Università degli Studi di Pisa
Facoltà di Scienze Matematiche Fisiche e Naturali

Corso di Laurea Specialistica in

Tecnologie Informatiche

A Calculus for Molecular Interaction Maps

Tesi di Laurea di

Aureliano Rama

Relatori:

Roberto Barbuti
Giovanni Pardini

Anno Accademico 2007-2008

Abstract

Molecular Interaction Maps are a graphical formalism used by biologists to describe complex interactions between molecules. We provide a formal description of MIMs using process algebras and determine its computational power.

Contents

List of Figures

dedicato a **Gabriella**
che mi è stata accanto e mi ha voluto bene
quando altri hanno abbandonato;

a **Pierluigi** e **Maurizio**
che non hanno mai mollato
anche quando avrebbero avuto tutte le ragioni per farlo;

a **Damiano**
che mi fa avanzare e crescere, volente o nolente;

e infine a tutti coloro che, magari a modo loro,
mi hanno aiutato e sostenuto
in questi anni lunghi e a volte frustranti.

Introduction

Molecular Interaction Maps are a graphical formalism used by biologists to describe complex interactions between molecules. They allow biologists and chemists to model large, complex systems with tens of molecules interacting with one another in many different ways, creating products, moving molecules between cells and membranes, synthetizing proteins and RNA fragments.

The motivation behind the creation of the MIM graphic formalism was mainly to have a tool to easily depict and explain those complex systems to colleagues and students.

MIMs, however, are a powerful tool that can be used, modifying its syntax, to describe complex system of any kind. In order for this to be really useful, we provide a formal description of MIMs using process algebras and we determine its computational power.

In chapter 1, we describe the prerequisites to our work. The Structural Operational Semantics is a generalization of Operational Semantics, which can be applied to programming or specification languages to obtain a description of the language component and their interactions through formal rules. Process Algebras represent a natural modeling for concurrent systems to whom they give very strong mathematical foundation. We devolve some space to describe a couple of interesting PA properties (like bisimilarity) that have a direct interest in our work.

In chapter 2, Molecular Interaction Maps are described in details. First we talk about the modeling needs they helped to solve, then we sketch the graphic formalism and finally we point out some of the intricacies that an ambiguous (but powerful) design poses to any attempt in formally describing them or in computer simulating their evolution.

In chapter 3, we give formal syntax and semantics to MIMs, keeping the original meaning where possible and choosing the next best approximation when not.

In chapter 4, we describe our result on computational power of MIM (and hence of our formalism) using Petri Nets as a well known modeling tool whose computational power has been studied for long time.

In chapter 5, finally, we sketch a very simple example on how a real life MIM is translated in our calculus.

Chapter 1

Prerequisites

These introductory section will sketch some of the prerequisites upon which out work is based. Structural Operational Semantics is a powerful tool to describe and give meaning to a formal language. Process Algebras allow us to have strong mathematical foundations for our work.

1.1 Structural Operational Semantics (SOS)

Structural operational semantics (SOS) provides a framework to give an operational semantics to programming and specification languages. In particular, because of its intuitive appeal and flexibility, SOS has found considerable application in the study of the semantics of concurrent processes, where the methods of denotational semantics appear to be difficult to apply in general.

SOS was introduced by Gordon Plotkin in [29] as a logical means to defining operational semantics. The basic idea behind SOS is to define the behavior of a program in terms of the behavior of its parts, thus providing a structural, i.e., syntax oriented and inductive, view on operational semantics. An SOS specification defines the behavior of a program in terms of a (set of) transition relation(s). SOS specifications take the form of a set of inference rules which define the valid transitions of a composite piece of syntax in terms of the transitions of its components.

In particular, SOS generates a labelled transition system, whose states are the closed terms over an algebraic signature, and whose transitions between states are obtained inductively from a collection of so-called transition rules of the form $\frac{\text{premises}}{\text{conclusion}}$. A typical example of a transition rule is:

$$\frac{x \xrightarrow{a} x'}{x \| y \xrightarrow{a} x' \| y}$$

stipulating that if $t \stackrel{a}{\rightarrow} t'$ holds for certain closed terms t and t', then so does $t \| u \stackrel{a}{\rightarrow} t' \| u$ for each closed term u. In general, validity of the premises of a transition rule, under a certain substitution, implies validity of the conclusion of this rule under the same substitution.

Clearly systems have some behavior and it is that which we wish to describe. In an operational semantics one focuses on the operations the system can perform whether internally or interactively with some supersystem or the outside world. For in our discrete (digital) computer systems behavior consists of elementary steps which are occurrences of operations. Such elementary steps are called here (and also in many other situations in Computer Science) transitions (= moves). Thus a transition steps from one configuration to another and as a first idea we take it to be a binary relation between configurations.

Definition (Transition System)**.** A Transition System (ts) is a structure $<\Gamma, \longrightarrow>$ where Γ is a set (of elements γ called configurations) and $\longrightarrow \subseteq \Gamma \times \Gamma$ is a binary relation (called the transition relation). Read $\gamma \longrightarrow \gamma'$ as saying that there is a transition from the configuration γ to the configuration γ'. (Other notations sometimes seen are $\vdash$,$\Rightarrow$ and $\rhd$).

