform the encoding we firstly apply the function $S2\kappa$ to S and then $B2A$ to $(\mathcal{L}(\mathtt{A}_1 \parallel \ldots \parallel \mathtt{A}_n), \xi)$. The application of $S2\kappa$ to S returns the triple $(\mathtt{Agent}, \Sigma, R^S)$. The set $\mathtt{Agent}$ contains an element $\lfloor \mathtt{sub}(I) \rfloor$ for each box $I[\,P\,] \in getBox(S)$. Given that $getBox(S) = \{\mathtt{A}\}$, the set $\mathtt{Agent}$ only contains the element $\{\lfloor p \rfloor, \lfloor b \rfloor\}$. For the sake of readability hereafter we use A to refer $\{\lfloor p \rfloor, \lfloor b \rfloor\}$. The function Σ is equal to $A \mapsto \{z, p, b\}$, therefore the agent A is provided with the special interface z and with two interfaces whose names correspond to the subjects of the interfaces of $\mathtt{A}$.

The definition of the set R^S is based on R^S_∞ and $R^S_{\not\infty}$. Rules can be generated for two reasons:

1. An equivalence class of $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$ is involved in a transition associated with simplified rate $\not\infty$ or ∞.
2. In the state space of $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$ there are two equivalence classes whose interfaces have sorts with capabilities allowing their interaction.

The system S does not contemplate sort capabilities with high priority, therefore all the rules of R^S_∞ are of the first kind. Each low priority transition of $\tilde{\Phi}^{\mathtt{A},S}_{\mathtt{Box},\equiv}$ (i.e. those depicted in Figure 5.1 and not associated with simplified rate $\not\infty$), because of the [INT] clause of Table 7.3, is responsible for the generation of one rule of R^S_∞. As a representative example we propose the generation of the rule $A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor^-\right) \to_\infty A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor^-\right)$. This rule exists because of the transition $[\![B_6]\!] \xrightarrow[\mathtt{Box},\equiv]{\langle PF,\otimes\rangle,(BF,BB)}{}^{\mathtt{A},S}_{\infty} [\![B_{10}]\!]$. The name of the agents involved in the rule is A, i.e. the canonical form of the subject set of the interfaces of B_6 (which is equal to that of

$$
\begin{aligned}
k_{21} &= A\left(z_{[\![B_1]\!]}\right) \to_{\not\infty} A\left(z_{[\![B_2]\!]}\right)\\
k_{22} &= A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor\right), A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor\right) \to_{\not\infty} A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor^1\right), A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor^1\right)\\
k_{23} &= A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor^1\right), A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor^1\right) \to_{\not\infty} A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor\right), A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor\right)\\
k_{24} &= A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor\right), A\left(z_{[\![B_{10}]\!]}, \lfloor p \rfloor\right) \to_{\not\infty} A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor^1\right), A\left(z_{[\![B_{10}]\!]}, \lfloor p \rfloor^1\right)\\
k_{25} &= A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor^1\right), A\left(z_{[\![B_{10}]\!]}, \lfloor p \rfloor^1\right) \to_{\not\infty} A\left(z_{[\![B_6]\!]}, \lfloor b \rfloor\right), A\left(z_{[\![B_{10}]\!]}, \lfloor p \rfloor\right)\\
k_{26} &= A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor\right), A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor\right) \to_{\not\infty} A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor^1\right), A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor^1\right)\\
k_{27} &= A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor^1\right), A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor^1\right) \to_{\not\infty} A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor\right), A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor\right)\\
k_{28} &= A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor\right), A\left(z_{[\![B_{11}]\!]}, \lfloor b \rfloor\right) \to_{\not\infty} A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor^1\right), A\left(z_{[\![B_{11}]\!]}, \lfloor b \rfloor^1\right)\\
k_{29} &= A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor^1\right), A\left(z_{[\![B_{11}]\!]}, \lfloor b \rfloor^1\right) \to_{\not\infty} A\left(z_{[\![B_6]\!]}, \lfloor p \rfloor\right), A\left(z_{[\![B_{11}]\!]}, \lfloor b \rfloor\right)\\
k_{30} &= A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor\right), A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor\right) \to_{\not\infty} A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor^1\right), A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor^1\right)\\
k_{31} &= A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor^1\right), A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor^1\right) \to_{\not\infty} A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor\right), A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor\right)\\
k_{32} &= A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor\right), A\left(z_{[\![B_{12}]\!]}, \lfloor p \rfloor\right) \to_{\not\infty} A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor^1\right), A\left(z_{[\![B_{12}]\!]}, \lfloor p \rfloor^1\right)\\
k_{33} &= A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor^1\right), A\left(z_{[\![B_{12}]\!]}, \lfloor p \rfloor^1\right) \to_{\not\infty} A\left(z_{[\![B_{10}]\!]}, \lfloor b \rfloor\right), A\left(z_{[\![B_{12}]\!]}, \lfloor p \rfloor\right)\\
k_{34} &= A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor\right), A\left(z_{[\![B_{12}]\!]}, \lfloor b \rfloor\right) \to_{\not\infty} A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor^1\right), A\left(z_{[\![B_{12}]\!]}, \lfloor b \rfloor^1\right)\\
k_{35} &= A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor^1\right), A\left(z_{[\![B_{12}]\!]}, \lfloor b \rfloor^1\right) \to_{\not\infty} A\left(z_{[\![B_{11}]\!]}, \lfloor p \rfloor\right), A\left(z_{[\![B_{12}]\!]}, \lfloor b \rfloor\right)
\end{aligned}
$$

Table 7.6 – Rules of $R_{\not\infty}$ without the rules which are concurrent with one belonging to R_∞.

A and B_{10}). The interface sequences start with the special interfaces z, which have internal state equal to the equivalent classes involved in the transition. The rest of the sequences is made of the interface $\lfloor b \rfloor^-$ because $icg(C) = \{\{(PF, \otimes)\}\}$ and $it(\lfloor b \rfloor^-, \lfloor I \rfloor) = \{(PF, \otimes)\}$ where $I = \{(p, BF), (b, PF)\}$ is the interface set of B_6. Summing up, this rule states that an agent mapping $[\![B_6]\!]$, if its interface b is bound, can transform in an agent mapping $[\![B_{10}]\!]$. This is the same information carried by the transition $[\![B_6]\!] \xrightarrow[\texttt{Box},\equiv]{\langle PF,\otimes\rangle,(BF,BB)}{}^{\texttt{A},S}_{\infty}\ [\![B_{10}]\!]$. The list of the rules belonging to R^S_∞ is reported in Table 7.5.

Rules of $R^S_{\not\infty}$ are generated through three clauses of Table 7.3 (where ∞ is replaced with $\not\infty$). The [INT] clause generates rule k_{21} of Table 7.6. Its generation is due to the low priority transition of $\tilde{\Phi}^{\texttt{A},S}_{\texttt{Box},\equiv}$ which involves the congruent classes $[\![B_1]\!]$ and $[\![B_2]\!]$. All the other rules exist because of the clauses [BIND] and [UNB]. For example, rule k_{22} encodes the capability of two boxes belonging to $[\![B_6]\!]$ to bind together. The names of the involved agents is again A (i.e. $\{\lfloor p \rfloor, \lfloor b \rfloor\}$) and the internal state of the special interface z is $[\![B_6]\!]$. The other interfaces are those involved in the binding. In the lhs of the rules they occur free, in the rhs they become bound. Note that in $[\![B_6]\!]$ the interfaces involved in the reaction have sorts with positive binding capability. The rules existing because of the [UNB] clause are constructed with a similar procedure. However, here we require the involved interfaces to occur bound in the lhs and free in the rhs and their sort to have positive unbinding capability. Some rules belonging to $R^S_{\not\infty}$ are concurrent with rules of R^S_∞, therefore they do not belong to R^S. It is the case for the rule

1.	$A\left(z_{[\![B_1]\!]}, b, p\right)$	15.	$A\left(z_{[\![B_8]\!]}, b, p^{b.A}\right)$
2.	$A\left(z_{[\![B_2]\!]}, b, p\right)$	16.	$A\left(z_{[\![B_9]\!]}, b, p\right)$
3.	$A\left(z_{[\![B_3]\!]}, b, p\right)$	17.	$A\left(z_{[\![B_{10}]\!]}, b, p\right)$
4.	$A\left(z_{[\![B_4]\!]}, b, p\right)$	18.	$A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p\right)$
5.	$A\left(z_{[\![B_4]\!]}, b^{p.A}, p\right)$	19.	$A\left(z_{[\![B_{10}]\!]}, b, p^{b.A}\right)$
6.	$A\left(z_{[\![B_5]\!]}, b, p\right)$	20.	$A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p^{b.A}\right)$
7.	$A\left(z_{[\![B_5]\!]}, b, p^{b.A}\right)$	21.	$A\left(z_{[\![B_{11}]\!]}, b, p\right)$
8.	$A\left(z_{[\![B_6]\!]}, b, p\right)$	22.	$A\left(z_{[\![B_{11}]\!]}, b^{p.A}, p\right)$
9.	$A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right)$	23.	$A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right)$
10.	$A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right)$	24.	$A\left(z_{[\![B_{11}]\!]}, b^{p.A}, p^{b.A}\right)$
11.	$A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right)$	25.	$A\left(z_{[\![B_{12}]\!]}, b, p\right)$
12.	$A\left(z_{[\![B_7]\!]}, b, p\right)$	26.	$A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p\right)$
13.	$A\left(z_{[\![B_7]\!]}, b^{p.A}, p\right)$	27.	$A\left(z_{[\![B_{12}]\!]}, b, p^{b.A}\right)$
14.	$A\left(z_{[\![B_8]\!]}, b, p\right)$	28.	$A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p^{b.A}\right)$

Table 7.7 – The set $lfp_V POST_v^{R^S}$.

$$E = A\left(z_{[\![B_5]\!]}, b\right), A\left(z_{[\![B_6]\!]}, p\right) \to_{\not\infty} A\left(z_{[\![B_5]\!]}, b^1\right), A\left(z_{[\![B_6]\!]}, p^1\right) = E'$$

In fact E matches $A\left(z_{[\![B_5]\!]}\right)$ and thus the rule is concurrent with $k_5 = A\left(z_{[\![B_5]\!]}\right) \to_\infty A\left(z_{[\![B_6]\!]}\right)$. This implies that if a mixture satisfies E, then it also satisfies the lhs of rule k_5. Therefore the rule $E \to_{\not\infty} E'$ encodes a BlenX binding with low priority which is concurrent with an immediate action. In a BlenX system this binding will never take place and thus we can avoid adding the rule $E \to_{\not\infty} E'$ to R^S. In Table 7.6 we list the rule of $R^S_{\not\infty} \cap R^S$.

The application of *B2A* to the pair (B, ξ) generates a mixture made of a sequence of n agents $A\left(z_{[\![B_1]\!]}, \lfloor b \rfloor, \lfloor p \rfloor\right)$. The name of the agents is A because the canonical form of the interface subjects of `A` is $\{\lfloor p \rfloor, \lfloor b \rfloor\}$, and all the interfaces are free because $\xi = \emptyset$.

Once we have encoded S in κ-calculus, we can apply the abstract interpretation based analysis of Section 2.3.4 to the actin model. In doing so, we generate a finite representation of an over-approximation of the complexes that can appear during the evolution of S. Hereafter, for the sake of space, we avoid explicitly writing the interfaces in their canonical form. As first step we extract the views from $B2A(S, \xi)$. This means to apply the abstraction function to $B2A(B, \xi)$: $\alpha(\alpha_c(B2A(B, \xi)))$. Given that in $B2A(B, \xi)$ we do not have bound interfaces, the generated set of views is trivially $V = \left\{A\left(z_{[\![B_1]\!]}, p, b\right)\right\}$. Now we compute an over-approximation of the views we would obtain by applying the function $\alpha \circ \alpha_c$ to the derivatives of $B2A(B, \xi)$. The set is generated computing $lfp_V POST_v^{R^S}$. In words, we apply, via abstract semantics, the rules of R^S to the elements of V. This process is iterated until we reach a fix-point. In Table 7.7 we

$A\left(z_{[\![B_1]\!]}, b, p\right)$	$\xrightarrow[\kappa]{k_1}{}^{\sharp}\!\!\infty$	$A\left(z_{[\![B_2]\!]}, b, p\right)$
$A\left(z_{[\![B_2]\!]}, b, p\right)$	$\xrightarrow[\kappa]{k_2}{}^{\sharp}\!\!\infty$	$A\left(z_{[\![B_3]\!]}, b, p\right)$
$A\left(z_{[\![B_3]\!]}, b, p\right)$	$\xrightarrow[\kappa]{k_3}{}^{\sharp}\!\!\infty$	$A\left(z_{[\![B_4]\!]}, b, p\right)$
$A\left(z_{[\![B_4]\!]}, b, p\right)$	$\xrightarrow[\kappa]{k_4}{}^{\sharp}\!\!\infty$	$A\left(z_{[\![B_6]\!]}, b, p\right)$
$A\left(z_{[\![B_6]\!]}, b, p\right), A\left(z_{[\![B_6]\!]}, b, p\right)$	$\xrightarrow[\kappa]{k_{22}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right), A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right)$
$A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right) \xrightarrow[\kappa]{k_8}{}^{\sharp}\!\!\infty A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p\right)$		$A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right) \xrightarrow[\kappa]{k_9}{}^{\sharp}\!\!\infty A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right)$
$A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right), A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right)$	$\xrightarrow[\kappa]{k_{22}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right), A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right)$
$A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p\right), A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right)$	$\xrightarrow[\kappa]{k_{31}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_{10}]\!]}, b, p\right), A\left(z_{[\![B_{11}]\!]}, b, p\right)$
$A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right), A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p\right)$	$\xrightarrow[\kappa]{k_{24}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right), A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p^{b.A}\right)$
$A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right), A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right)$	$\xrightarrow[\kappa]{k_{28}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right), A\left(z_{[\![B_{11}]\!]}, b^{p.A}, p^{b.A}\right)$
$A\left(z_{[\![B_{10}]\!]}, b, p\right) \xrightarrow[\kappa]{k_{16}}{}^{\sharp}\!\!\infty A\left(z_{[\![B_5]\!]}, b, p\right)$		$A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p^{b.A}\right) \xrightarrow[\kappa]{k_{17}}{}^{\sharp}\!\!\infty A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p^{b.A}\right)$
$A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right), A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p^{b.A}\right)$	$\xrightarrow[\kappa]{k_{27}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right), A\left(z_{[\![B_{10}]\!]}, b, p^{b.A}\right)$
$A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right), A\left(z_{[\![B_{11}]\!]}, b^{p.A}, p^{b.A}\right)$	$\xrightarrow[\kappa]{k_{31}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right), A\left(z_{[\![B_{11}]\!]}, b^{p.A}, p\right)$
$A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p\right) \xrightarrow[\kappa]{k_{16}}{}^{\sharp}\!\!\infty A\left(z_{[\![B_5]\!]}, b^{p.A}, p\right)$		$A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right) \xrightarrow[\kappa]{k_{18}}{}^{\sharp}\!\!\infty A\left(z_{[\![B_4]\!]}, b, p^{b.A}\right)$
$A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p\right), A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p^{b.A}\right)$	$\xrightarrow[\kappa]{k_{33}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_{10}]\!]}, b, p\right), A\left(z_{[\![B_{12}]\!]}, b, p^{b.A}\right)$
$A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right), A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p^{b.A}\right)$	$\xrightarrow[\kappa]{k_{32}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_{11}]\!]}, b, p\right), A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p\right)$
$A\left(z_{[\![B_{10}]\!]}, b^{p.A}, p\right), A\left(z_{[\![B_{12}]\!]}, b, p^{b.A}\right)$	$\xrightarrow[\kappa]{k_{33}}{}^{\sharp}\!\!\not\infty$	$A\left(z_{[\![B_{10}]\!]}, b, p\right), A\left(z_{[\![B_{12}]\!]}, b, p\right)$
$A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p\right) \xrightarrow[\kappa]{k_{23}}{}^{\sharp}\!\!\infty A\left(z_{[\![B_7]\!]}, b^{p.A}, p\right)$		$A\left(z_{[\![B_{12}]\!]}, b, p^{b.A}\right) \xrightarrow[\kappa]{k_{24}}{}^{\sharp}\!\!\infty A\left(z_{[\![B_8]\!]}, b, p^{b.A}\right)$
$A\left(z_{[\![B_{12}]\!]}, b, p\right) \xrightarrow[\kappa]{k_{23}}{}^{\sharp}\!\!\infty A\left(z_{[\![B_7]\!]}, b, p\right)$		$A\left(z_{[\![B_{12}]\!]}, b, p\right) \xrightarrow[\kappa]{k_{24}}{}^{\sharp}\!\!\infty A\left(z_{[\![B_8]\!]}, b, p\right)$
$A\left(z_{[\![B_7]\!]}, b, p\right)$	$\xrightarrow[\kappa]{k_{10}}{}^{\sharp}\!\!\infty$	$A\left(z_{[\![B_9]\!]}, b, p\right)$

Table 7.8 – Iterative steps necessary for the generation of the views.

list the elements of the computed set of views and in Table 7.8 we show the steps leading to its generation. At this point the analysis ensures that, applying the concretization function γ to $lfp_V POST_v^{R^S}$, we obtain an over-approximation of the complexes which can form during the possible evolutions of $B2A(B, \xi)$. In this case the over-approximation is infinite. In fact, for example, we can build an arbitrary long chain of $A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p^{b.A}\right)$ which has $A\left(z_{[\![B_{10}]\!]}, b, p\right)$ and $A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right)$ as tips. In the next chapter we present how to exploit the information enclosed in $lfp_V POST_v^{R^S}$.

7.2.1 Improve the Precision of the Analysis

Here we investigate how to improve the precision of the proposed analysis. We observe that at the sixth iteration (see Table 7.8), the computation of $lfp_V POST_v^{R^S}$ applies rule k_{22} to views $A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right)$ and $A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right)$:

$$A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right), A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right) \xrightarrow[\kappa]{k_{22}}{}^{\sharp} \not\bowtie A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right), A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right)$$

This generates the view $A\left(z_{[\![B_6]\!]}, b^{p.A}, p^{b.A}\right)$, which is without counterpart in the actin BlenX model where a box congruent to B_6 cannot be bound on both its interfaces (see Figure 4.1). This imprecision happens because abstract and concrete semantics of the κ-calculus do not take into account rule priorities. If they would, the incriminated application of rule k_{22} could not take place. In fact it is concurrent with the high priority rules k_8 and k_9.

Another way to improve the quality of the analysis is to make more informative the agent views. Currently, the bound state of a view interface stores the name of the interface and of the agent it is bound to. Therefore the first view generated by the abstract transition

$$A\left(z_{[\![B_6]\!]}, b, p\right), A\left(z_{[\![B_6]\!]}, b, p\right) \xrightarrow[\kappa]{k_{22}}{}^{\sharp} \not\bowtie A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right), A\left(z_{[\![B_6]\!]}, b^{p.A}, p\right)$$

forgets the internal state and the bound state of the interfaces of the remote agent involved in the interaction. This implies that the concretization function applied to $lfp_V POST_v^{R^S}$ binds the view $A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right)$ with whatever view having the capability to bind its b interface to the p interface of an agent named A. For example, it generates complexes where the view $A\left(z_{[\![B_6]\!]}, b, p^{b.A}\right)$ occurs bound with $A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p\right)$, which actually cannot be the case. To improve the precision of the analysis we propose to store more information in the views. Considering the previous abstract transitions we would like to have:

$$A\left(z_{[\![B_6]\!]}, b, p\right), A\left(z_{[\![B_6]\!]}, b, p\right) \xrightarrow[\kappa]{k_{22}}{}^{\sharp} \not\bowtie A\left(z_{[\![B_6]\!]}, b, p^{b.A\left(z_{[\![B_6]\!]}, b^-, p\right)}\right), A\left(z_{[\![B_6]\!]}, b^{p.A\left(z_{[\![B_6]\!]}, b, p^-\right)}, p\right)$$

In doing so we prevent the concretization function to join $A\left(z_{[\![B_6]\!]}, b, p^{b.A\left(z_{[\![B_6]\!]}, b^-, p\right)}\right)$ with an agent named A whose z interface has internal state $[\![B_{12}]\!]$. To increase the information carried by views is not a trivial step, in fact it requires to heavily modifying the abstract semantics.

Chapter 8

Optimized simulation algorithm

In this chapter we exploit the encoding of BlenX into κ-calculus to define an optimized simulation algorithm for BlenX. The proposed optimization minimizes the number of visited vanishing states, without modifying the probabilities of visiting a given tangible state.

8.1 Distance One Expressions

In [32] the set of views generated from a mixture is used to generate all the complexes which can appear during its evolution. Here we are interested in extracting different information. We want to generate all the possible subgraphs which can appear during the evolution of the mixture. We refer to this category of expressions as *distance one expressions*.

Definition 8.1.1 (Subgraph) *Given a mixture E the expression $A_1(\sigma_1), \ldots, A_n(\sigma_n)$ is a subgraph of E if its agents are fully specified and if $E \vDash_\kappa A_1(\sigma_1), \ldots, A_n(\sigma_n)$.*

Definition 8.1.2 (Distance one expression with n central views) *A distance one expression with n central views is generated taking n views, which we call* central views of the expression*, and linking their bound interfaces with a compatible view. The binding state of bound interfaces not belonging to central views is replaced with "$-$". We refer a distance one complex with n central views as (E, n) where the central views are the first $n \in \mathbb{N}$ agents occurring in the expression E.*

Notation 8.1.3 *We refer the set of distance one expressions generated from the views of the system S and having n central views as $\mathcal{V}^{S,n}$.*

In order to become familiar with this class of expressions we propose some examples using the views of Table 7.7 . The smallest distance one expression we can generate is made of a central view whose interfaces are free:

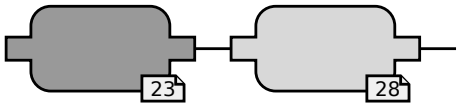
$$\left(A\left(z_{[\![B_1]\!]}, b, p\right), 1\right)$$

If we choose a central view with a bound interface, such as $A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right)$, to obtain a distance one expression we have to bind its p interface with a compatible view. For example, choosing: $A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p^{b.A}\right)$, we get:

$$\left(\left(A\left(z_{[\![B_{11}]\!]}, b, p^1\right), A\left(z_{[\![B_{12}]\!]}, b^1, p^-\right)\right), 1\right)$$

Note that the binding state of the interface p of the compatible view (which is not a central view) is set to "−".

In order to make the representation of distance one expressions more intuitive, we introduce a graphical notation. In this notation, the previous example looks like this:



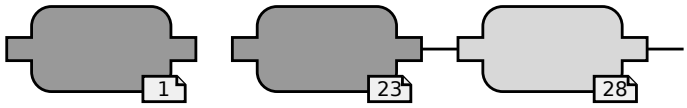
Each view looks like a BlenX box whose label corresponds to the numbering introduced in Table 7.7. Boxes have two interfaces, the one on the left corresponds to the interface b of the view, the other to p; we do not represent the interface z. An interface whose binding state is "−" corresponds to an interface with an exiting arc (as it is the case in the previous example for the p interface of the view labelled with 28). Central views are filled with darker grey.

The next examples show distance one expressions with two central views.

An example of this, using two expressions with one central view is shown below:

$$\left(\left(A\left(z_{[\![B_1]\!]}, b, p\right), A\left(z_{[\![B_{11}]\!]}, b, p^1\right), A\left(z_{[\![B_{12}]\!]}, b^1, p^-\right)\right), 2\right)$$

In graphical notation:



Distance one expression with two central views can also be generated by joining two compatible views, which become the central views, and then binding other compatible views to the remaining bound interfaces. As the central views we choose

$$A\left(z_{[\![B_{11}]\!]}, b, p^{b.A}\right) \text{ and } A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p^{b.A}\right)$$

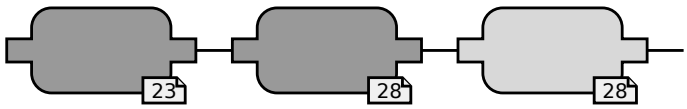
and join them through the interface p of the first and the interface b of the second:

$$A\left(z_{[\![B_{11}]\!]}, b, p^1\right), A\left(z_{[\![B_{12}]\!]}, b^1, p^{b.A}\right)$$

As a next step, we join a compatible view to the interface p of the second central view. Using: $A\left(z_{[\![B_{12}]\!]}, b^{p.A}, p^{b.A}\right)$ we obtain:

$$\left(\left(A\left(z_{[\![B_{11}]\!]}, b, p^1\right), A\left(z_{[\![B_{12}]\!]}, b^1, p^2\right), A\left(z_{[\![B_{12}]\!]}, b^2, p^-\right)\right), 2\right)$$

Or, in graphical notation:



We now show how to apply rules to distance one expressions, which is equivalent to use graph rewriting rules on subgraphs. We only apply to distance one expressions rules which do not have ghost agents (i.e. agent denoted as $\emptyset$ and used to define rules which destroy or create agents) in their rhs and lhs.

Therefore, the application of a rule never changes the number of agents in the expression. Given $((a_1, \ldots, a_n, E), n))$, $((a'_1, \ldots, a'_n, E), n))$ two distance one expressions and $r = E_l \to_p E_r$ a rule, if an expression $E_1 \equiv_\kappa a_1, \ldots, a_n$ exists such that $E_1 \vDash_\kappa E_l$ and $E_1[E_r] \equiv_\kappa a'_1, \ldots, a'_n$ then we write

$$((a_1, \ldots, a_n, E), n) \xrightarrow[\kappa]{r}_p \left(\left(a'_1, \ldots, a'_n, E\right), n\right)$$

Note that the application of a rule to a distance one expression exclusively modifies its central views.

Now, we define two functions which extract information from distance one expressions. The function *borders*, given a distance one expression, returns the set of pairs of agents having the following features: the first agent is not a central view and the second is the central view the first agent is connected to.

Definition 8.1.4 *The pair* $(A_i(\sigma'_i), A_j(\sigma_j))$ *belongs to*

$$borders(((A_1(\sigma_1), \ldots, A_n(\sigma_n), A_{n+1}(\sigma_{n+1}), \ldots, A_m(\sigma_m)), n))$$

if

1. $i \in [1, n]$ *and* $j \in (n, m]$*;*
2. σ_i *and* σ_j *have an interface with the same binding state* $v \in \mathbb{N}$*;*
3. σ'_i *is obtained replacing the binding states of the interfaces of* σ_i *with "–" if they differ from* v *or* ϵ_λ *(i.e. they are free).*

The second function we define is called *central view extractor*.

Definition 8.1.5 *The function central view extractor, henceforth referred to as cve, returns the set of their central views given a set of distance one expressions, replacing the binding state of the interfaces with "–" if it differs from* ϵ_λ*.*

Here we highlight an interesting property of $\mathcal{V}^{S_o,1}$: if an agent appears in a mixture encoding a derivative of S_o, a congruent version of the same agent is the central view of at least one element of $\mathcal{V}^{S_o,1}$.

Proposition 8.1.6 *Given a system* $S = (B, E, \xi) \in ds(S_o)$ *and* $(E', A(\sigma), E'') = B2A(B, \xi)$, $\exists\,((A(\sigma'), E'''), 1) \in \mathcal{V}^{S_o,1}$ *such that* $A(\sigma'') \equiv_\kappa A(\sigma''')$ *where* σ'' *and* σ''' *are obtained replacing the dangling bounds of* σ *and* σ' *with "–", respectively.*

Sketch. It follows by construction of the set $\mathcal{V}^{S,1}$.

□

8.2 Compressed Rules

Compressed rules are made of three distance one expressions with the following characteristics:

1. The second distance one expression results from the application of a low priority rule to the first one;
2. The application of all the possible sequences of high priority rules to the second distance one expression generates an expression congruent to the third distance one expression.

A formal definition follows:

Definition 8.2.1 (Compressed Rule) *A triple* $((E,n),(E',n),(E'',n))$ *is a compressed rule of* S *if:*

1. $E = A_1(y_{\eta_1},\sigma_1),\ldots,A_m(y_{\eta_m},\sigma_m)$ *and* $((A_1(\sigma_1),\ldots,A_m(\sigma_m)),n) \in \mathcal{V}^{S,n}$
2. $\exists r \in R^S$ *s.t.* $(E,n) \xrightarrow[\kappa]{r}_{\not\infty} (E',n)$ *. We refer this rule as* trigger rule*;*
3. *An infinite sequence of transitions starting from* (E',n) *does not exists.*
4. *For each sequence (at least one has to exists)*

$$(E',n) \xrightarrow[\kappa]{r_1}_{\infty} (E_1,n) \xrightarrow[\kappa]{r_2}_{\infty} \ldots \xrightarrow[\kappa]{r_m}_{\infty} ((a_1,\ldots,a_n,E_m),n)$$

such that $\nexists E_l \rightarrow_\infty E_r \in R^S$ *such that* $a_1,\ldots,a_n,E_m \vDash_\kappa E_l$ *and with* $r_1,\ldots,r_m \in R^S$ *the following holds:*

   (a) $a_1,\ldots,a_n,E_m \equiv_\kappa E''$*;*
   (b) $\nexists E_l \rightarrow_\infty E_r \in R_\infty^{S,UNB} \cup R_\infty^{S,BCOM}$ *and* a_1,a_2,a_1',a_2' *s.t.* $(a_1,a_2) \in borders((E_i,n))$ *with* $i \in [1,m)$*,* $a_1',a_2' \equiv_\kappa a_1,a_2$ *and* $a_1',a_2' \vDash_\kappa E_l$*;*
   (c) $\nexists E_l \rightarrow_\infty E_r \in R_\infty^{S,BIND} \cup R_\infty^{S,UCOM}$*,* a_1,a_2,a_1',a_2' *s.t.* $a_1',a_2' \equiv_\kappa a_1,a_2$ *and* $a_1',a_2' \vDash_\kappa E_l$ *and* $a_1 \in cve(\{(E_1,n),\ldots,((a_1,\ldots,a_n,E_m),n)\}), a_2 \in cve(\mathcal{V}^{S,1})$*.*
   (d) $\nexists E_l \rightarrow_\infty E_r \in R_\infty^{S,EV}$ *and* $a \in cve(\{E_1,\ldots,(a_1,\ldots,a_n,E_m)\})$*, s.t.* $a' \equiv_\kappa a$ *and* $a' \vDash_\kappa E_l$*.*

The first point of this definition states that (E,n) results from a distance one expression belonging to $\mathcal{V}^{S,n}$ whose agents have been provided with a special interface with the reserved name y. Each y interface has, as internal state, a different label. Rules of R^S never modify y because they are not aware of its existence. By adding this label we can easily trace the evolution of single agents when rules are applied to (E,n).

The second and third points are self-explanatory. The fourth point verifies that all the applications of sequences of high priority rules to (E',n) lead to the same result (condition 4a). Moreover, it also insures that the sequence of rule applications cannot be interrupted by interactions of the central views with other agents. This check is performed by verifying three conditions: with condition 4b we require that central views of each intermediate distance one expression (E_i,n) do not communicate over link or break links with agents which are part of

(E_i, n) but are not central views. Condition 4c verifies that central views cannot interact through bindings or communications with agents not belonging to (E_i, n) but which could coexist in a system with it (we know that $cve(\mathcal{V}^{S,1})$ over-approximates the set of all the agents not belonging to (E_i, n) because of Proposition 8.1.6). The last condition verifies that central views do not satisfy events with high priority.

We generate the set of compressed-rules of S in an iterative manner. We start with the generation of rules which involves distance one expression belonging to $\mathcal{V}^{S,1}$. Then we proceed with those belonging to $\mathcal{V}^{S,2}$, $\mathcal{V}^{S,3}$ and so on. Given that, in general, this process does not terminate, we have to fix a limit to the number of iterations to perform. We refer the set of compressed rule generated by considering distance one expression belonging to $\bigcup_{0<i\leq n} \mathcal{V}^{S,n}$ as $\mathcal{C}^{S,n}$.

8.3 Properties of Compressed Rules

In this section we explain why we are interested in compressed rules. Suppose we have a compressed rule: $((E'_1, n), (E'_2, n), (E'_3, n))$. Consider the distance one expressions (E_1, n), (E_2, n) and (E_3, n) obtained by removing the special interface y from agents of (E'_1, n), (E'_2, n) and (E'_3, n), respectively. Now we take $(B_1, E, \xi_1) \in ds(S_o)$ such that:

1. If $M \equiv_\kappa B2A(B_1, \xi_1)$ then $\nexists E_l \rightarrow_\infty E_r \in R^{S_o}$ such that $M \vDash_\kappa E_l$;
2. $\exists M_1 = a_{1,1}, \ldots, a_{1,n}, E \equiv_\kappa B2A(B_1, \xi_1)$ such that $M_1 \vDash_\kappa E_1$.

Here, with the first constraint, we require that a rule with high priority cannot be applied to $B2A(B_1, \xi_1)$. The second constraint, given the transitivity of the operator $\vDash_\kappa$, requires that the trigger rule r, which leads to the definition of $((E'_1, n), (E'_2, n), (E'_3, n))$ (condition 2 of Definition 8.2.1), can be applied to $B2A(B_1, \xi_1)$.

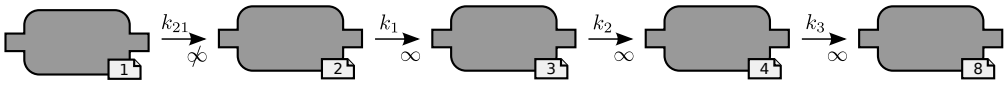
At this point, we know that $M_1 \xrightarrow[\kappa]{r}_\infty M_2$. The application of r to M_1 corresponds to the replacement of the subgraph represented by E_1 with the one represented by E_2. This implies that $\exists M'_2 = a_{2,1}, \ldots, a_{2,n}, E \equiv_\kappa M_2$ such that $M'_2 \vDash_\kappa E_2$. If M'_2 matches E_2 then we can apply to M_2 the sequences of high priority rules which transform (E_2, n) into (E_3, n) (we know that at least one of these sequences exists because of condition 4 of Definition 8.2.1). Moreover, thanks to the constraints introduced with conditions 4b, 4c and 4d of Definition 8.2.1, we know that the application of these rules to M_2 cannot be interrupted by application of other high priority rules. This means that, whatever sequence of high priority rules we apply to M_2, we always obtain a mixture resulting from the replacement of the subgraph represented by E_2 with a subgraph represented by E_3.

We are interested in these properties because they naturally transfer to the system (B_1, E, ξ_1). In fact, if no high priority rules can be applied to M_1, by the completeness result of Theorem 7.1.5, we know that the same holds for (B, E, ξ_1). Moreover, if we verify that (B_1, E, ξ_1) performs an action which corresponds to the application of the trigger rule r to M_1, then we know that it becomes the system (B_2, E, ξ_2) where $B2A(B_2, \xi_2) \equiv_\kappa M_2$. At this point, again by Theorem 7.1.5, we can state that the system (B_2, E, ξ_2) is a vanishing system and it starts a sequence of immediate actions leading to a system (B_3, E, ξ_3) for which it holds that $B2A(B_3, \xi_3) \equiv_\kappa M_3$.

What is particularly interesting is that, by definition of $B2A$, we know that subgraphs of M_2 map to subgraphs of (B_2, E, ξ_2), and the same holds for M_3 and its corresponding system

(B_3, E, ξ_3). This implies that we can obtain a system congruent to (B_3, E, ξ_3) by replacing in (B_2, E, ξ_2) the subgraph represented by E_2 with that represented by E_3. In the next section we formally explain how to identify and replace a distance one expression in a BlenX system.

Here we give some examples of compressed rules generated in the context of the actin case study. Consider the distance one expression $\left(A\left(z_{[\![B_1]\!]}, b, p\right), 1\right)$. It is a good candidate to become the first component of a compressed rule, indeed it matches the lhs of the low priority rule k_{21} of Table 7.6. The application of k_{21} transforms $\left(A\left(z_{[\![B_1]\!]}, b, p\right), 1\right)$ into $\left(A\left(z_{[\![B_2]\!]}, b, p\right), 1\right)$, which becomes the second component of the compressed rule. We can apply to $\left(A\left(z_{[\![B_2]\!]}, b, p\right), 1\right)$ a unique sequence of high priority rules (k_1, k_2, and k_3 of Table 7.5) generating $A\left(z_{[\![B_6]\!]}, b, p\right)$, the third component of the compressed rule. In the following picture we give a graphical representation of the transitions leading to the generation of the compressed rule:



Observe that, as required by conditions 4b, 4c and 4d of Definition 8.2.1, each central view of the vanishing distance one expressions obtained by the application of the rules k_2 and k_3, together with other agents that could potentially coexist, does not match any rule belonging to $R_\infty^{S,\mathrm{BIND}}$, $R_\infty^{S,\mathrm{UNB}}$ and $R_\infty^{S,\mathrm{UCOM}}$. Moreover, these central views do not match any rule of $R_\infty^{S,\mathrm{EV}}$. Here these conditions are trivially satisfied because $R_\infty^{S,\mathrm{BIND}}$, $R_\infty^{S,\mathrm{UNB}}$, $R_\infty^{S,\mathrm{UCOM}}$, and $R_\infty^{S,\mathrm{EV}}$ are empty. The generated compressed rule

$$\left(\left(A\left(y_\eta, z_{[\![B_1]\!]}, b, p\right), 1\right), \left(A\left(y_\eta, z_{[\![B_2]\!]}, b, p\right), 1\right), A\left(y_\eta, z_{[\![B_6]\!]}, b, p\right)\right)$$

carries the following information (note that we added the special interface y): if an inactive free monomer executes the first step of the activation process, it starts a sequence of immediate actions, which cannot be interrupted, and becomes active. This implies that, during the evolution of the system S, if a low priority transition transforms a box belonging to $[\![B_1]\!]$ into a box belonging to $[\![B_2]\!]$, it goes through a sequence of immediate actions which ends with a box congruent to B_6.

A more complex example involves the distance one expression

$$\left(\left(A\left(z_{[\![B_1]\!]}, b, p\right), A\left(z_{[\![B_{11}]\!]}, b, p^1\right), A\left(z_{[\![B_{12}]\!]}, b^1, p^-\right)\right), 2\right)$$

As shown in Figure 8.1, the trigger action cause the binding between the two central views. After the binding, the involved monomers can start two different sequences of immediate transitions which lead to the same distance one expression. These sequences of transitions are responsible of the generation of the following compressed rule:

$$\begin{array}{c}\left(\left(\left(A\left(y_{\eta_1}, z_{[\![B_{10}]\!]}, b^1, p\right), A\left(y_{\eta_2}, z_{[\![B_6]\!]}, b, p\right), A\left(y_{\eta_3}, z_{[\![B_{12}]\!]}, b^-, p^1\right)\right), 2\right),\\ \left(\left(A\left(y_{\eta_1}, z_{[\![B_{10}]\!]}, b^1, p^2\right), A\left(y_{\eta_2}, z_{[\![B_6]\!]}, b^2, p\right), A\left(y_{\eta_3}, z_{[\![B_{12}]\!]}, b^-, p^1\right)\right), 2\right)\\ \left(\left(A\left(y_{\eta_1}, z_{[\![B_{12}]\!]}, b^1, p^2\right), A\left(y_{\eta_2}, z_{[\![B_{10}]\!]}, b^2, p\right), A\left(y_{\eta_3}, z_{[\![B_{12}]\!]}, b^-, p^1\right)\right), 2\right)\end{array}$$

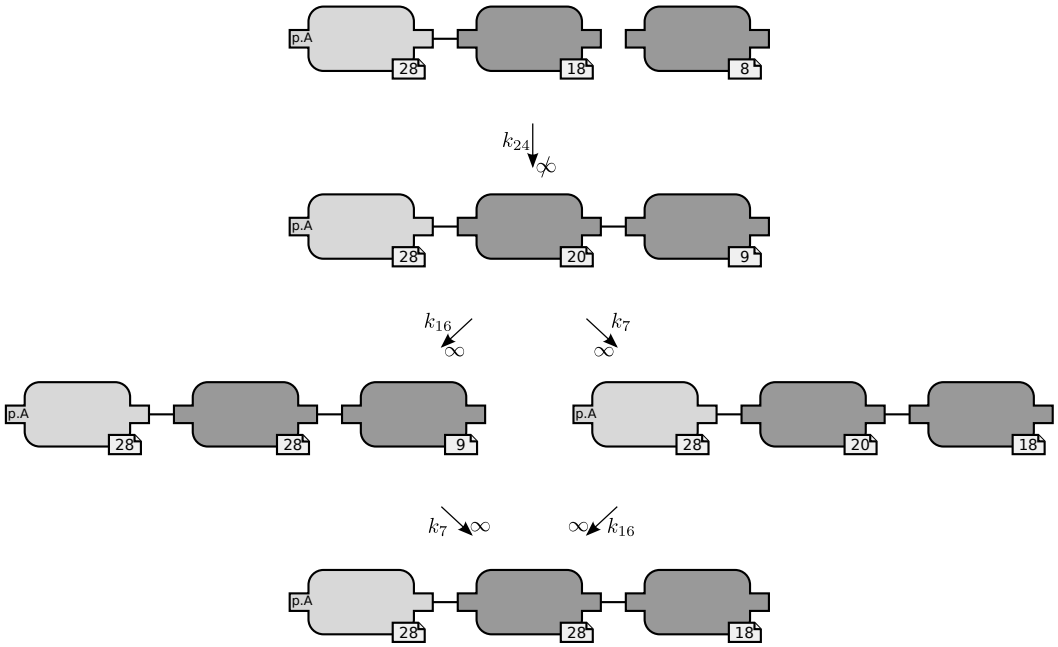


Figure 8.1 – A distance one expression with two central views leading to the generation of a compressed rule. The trigger rule binds the central views.

The first, the second and the third box of the distance one expressions in the picture, correspond to the agents whose interfaces y have internal state η_3, η_1 and η_2, respectively. Observe that the internal state of the interface y allows us to trace how agents transform through the three distance one expressions of the compressed rule.

As a last example, we propose a compressed rule whose trigger rule is k_{33} and causes the unbinding of the central views. The involved distance one expression is

$$\left(\left(A\left(z_{[\![B_{12}]\!]}, b^1, p^2\right), 2\right), \left(A\left(z_{[\![B_{10}]\!]}, b^2, p^1\right), 2\right), \left(A\left(z_{[\![B_{12}]\!]}, b^-, p\right), 2\right)\right)$$

which, through the application of k_{33}, enters the following vanishing configuration:

$$\left(\left(A\left(z_{[\![B_{12}]\!]}, b^1, p\right), 2\right), \left(A\left(z_{[\![B_{10}]\!]}, b, p^1\right), 2\right), \left(A\left(z_{[\![B_{12}]\!]}, b^-, p\right), 2\right)\right)$$

We can apply two independent sequences of rules to this distance one expression: k_{15}, k_5 and k_{19}, k_{10}. The two sequences of actions can be applied concurrently, leading to the same distance one expression. Therefore we have six different sequences of immediate transitions transforming the vanishing distance one expression in the same way (see Figure 8.2):

$$\left(\left(A\left(z_{[\![B_{10}]\!]}, b^1, p\right), 2\right), \left(A\left(z_{[\![B_6]\!]}, b, p^1\right), 2\right), \left(A\left(z_{[\![B_{12}]\!]}, b^-, p\right), 2\right)\right)$$

By properly adding the y interface to the previous distance one expressions, we obtain the compressed rules.

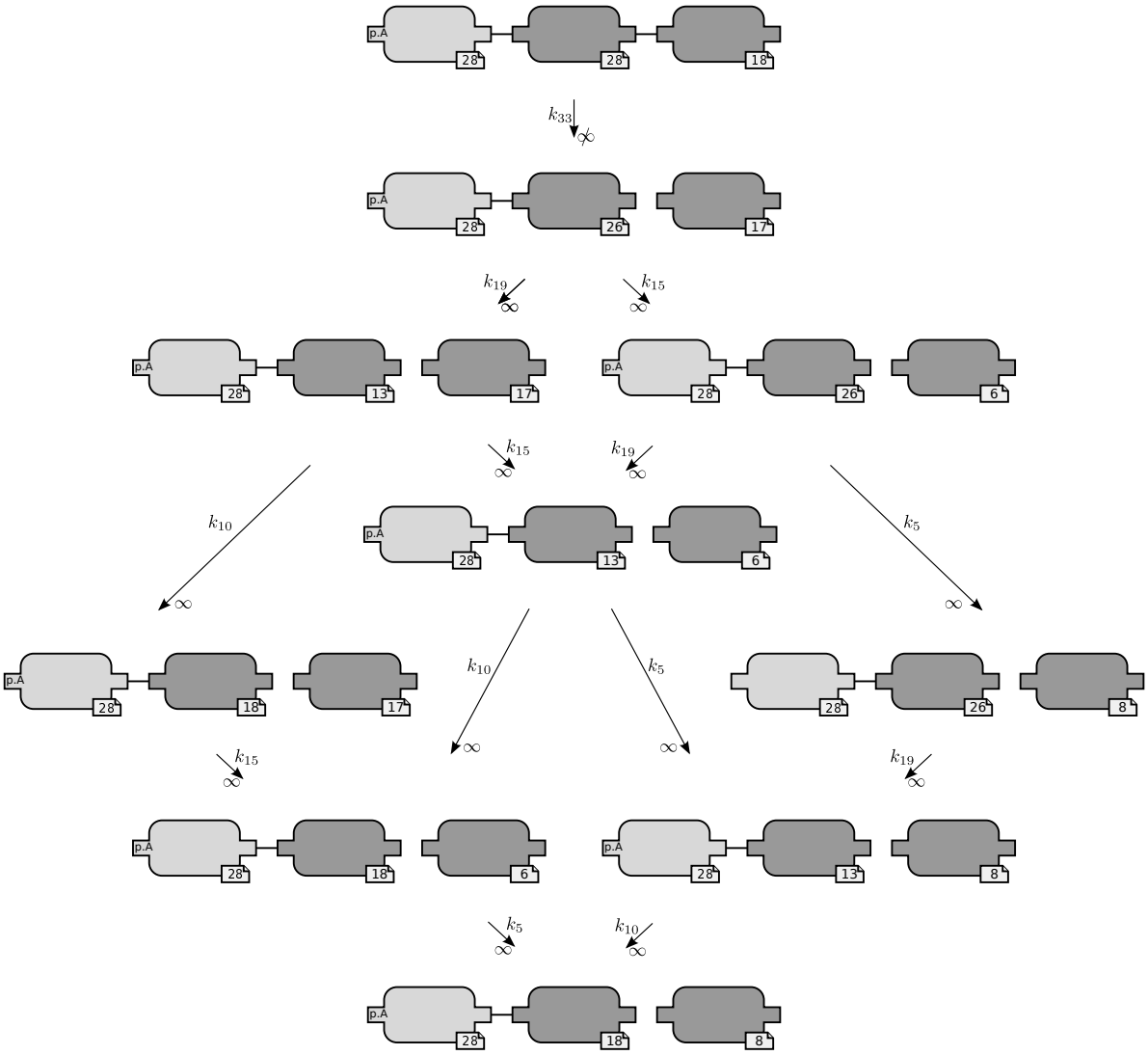


Figure 8.2 – A distance one expression with two central views leading to the generation of a compressed rule. The trigger rule unbinds the central views.

8.4 Using Compressed Rules

Given two systems $S_1 = (B_1, E, \xi_1)$, $S_2 = (B_2, E, \xi_2) \in ds(S)$, the transition $S_1 \rightarrow_r^S S_2$ with $r \in \mathbb{R}$ and a compressed rule $((E_1, n), (E_2, n), (E_3, n))$ we want:

1. To verify if the subgraph represented by E_1 appears in S_1;
2. In case the subgraph represented by E_1 appears in S, to verify if it changes in the subgraph defined by E_2 inside S_2 (i.e. to verify if the trigger rule of the compressed rule has been applied);
3. If the second point holds, to substitute in S_2 the subgraph identified by E_2 with the subgraph represented by E_3.

8.4.1 Identifying Subgraphs in a System

Here we verify if the agents of

$$E_1 = A_1\left(y_{\eta_1}, z_{[\![I_1[\,P_1\,]]\!]}, \dots\right), \dots, A_m\left(y_{\eta_m}, z_{[\![I_m[\,P_m\,]]\!]}, \dots\right)$$

map to a subset of the boxes of S_1. This is true, if:

$$\exists \gamma'_1, \dots, \gamma'_m \text{ distinct labels s.t. } get(\gamma'_i, S_1) = I'_i[\,P'_i\,]_{\gamma'_i} \text{ and } I'_i[\,P'_i\,] \in [\![I_i[\,P_i\,]]\!] \text{ for } i \in [1, m]$$

Then we verify if the identified boxes are linked in the same way as the agents they map. First we check the links which are internal with respect to E_1:

$$\{(\gamma'_i, T_i), (\gamma'_j, T_j)\} \in \xi_1 \text{ iff } A_a\left(y_{\eta_a}, z_{[\![I_a[\,P_a\,]]\!]}, \dots, {s_a}^c, \dots\right)$$
$$\text{with } (\lfloor s_a \rfloor,\, T_a)^{r_a} \in \lfloor I_a \rfloor \text{ for } a \in \{i, j\} \text{ and } c \in \mathbb{N}$$

After that, we check the external links by verifying that box interfaces mapping to bound interfaces of E_1, are bound as well.

$$\exists L \in \xi \text{ s.t. } (\gamma'_i, T_i) \in L \text{ iff } A_i\left(y_{\eta_i}, z_{[\![I_i[\,P_i\,]]\!]}, \dots, s_i^c, \dots\right) \text{ with } (\lfloor s_i \rfloor,\, T_i)^{r_i} \in I_i \text{ and } c \neq \epsilon_\lambda$$

8.4.2 Verifying Trigger Rules Execution

Here we verify that, in the transition $S_1 \to_r S_2$, the only boxes which change their state are those mapping to the subgraph represented by E_1. Moreover, we check that in S_2 they map exactly to the subgraph represented by E_2. Here we keep in mind the bijection identified in the previous step between the labels of the boxes of S_1 and the labels used on the y interfaces of the agents of E_1: we know that the box labelled with γ'_i maps to the agent whose y interface has label η_i for $i \in [1, m]$.
As a first step, we require boxes labelled with $\gamma \notin \{\gamma'_1, \dots, \gamma'_m\}$ to preserve their state:

$$\forall \gamma \notin \{\gamma'_1, \dots, \gamma'_2\}, get(\gamma, S_1) = get(\gamma, S_2)$$

Then, we check if the bijection is still valid among boxes of S_2 and agents of E_2:

$$get(\gamma'_i, S_2) = I'_i[\,P'_i\,]_{\gamma'_i} \text{ iff we have an agent } A_i\left(y_{\eta_i}, z_{[\![I_i[\,P_1\,]]\!]}, \dots\right)$$
$$\text{with } I'_i[\,P'_i\,] \in [\![I_i[\,P_i\,]]\!] \text{ in } (E_2, n)$$

As a last step, we verify that internal links of the subgraph defined by E_2 are coherently mapped by the boxes of S_2, and that the transition does not modify other links. In order to check the internal links, we proceed as previously described:

$$\{(\gamma'_i, T_i), (\gamma'_j, T_j)\} \in \xi_2 \text{ iff } A_a\left(y_{\eta_a}, z_{[\![I_a[\,P_a\,]]\!]}, \dots, s_a^c, \dots\right)$$
$$\text{is an agent of } (E_2, n) \text{ with } (\lfloor s_a \rfloor,\, T_a)^{r_a} \in I_a \text{ for } a \in \{i, j\} \text{ and } c \in \mathbb{N}$$

For the other links, we require:

$$\text{If } \gamma_a \notin \{\gamma'_1, \dots, \gamma'_m\} \text{ then } \{(\gamma_a, T_a), (\gamma_b, T_b)\} \in \xi_2 \text{ iff } \{(\gamma_a, T_a), (\gamma_b, T_b)\} \in \xi_1$$

8.4.3 Replacing Subgraphs

Finally we give the procedure to obtain the system $S_3 = (B_3, E, \xi_3)$ which results from the substitution of the subgraph represented by E_3 into the system S_2. The first step, exploiting the bijection between the labels of box of S_2 and those of E_2, updates the state of the boxes. In order to obtain B_3, for $i \in [1, m]$, we replace in B_2 the box $I'[\,P'\,]_{\gamma'_i}$ with $I[\,P\,]_{\gamma'_i}$ where the agent $A\left(y_{\eta_i}, z_{[\![I[\,P\,]]\!]}, \dots\right)$ occurs in E_3.

In order to update the links we observe that the mutable links map to the internal links of E_2. Therefore, in order to obtain ξ_3, we remove all the links mapping to those of E_2 from ξ_2 and then add those mapping the links of E_3. For the sake of clarity we define two sets, ξ_+ and ξ_-. They represent the links mappings the internal links of E_2 and E_3, respectively.

$$\{(\gamma'_i, T_i), (\gamma'_j, T_j)\} \in \xi_+ \quad \text{iff} \quad \begin{array}{l} A_a\left(y_{\eta_a}, z_{[\![I_a[\,P_a\,]]\!]}, \dots, s_a^c, \dots\right) \text{ is an agent of } E_3 \\ \text{with } (\lfloor s_a \rfloor, T_a)^{r_a} \in \lfloor I_a \rfloor \text{ for } a \in \{i, j\} \text{ and } c \in \mathbb{N} \end{array}$$

The set ξ_- has the same definition of ξ_+, where we replace the occurrences of E_3 with E_2. We can now give the definition of ξ_3:

$$\xi_3 = (\xi_2 \setminus \xi_-) \cup \xi_+$$

8.4.4 An Example

Given the compressed rule generated by the transitions of Figure 8.1

$$\begin{array}{c} \left(\left(\left(E_1 = A\left(y_{\eta_1}, z_{[\![B_{10}]\!]}, b^1, p\right), A\left(y_{\eta_2}, z_{[\![B_6]\!]}, b, p\right), = A\left(y_{\eta_3}, z_{[\![B_{12}]\!]}, b^-, p^1\right)\right), 2\right), \\ \left(\left(E_2 = A\left(y_{\eta_1}, z_{[\![B_{10}]\!]}, b^1, p^2\right), A\left(y_{\eta_2}, z_{[\![B_6]\!]}, b^2, p\right), A\left(y_{\eta_3}, z_{[\![B_{12}]\!]}, b^-, p^1\right)\right), 2\right) \\ \left(\left(E_3 = A\left(y_{\eta_1}, z_{[\![B_{12}]\!]}, b^1, p^2\right), A\left(y_{\eta_2}, z_{[\![B_{10}]\!]}, b^2, p\right), A\left(y_{\eta_3}, z_{[\![B_{12}]\!]}, b^-, p^1\right)\right), 2\right)\right) \end{array}$$

and the systems S_1 and S_2 of Figure 8.3(a) and 8.3(b), we want to perform the following operations:

1. To verify that the subgraph represented by E_1 is present in S_1;
2. To verify that, in the transition from S_1 to S_2, the subgraph identified through E_1 transforms into the one represented by E_2.
3. To substitute in S_2 the subgraph represented by E_2 with the one represented by E_3.

Henceforth we refer to the agents using the internal state of their interface y. The first verification takes place observing that the function

$$\begin{array}{rcl} m(\eta_1) & = & \gamma_3 \\ m(\eta_2) & = & \gamma_4 \\ m(\eta_3) & = & \gamma_2 \end{array}$$

identifies in S_1 the subgraph represented by E_1. In fact it holds that:

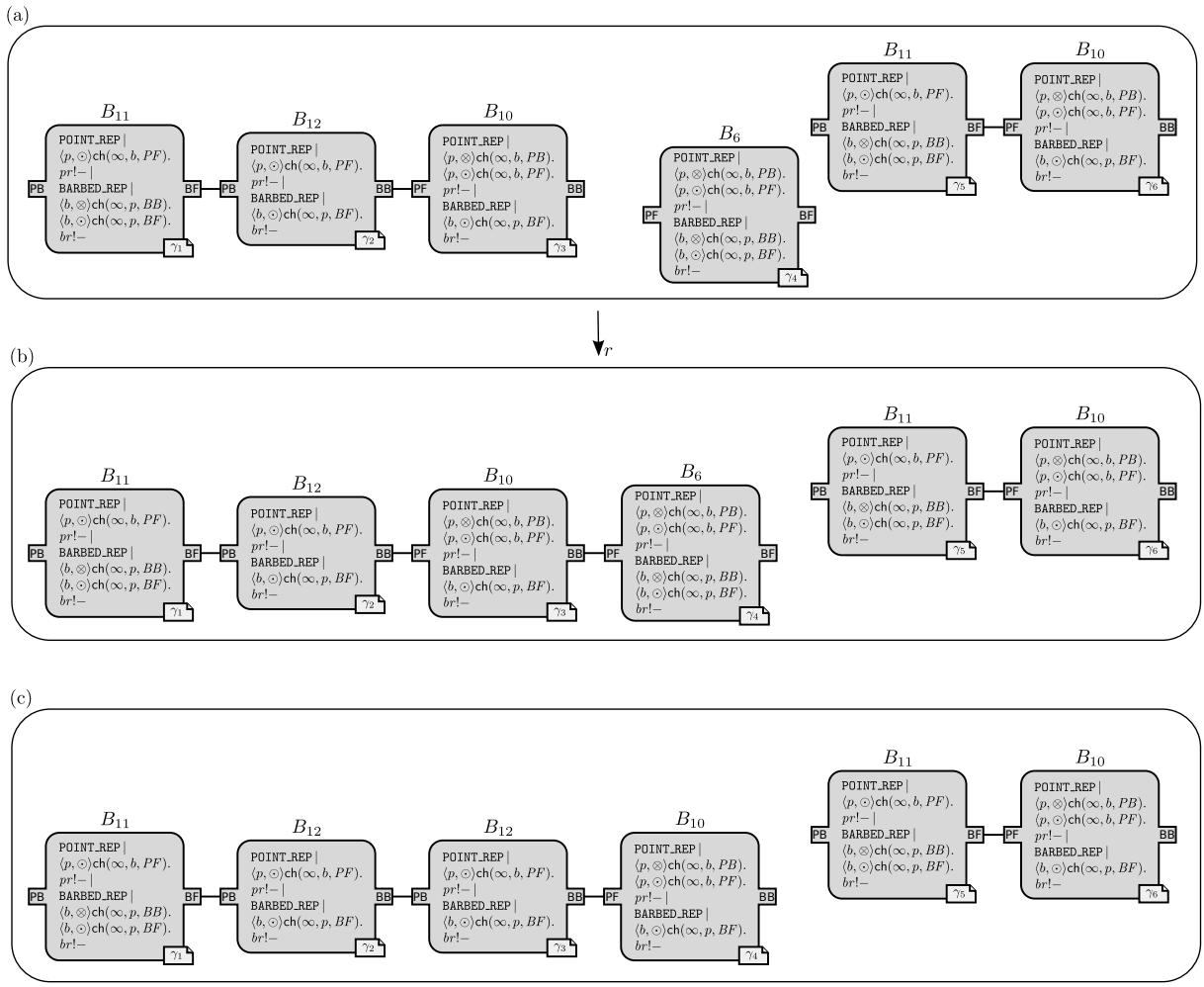


Figure 8.3 – Graphical representation of the systems used in the example. The identifiers used for boxes are those introduced in Figure 4.1. The label on the bottom right of the boxes corresponds to their label in the system.

1. For $i \in [1, 3]$, the box $m(\eta_i)$ belongs to the equivalence class registered in the internal state of the interface z of the agent η_i;

2. The same internal links occur among the boxes γ_1, γ_2 and γ_3 of S_1 and the agents of E_1. This means that the binding between the interface p of box $m(\eta_1) = \gamma_3$ and the interface b of box $m(\eta_2) = \gamma_4$ also occurs between the interface p of the agent η_1 and the interface b of the agent η_2.

3. For $i \in [1, 3]$, if an interface of η_i is involved in an external link (i.e. it is has binding state "−"), the same interface of $m(\eta_i)$ is involved in a link with a box whose label is not in $\{\gamma_2, \gamma_3, \gamma_4\}$. This condition holds because the interface b of the agent η_3 is involved in an external link and the same holds for the interface b of the box $m(\eta_3) = \gamma_2$.

We now verify that in the transition from S_1 to S_2, the subgraph made of the boxes γ_1, γ_2

and γ_3 transforms into the subgraph represented by E_2. This holds if the mapping function m, which identifies in S_1 the subgraph represented by E_1, also identifies in S_2 the subgraph represented by E_2. The verification is performed as in the case of the identification of E_1 in S_1.

As a last step, we substitute in S_2 the subgraph represented by E_2 with the subgraph represented by E_3. Again we exploit the function m. For $i \in [1,3]$, we replace in S_2 the box $m(\eta_i)$ with a box $I_i[\,P_i\,]$ belonging to the equivalent class recorded in the internal state of the interface z of the agent η_i. The agents changing their state from E_1 to E_2 are η_1 and η_2. We therefore update the state of the boxes labelled with $m(\eta_1) = \gamma_3$ and $m(\eta_2) = \gamma_4$. This implies to change the state of the box labelled with γ_3 from B_{10} to B_{12}, and that of the box labelled with γ_4 from B_6 to B_{10}. Note that in place of B_{12} and B_{10} we could have used any of the boxes belonging to the same equivalent classes. This leads to a non-deterministic behaviour of the substitution operation, which in fact does not compute a unique system but a set of systems belonging to the same equivalence class. We defined this operation in such a way because we use it in the context of equivalence classes of systems (see Section 8.5). To complete the substitution, we update the links among the boxes γ_2, γ_3 and γ_4, which requires us to remove the internal links of E_2, and to add those of E_3. Given that E_2 and E_3 have the same links this operation does not have any effect. The resulting system is represented in Figure 8.3(c).

8.5 Optimized Simulation Algorithm

Using the previous sections, we have concluded that, given a compressed rule

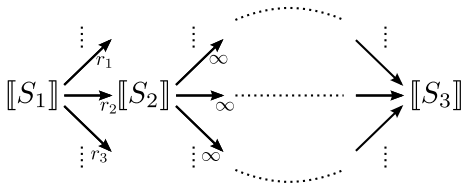
$$((E_1, n), (E_2, n), (E_3, n)) \in \mathcal{C}^{S,n'}$$

we can "apply" it to a system $S_2 \in ds(S)$ if we have a transition $S_1 \rightarrow_r^S S_2$ with this property: E_1 appears in S_1 and, in S_2, it transforms in E_2. Applying the compressed rule to S_2 we obtain S_3, which represents the equivalence class of the systems which S_2, after a sequence of immediate actions, can change to. In other words we know that if:

$$S_1 \rightarrow_r^S S_2 \rightarrow_\infty^S S'_1 \rightarrow_\infty^S \ldots \rightarrow_\infty^S S'_m$$

with S'_m a tangible system, then $S'_m \in [\![S_3]\!]$. Moreover we can compute $[\![S'_m]\!]$ in one shot, without traversing states $S'_1, \ldots, S'_{n-1}$.

If we move our point of view to transition system and congruence classes we have:



This means that the tangible congruence class $[\![S_1]\!]$ can perform an action and transforms in $[\![S_2]\!]$. $[\![S_2]\!]$ is a vanishing state which, through different sequences of immediate actions, inevitably transforms into the tangible congruence class $[\![S_3]\!]$. This subset of the transition system of S, if we do not care about vanishing states, is equivalent to:

$$[\![S_1]\!] \rightarrow_r [\![S_2]\!] \rightarrow_\infty [\![S_3]\!]$$

In fact they have the same underling CTMC: $[\![S_1]\!] \rightarrow_r [\![S_3]\!]$. This equivalence, as highlighted in section 3.3, suggests a path to an optimized algorithm. The underlying intuition is that when the algorithm deals with a system congruent to $[\![S_2]\!]$ it can immediately jump to $[\![S_3]\!]$, without computing the intermediate vanishing configurations. Here, we give a high level description of the original simulation algorithm applied to a system S and then we introduce its optimization.

1. The simulation time is set to zero and the *current configuration* is set to S. We refer the current configuration as S_c;
2. Through operational semantics the active actions of S_c are computed;
3. If S_c is tangible:
   (a) The simulation time and $[\![S_c]\!]$ are stored in the result;
   (b) The rate of the active actions is used to determine the next action and the timestep of the simulation;
   (c) The computed configuration is stored in S_c and the simulation timestep of the fired action is added to the simulation time;
   (d) The procedure restarts from 2;
4. If S_c is vanishing:
   (a) According to the policy of the scheduler one of the active actions is chosen;
   (b) Via operational semantics the selected action is applied to S_c. S_c is updated with the just computed system;
   (c) The procedure restarts from 2.

In order to optimize the algorithm we modify the handling of vanishing states (i.e. the body of the *if* statement at line 4). The mechanism exploits the notion of compressed rule, whose set is computed in the initialization of the system (see line 1). At line 4a, the optimized algorithm checks if the last transition (which we refer as $S_p \rightarrow_r^S S_c$), corresponds to the application of a trigger rule; in other words it looks for the existence of a compressed rule $((E_1, n), (E_2, n), (E_3, n))$ such that the subgraph represented by E_1 appears in S_p and, as consequence of the transition, modifies into a subgraph represented by E_2 inside S_c . If this is the case, the algorithm knows that S_c is going to perform a sequence of immediate actions and, instead of executing them, computes the next tangible configuration. It performs the computation substituting the occurrence of the subgraph represented by E_2 in S_c with the subgraph represented by E_3. Note that this procedure requires us to store two configurations: the current configuration S_c (as in the original algorithm) and the *previous configuration* S_p which stores the configuration whose transformation led to S_c. In order to properly update S_p, the optimized algorithm is enriched with the instructions of points 3b and 4c. At point 4c we set S_p to an invalid system because we triggered an immediate action, and immediate actions cannot trigger compressed rules (trigger rules always have low priority).

1. The simulation time is set to zero, S_c is set to S **and $\mathcal{C}^{S,n'}$ is computed**. We refer the current configuration as S_c;

2. Through operational semantics the active actions of S_c are computed;
3. If S_c is tangible:
   (a) The simulation time and $[\![S_c]\!]$ are stored in the result;
   (b) S_c **is stored in the *previous configuration*. We refer the previous configuration as S_p;**
   (c) The rate of the active actions is used to determine the action to be fired and the timestep of the simulation;
   (d) The computed configuration is stored in S_c and the timestep of the fired action is added to the simulation time;
   (e) The procedure restarts from 2.
4. If S_c is vanishing:
   (a) **if a compressed rule exists $((E_1, n), (E_2, n), (E_3, n)) \in \mathcal{C}^{S,n'}$ such that E_1 represents a subgraph in S_p which, inside S_c, transforms into the one represented by E_2:**
      i. **inside S_c, the graph represented by E_2 is replaced with that represented by E_3. S_c is set to the newly computed system;**
   (b) **otherwise:**
      i. According to the policy of the scheduler one of the active action is chosen;
      ii. Via operational semantics the selected action is applied to S_c. S_c is set to the just computed system;
   (c) S_p **is set to NULL (i.e. an invalid system which cannot satisfy the condition of 4a);**
   (d) The procedure restarts from 2.

Thanks to the properties of compressed-rules, given a system S, the two algorithms visit the same sequences of tangible states and with the same probability. However the second algorithm avoids computing some vanishing states, and therefore computes less steps then the first. We show the effects of the optimization on the system S_1 of Section 8.4.4 which consists of two actin filaments and a free monomer. One of the possible evolution of S_1 is shown in Figure 8.4 where S_1 transforms into the vanishing system S_2 after the binding between the free monomer and the longest filament. This causes the execution of two immediate change actions leading to the update of the interfaces of boxes labelled with γ_3 and γ_4. The current simulation algorithm, in order to jump from S_1 to the system of Figure 8.4(d), computes two vanishing systems. By exploiting our optimization, we can avoid computing one of them. In the case of the optimized algorithm, after the binding between the monomer and the actin filament, the variable S_c is set to S_2 and the variable S_p is set to S_1. Given that S_2 is a vanishing system, the algorithm enters the body of the *if* statement at line 4 where it recognizes that E_1 identifies a subgraph in S_p which, inside S_c, transforms into the one represented by E_2. Therefore the algorithm computes the next system by substituting in S_c the graph E_2 with the E_3. The operations of identification and substitution of subgraphs are performed as described in the example of Section 8.4.4. In conclusion, the algorithm computes the tangible system of Figure 8.4(d) without computing the vanishing system of Figure 8.4(c).

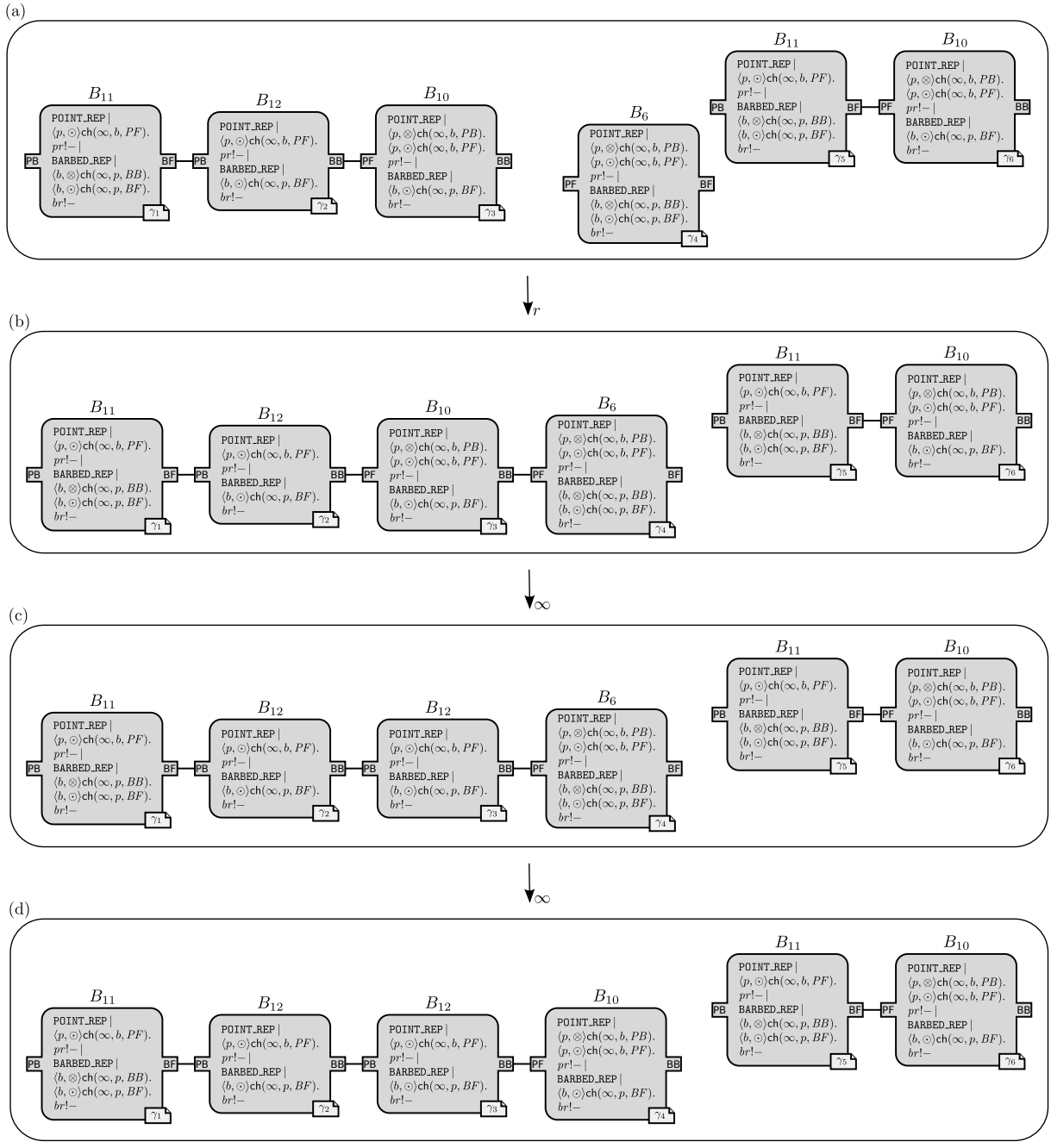


Figure 8.4 – One of the possible evolutions of the system S_1.

Chapter 9

Conclusions

In this thesis we applied static analysis techniques to improve the execution of BlenX models. The optimization technique we describe is based on the partitioning of BlenX systems into two categories: vanishing (which perform immediate actions) and tangible (which do not). By treating vanishing systems as uninformative steps, we are able to speed up the simulation by jumping from tangible configuration to another tangible configuration without computing vanishing ones.

The optimized simulation algorithm is based on a method which, given a BlenX system, generates an over-approximation of the state space of the boxes, i.e., the boxes which can appear during its evolution. In order to make the state space of the boxes computable, we introduced some constraints in the language syntax which ensure the finiteness of the number of configurations assumable by boxes. To reduce the box expressivity, beside being necessary for computational issues, also is in agreement with their biological interpretation. As boxes are intended to model molecules which assume a finite number of interesting conformations, their ability to generate infinite configurations is inappropriate. To generate the state space of the boxes, we defined a control flow analysis for BlenX.

Another fundamental step in the definition of the optimized simulation algorithm is the capability to over-approximate the subgraphs of boxes which can be formed during the evolution of a BlenX system. Our approach is an abstract interpretation based method, inspired from [32], a work developed for the κ-calculus language. In order to exploit the results of [32], we defined an encoding from BlenX to κ-calculus which is based on the generation of the approximated state space of the boxes.

After that, combining the information gathered through abstract interpretation and the encoding, we defined a method which, given a system, generates a list of compressed rules. A compressed rule is a triple of subgraphs (G_1, G_2, G_3) which carries the following information: if during the evolution of a BlenX system the subgraph G_1 appears and transforms in G_2, then we can safely replace G_2 with G_3. This replacement can take place because the analysis ensures that if G_1 transforms into G_2 it starts a sequence of immediate actions (i.e. the system enters a sequence of uninformative vanishing systems) which cannot be interrupted and finally leads to the tangible configuration G_3.

The optimization of the simulation algorithm consists in applying compressed rules whenever possible. In doing so we avoid the computation of some vanishing configurations and thus decrease the number of computed uninformative simulation steps.

9.1 Future work

At the moment we are working on the implementation of the optimized simulation algorithm. We think that it will considerably increase the performance of the current BlenX simulator. Our optimism is supported by the observation that the current simulator, when dealing with realistic models, spends most of its time executing immediate actions.

Beside implementing the simulation algorithm, we are working on improving the proposed analyses. In particular, we are refining the control flow analysis of BlenX. In the literature, we found numerous possible extensions, e.g. it would be interesting to introduce a mechanism for the detection of dead code taking inspiration from [12, 13, 81, 85]. Another point we are working on is the encoding from BlenX to κ-calculus. On this side, we note that the current solution is limited by the absence of immediate actions in the κ-calculus language. To define a new version of the language with such actions would considerably increase the precision of the encoding. This modification should also include an adaptation of the abstract interpretation based method which, at the moment, cannot distinguish between immediate and standard actions.

Another important point we are dealing with is the dimension of the state space of the boxes. Even if we introduced some constraints on the syntax of the language in order to ensure its finiteness, in some cases it can be of intractable size. In practical cases, this can happen when the system involves boxes with several interfaces which can be associated with more than one sort.

Finally, to apply the algorithm, it is also fundamental to define an efficient way to identify subgraphs of boxes inside a BlenX system. This operation is necessary to understand if a compressed rule can be applied. In general the identification of subgraphs is an NP-complete problem. However we are confident that this in not the case for BlenX systems, as the nodes of the graph (i.e. the boxes) have some peculiarities (e.g. each interface is associated with a different sort) which should make it possible to efficiently recognize subgraphs. If this does not turn out to be the case, the optimized simulation algorithm could still be applied, but we should limit the generation of compressed rules to those involving small subgraphs.

Bibliography

[1] http://www.cosbi.eu/downloads/attachment/Models.tar.gz.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular biology of the cell (IV ed.).* Garland science, 2002.

[3] C. Alexopoulos. Statistical analysis of simulation output: state of the art. In *Simulation Conference, 2007 Winter*, pages 150–161. IEEE, 2008.

[4] W.J. Anderson and J.B. Robertson. *Continuous-time Markov chains.* Springer-Verlag New York, 1991.

[5] M. Blinov, J. Yang, J. Faeder, and W. Hlavacek. Graph theory for rule-based modeling of biochemical networks. *Transactions on Computational Systems Biology VII*, pages 89–106, 2006.

[6] M.L. Blinov, J.R. Faeder, B. Goldstein, and W.S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289, 2004.

[7] M.L. Blinov, J.R. Faeder, B. Goldstein, and W.S. Hlavacek. A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. *Biosystems*, 83(2-3):136–151, 2006.

[8] C. Bodei. A Control Flow Analysis for Beta-binders with and without static compartments. *Theoretical Computer Science*, 410(33-34):3110–3127, 2009.

[9] C. Bodei, A. Bracciali, and D. Chiarugi. Control flow analysis for brane calculi. *Electronic Notes in Theoretical Computer Science*, 227:59–75, 2009.

[10] C. Bodei, P. Degano, F. Nielson, and H.R. Nielson. Control Flow Analysis for the π-calculus. volume 1466 of *LNCS*, pages 481–488. Springer, 1998.

[11] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static Analysis for the π-Calculus with Applications to Security. *Information and Computation*, 168(1):68–92, 2001.

[12] C. Bodei, P. Degano, and C. Priami. Checking security policies through an enhanced control flow analysis. *Journal of Computer Security*, 13(1):49–85, 2005.

[13] C. Bodei, P. Degano, C. Priami, and N. Zannone. An enhanced cfa for security policies. In *Proceeding of the workshop on Issues on the Theory of Security (WITS)*. Citeseer, 2003.

[14] N. Busi, M. Gabbrielli, and G. Zavattaro. On the expressive power of recursion, replication and iteration in process calculi. *Mathematical Structures in Computer Science*, 19(06):1191–1222, 2009.

[15] N. Busi and G. Zavattaro. On the expressive power of movement and restriction in pure mobile ambients* 1. *Theoretical Computer Science*, 322(3):477–515, 2004.

[16] L. Cardelli. Brane calculi. In *Computational methods in systems biology*, pages 257–278. Springer, 2005.

[17] L. Cardelli, E. Caron, P. Gardner, O. Kahramanoğulları, and A. Phillips. A Process Model of Actin Polymerisation. *Electronic Notes in Theoretical Computer Science*, 229(1):127–144, 2009.

[18] L. Cardelli and A. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures*, pages 140–155. Springer, 1998.

[19] A. E. Carlsson, M. A. Wear, and J. A. Cooper. End versus side branching by Arp2/3 complex. *Biophysical journal*, 86(2):1074–1081, 2004.

[20] D. Chiarugi, M. Curti, P. Degano, and R. Marangoni. Vice: A virtual cell. In *Computational Methods in Systems Biology*, pages 207–220. Springer, 2005.

[21] F. Ciocchetta, A. Duguid, S. Gilmore, M.L. Guerriero, and J. Hillston. The Bio-PEPA tool suite. In *Proceedings of the 2009 Sixth International Conference on the Quantitative Evaluation of Systems*, pages 309–310. IEEE Computer Society, 2009.

[22] F. Ciocchetta, S. Gilmore, M.L. Guerriero, and J. Hillston. Integrated simulation and model-checking for the analysis of biochemical systems. *Electronic Notes in Theoretical Computer Science*, 232:17–38, 2009.

[23] F. Ciocchetta and J. Hillston. Bio-PEPA: an extension of the process algebra PEPA for biochemical networks. *Electronic Notes in Theoretical Computer Science*, 194(3):103–117, 2008.

[24] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the second International Symposium on Programming*, pages 106–130. Paris, 1976.

[25] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[26] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, page 282. ACM, 1979.

[27] F.H.C. Crick. The biological replication of macromolecules. In *Symp. Soc. Exp. Biol*, volume 12, pages 138–163, 1958.

[28] P. Curien, V. Danos, J. Krivine, and M. Zhang. Computational self-assembly. *Theor. Comput. Sci.*, 404(1-2):61–75, 2008.

[29] Feret J. Fontana W. Harmet R Danos, V. and J. Krivine. Abstracting the differential semantics of rule-based models: exact and automated model reduction. In *Logic in Computer Science*, to appear 2010.

[30] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-based modelling of cellular signalling. In *CONCUR*, volume 4703 of *LNCS*, pages 17–41, 2007.

[31] V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable modelling of biological pathways. In *Proceedings of APLAS*, volume 4807 of *LNCS*, pages 139–157, 2007.

[32] V. Danos, J. Feret, W. Fontana, and J. Krivine. Abstract interpretation of cellular signalling networks. In *Verification, Model Checking, and Abstract Interpretation*, volume 4905 of *LNCS*, pages 83–97. Springer, 2008.

[33] V. Danos and C. Laneve. Formal molecular biology. *Theoretical Computer Science*, 325(1):69–110, 2004.

[34] V. Danos and S. Pradalier. Projective brane calculus. In *Computational Methods in Systems Biology*, volume 3082 of *LCNS*, pages 134–148. Springer, 2005.

[35] M. David, J. Bantang, and E. Mendoza. A Projective Brane Calculus with Activate, Bud and Mate as Primitive Actions. *Transactions on Computational Systems Biology XI*, pages 164–186, 2009.

[36] 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.

[37] E. K. Drexler. *Nanosystems: molecular machinery, manufacturing, and computation*. Wiley, 1992.

[38] J.R. Faeder, M.L. Blinov, B. Goldstein, and W.S. Hlavacek. Rule-based modeling of biochemical networks. *Complexity*, 10(4):22–41, 2005.

[39] J.R. Faeder, M.L. Blinov, and W.S. Hlavacek. Graphical rule-based representation of signal-transduction networks. In *Proceedings of the 2005 ACM symposium on Applied computing*, page 140. ACM, 2005.

[40] F. Fages, S. Soliman, and N. Chabrier-Rivier. Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. *Journal of Biological Physics and Chemistry*, 4:64–73, 2004.

[41] J. Fass, C. Pak, J. Bamburg, and A. Mogilner. Stochastic simulation of actin dynamics reveals the role of annealing and fragmentation. *Journal of theoretical biology*, 252(1):173–183, 2008.

[42] J. Feret. Dependency analysis of mobile systems. *Programming Languages and Systems*, pages 223–267, 2002.

[43] J. Fisher and T.A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–1249, 2007.

[44] D.T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of computational physics*, 22(4):403–434, 1976.

[45] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[46] D. Harel and A. Pneuli. *On the development of reactive systems.* Microelectronics and Computer, Technology Corporation, 1985.

[47] J. Hasty, D. McMillen, and J. J. Collins. Engineered gene circuits. *Nature*, 420(6912):224–230, 2002.

[48] J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257, 2008.

[49] A.V. Hill et al. The possible effects of the aggregation of the molecules of haemoglobin on its dissociation curves. *J Physiol*, 40(4), 1910.

[50] J. Hillston. *A compositional approach to performance modelling.* Cambridge Univ Pr, 1996.

[51] W.S. Hlavacek, J.R. Faeder, M.L. Blinov, R.G. Posner, M. Hucka, and W. Fontana. Rules for modeling signal-transduction systems. *Science's STKE*, 2006(344), 2006.

[52] S. Hoops, S. Sahle, R. Gauges, C. Lee, J. Pahle, N. Simus, M. Singhal, L. Xu, P. Mendes, and U. Kummer. COPASI–a COmplex PAthway SImulator. *Bioinformatics*, 22(24):3067, 2006.

[53] T. Ideker, T. Galitski, and L. Hood. A New APPROACH TO DECODING LIFE. *Annu. Rev. Genomics Hum. Genet*, 2:343–72, 2001.

[54] M. John, C. Lhoussaine, J. Niehren, and A. Uhrmacher. The attributed pi calculus. In *Computational Methods in Systems Biology*, pages 83–102. Springer, 2008.

[55] R.N. Jorissen, F. Walker, N. Pouliot, T.P.J. Garrett, C.W. Ward, and A.W. Burgess. Epidermal growth factor receptor: mechanisms of activation and signalling. *Experimental cell research*, 284(1):31–53, 2003.

[56] D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte. *Modelling with generalized stochastic Petri nets.* John Wiley & Sons, Inc. New York, NY, USA, 1994.

[57] E.N. Kersh, A.S. Shaw, and P.M. Allen. Fidelity of T Cell Activation Through Multistep T Cell Receptor ζ Phosphorylation. *Science*, 281(5376):572, 1998.

[58] B.N. Kholodenko, O.V. Demin, G. Moehren, and J.B. Hoek. Quantification of short term signaling by the epidermal growth factor receptor. *Journal of Biological Chemistry*, 274(42):30169, 1999.

[59] G.A. Kildall. Global expression optimization during compilation. 1972.

[60] M.W. Kirschner. The meaning of systems biology. *Cell*, 121(4):503, 2005.

[61] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662, 2002.

[62] E. Klavins. Automatic synthesis of controllers for assembly and formation forming. In *International Conference on Robotics and Automation*, 2002.

[63] B. Knaster. Un théoreme sur les fonctions densembles. *Ann. Soc. Polon. Math*, 6(133):2013134, 1928.

[64] C. Kuttler, C. Lhoussaine, and M. Nebut. Rule-based modeling of transcriptional attenuation at the tryptophan operon. In *Formal Methods in Molecular Biology*, 2009.

[65] C. Kuttler and J. Niehren. Gene regulation in the pi calculus: Simulating cooperativity at the lambda switch. *Transactions on computational systems biology VII*, pages 24–55, 2006.

[66] R. Larcher, C. Priami, and A. Romanel. Modelling self-assembly in BlenX. *Transactions on Computational Systems Biology XII*, pages 163–198, 2010.

[67] N. Le Novere and T.S. Shimizu. STOCHSIM: modelling of stochastic biomolecular processes. *Bioinformatics*, 17(6):575, 2001.

[68] 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, 2004.

[69] H. Li, Y. Cao, and L. Petzold. StochKit: A stochastic simulation toolkit. 2008.

[70] H.F. Lodish. *Molecular cell biology*. WH Freeman, 2003.

[71] L. Lok and R. Brent. Automatic generation of cellular reaction networks with Moleculizer 1.0. *Nature biotechnology*, 23(1):131–136, 2005.

[72] S. Maffeis and I. Phillips. On the computational strength of pure ambient calculi. *Theoretical Computer Science*, 330(3):501–551, 2005.

[73] A. Matzavinos and H.G. Othmer. A stochastic analysis of actin polymerization in the presence of twinfilin and gelsolin. *Journal of theoretical biology*, 249(4):723–736, 2007.

[74] L. Michaelis and M. Menten. The kinetics of invertase activity. *Biochemische Zeitschrift*, 49(333):166–180, 1913.

[75] R. Milner. *Communication and concurrency*. 1989.

[76] R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, 1999.

[77] A. Mogilner and L. Edelstein-Keshet. Regulation of actin dynamics in rapidly moving cells: a quantitative analysis. *Biophysical journal*, 83(3):1237–1258, 2002.

[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] S. Nanz, F. Nielson, and H. Nielson. Topology-dependent abstractions of broadcast networks. In *Concurrency Theory (CONCUR'07)*, pages 226–240.

[80] F. Nielson, R.R. Hansen, and H.R. Nielson. Abstract interpretation of mobile ambients. *Science of Computer Programming*, 47(2-3):145–175, 2003.

[81] F. Nielson, H. Riis Nielson, and R.R. Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(2):381–418, 2002.

[82] F. Nielson, H. Riis Nielson, C. Priami, and D. Rosa. Control Flow Analysis for BioAmbients. *Electronic Notes in Theoretical Computer Science*, 180(3):65–79, 2007.

[83] H. Nielson and F. Nielson. Data flow analysis for CCS. *Program analysis and compilation, theory and practice*, pages 311–327, 2007.

[84] H.R. Nielson and F. Nielson. A flow-sensitive analysis of privacy properties. In *20th IEEE Computer Security Foundations Symposium, 2007. CSF'07*, pages 249–264, 2007.

[85] H.R. Nielson, F. Nielson, and H. Pilegaard. Spatial analysis of BioAmbients. In *Static Analysis Symposium*, volume 3148 of *LNCS*, pages 1–23. Springer, 2004.

[86] D. Noble. The rise of computational biology. *Nature Reviews Molecular Cell Biology*, 3(6):459–463, 2002.

[87] J.R. Norris. *Markov chains*. Cambridge Univ Pr, 1998.

[88] D. Pantaloni, R. Boujemaa, D. Didry, P. Gounon, and M. F. Carlier. The Arp2/3 complex branches filament barbed ends: functional antagonism with capping proteins. *Nature cell biology*, 2(7):385–391, 2000.

[89] T. Pawson. Specificity in Signal Transduction:: From Phosphotyrosine-SH2 Domain Interactions to Complex Cellular Systems. *Cell*, 116(2):191–203, 2004.

[90] A. Phillips and L. Cardelli. Efficient, Correct Simulation of Biological Processes in the Stochastic Pi-calculus. In *CMSB 2007*, volume 4695 of *LNCS*, page 184. Springer, 2007.

[91] H. Pilegaard. *Language based techniques for systems biology*. PhD thesis.

[92] G.D. Plotkin. A structural approach to operational semantics, 1981.

[93] D. Prandi. *A Formal Study of Biological Interactions*. PhD thesis.

[94] D. Prandi, C. Priami, and P. Quaglia. Communicating by compatibility. *JLAP*, 75:167, 2008.

[95] C. Priami. Stochastic π-calculus. *The Computer Journal*, 38(6):578–589, 1995.

[96] C. Priami and P. Quaglia. Beta binders for biological interactions. In *CMSB*, volume 3082 of *LNCS*, page 20. Springer, 2004.

[97] C. Priami, P. Quaglia, and A. Romanel. BlenX Static and Dynamic Semantics. In *CONCUR*, volume 5710 of *LNCS*, pages 37–52. Springer, 2009.

[98] C. Priami, A. Regev, E.Y. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Inf. Process. Lett.*, 80(1):25–31, 2001.

[99] S. Ramsey, D. Orrell, and H. Bolouri. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *Journal of bioinformatics and computational biology*, 3(2):415–436, 2005.

[100] A. Regev, E.M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science*, 325(1):141–167, 2004.

[101] A. Regev and E. Shapiro. Cells as computations. *Nature*, 419:343, 2002.

[102] A. Regev, W. Silverman, and E. Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In *Pacific symposium on biocomputing*, volume 6, pages 459–470, 2001.

[103] H.G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[104] A. Romanel. *Dynamic Biological Modelling a language-based approach*. PhD thesis, 2010.

[105] A. Romanel and C. Priami. On the decidability and complexity of the structural congruence for beta-binders. *Theor. Comput. Sci.*, 404(1-2):156–169, 2008.

[106] A. Romanel and C. Priami. On the computational power of BlenX. *Theoretical Computer Science*, 411(2):542–565, 2010.

[107] D. Sangiorgi and D. Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge Univ Pr, 2003.

[108] H. Schmidt. Systems Biology Toolbox for MATLAB, 2006.

[109] W.X. Schulze, L. Deng, and M. Mann. Phosphotyrosine interactome of the ErbB-receptor kinase family. *Molecular systems biology*, 1(1), 2005.

[110] O. Shivers. Control flow analysis in scheme. *ACM SIGPLAN Notices*, 23(7):174, 1988.

[111] M.H.V. Van Regenmortel. Reductionism and complexity in molecular biology. *EMBO reports*, 5(11):1016–1020, 2004.

[112] C. Versari. A core calculus for a comparative analysis of bio-inspired calculi. *Programming Languages and Systems*, pages 411–425, 2007.

[113] C. Versari and R. Gorrieri. π@: a π-based process calculus for the implementation of compartmentalised bio-inspired calculi. In *SFM*, volume 5016 of *LNCS*, pages 449–506. Springer, 2008.

[114] H.V. Westerhoff and B.O. Palsson. The evolution of molecular biology into systems biology. *Nature biotechnology*, 22(10):1249–1252, 2004.

[115] P. Yin, H. M. T. Choi, C. R. Calvert, and N. A. Pierce. Programming biomolecular self-assembly pathways. *Nature*, 451(318-322), 2008.