Definition (Terminal Transition System)**.** A Terminal Transition System (tts) is a structure $<\Gamma, \longrightarrow, T>$ where $<\Gamma, \longrightarrow>$ is a ts, and $T \subseteq \Gamma$ (the set of final configurations) satisfy $\forall \gamma \in T, \forall \gamma' \in \Gamma \,.\, \gamma \not\longrightarrow \gamma'$

A point to watch is to make a distinction between internal and external behavior. Internally a system's behavior is nothing but the sum of its transitions. (We ignore here the fact that often these transitions make sense only at a certain level; what counts as one transition for one purpose may in fact consist of many steps when viewed in more detail. Part of the spirit of our method is to choose steps of the appropriate "size".) However externally many of the transitions produce no detectable effect. It is a matter of experience to choose the right definition of external behavior. Often two or more definitions of behavior (or of having the same behavior) are possible for a given transition system. Indeed on occasion one must turn the problem around and look for a transition system which makes it possible to obtain an expected notion of behavior.

Transition systems in general do not give the opportunity of saying very much about any individual transition. By adding the possibility of such information we arrive at a definition.

Definition (Labelled Transition System)**.** A Labelled Transition System (lts) is a structure $< \Gamma, A, \longrightarrow>$ where Γ is a set of configurations and A is a

set of actions (or labels or operations) and $\longrightarrow \subseteq \Gamma \times A \times \Gamma$ is the transition relation.

We write a transition as $\gamma \xrightarrow{a} \gamma'$ where γ, γ' are configurations and a is an action. The idea is that an action can give information about what went on in the configuration during the transition (internal actions) or about the interaction between the system and its environment (external actions) (or both). The labels are particularly useful for specifying distributed systems where the actions may relate to the communications between sub-systems. The idea of Labelled Terminal Transition Systems $< \Gamma, A, \longrightarrow, T >$ is pretty obvious.

Definition (Reflexive Transitive Closure)**.** For any lts, let γ and γ' be configurations and take $x = a_1 \ldots a_k \in A^*$ then:

$$\gamma \xrightarrow{x}{}^{*} \gamma' \quad \stackrel{\text{def}}{=} \quad \exists \gamma_1, \ldots, \gamma_k \cdot \gamma_1 \xrightarrow{a_1} \gamma_2 \ \ldots \ \xrightarrow{a_k} \gamma_k = \gamma'$$

where $k \geq 0$.

With the reflexive transitive closure, we have a way to simulate a complete execution of a system.

Recently, SOS has been successfully applied as a formal tool to establish results that hold for classes of process description languages. This has allowed for the generalization of well-known results in the field of process algebra, and for the development of a meta-theory for process calculi based on the realization that many of the existent results in this field only depend upon general semantic properties of language constructs.

1.2 Process Algebras

The term process algebra is used in different meanings [6]. First of all, consider the word "process". It refers to behavior of a system. A system is anything showing behavior, in particular the execution of a software system, the actions of a machine or even the actions of a human being. Behavior is the total of events or actions that a system can perform, the order in which they can be executed and maybe other aspects of this execution such as timing or probabilities. Always, we describe certain aspects of behavior, disregarding other aspects, so we are considering an abstraction or idealization of the "real" behavior. Rather, we can say that we have an observation of behavior, and an action is the chosen unit of observation. Usually, the actions are

thought to be discrete: occurrence is at some moment in time, and different actions are separated in time. This is why a process is sometimes also called a discrete event system.

The word "algebra" denotes that we take an algebraic/axiomatic approach in talking about behavior. That is, we use the methods and techniques of universal algebra.

The simplest model of behavior is to see behavior as an input/output function. A value or input is given at the beginning of the process, and at some moment there is a value as outcome or output.

When dealing with interacting systems, we say we are doing concurrency theory, so concurrency theory is the theory of interacting, parallel and/or distributed systems. When talking about process algebra, we usually consider it as an approach to concurrency theory, so a process algebra will usually (but not necessarily) have parallel composition as a basic operator.

Thus, we can say that process algebra is the study of the behavior of parallel or distributed systems by algebraic means. It offers means to describe or specify such systems, and thus it has means to talk about parallel composition. Besides this, it can usually also talk about alternative composition (choice) and sequential composition (sequencing). Moreover, we can reason about such systems using algebra, i.e. equational reasoning. By means of this equational reasoning, we can do verification, i.e. we can establish that a system satisfies a certain property.

What are these basic laws of process algebra? We can list some, that are usually called structural or static laws. We start out from a given set of atomic actions, and use the basic operators to compose these into more complicated processes. As basic operators, we use + denoting alternative composition, ; denoting sequential composition and — denoting parallel composition. Usually, there are also neutral elements for some or all of these operators, but we do not consider these here. Some basic laws are the following (+ binding weakest, ; binding strongest).

— x + y = y + x (commutativity of alternative composition)

— x + (y + z) = (x + y) + z (associativity of alternative composition)

— x + x = x (idempotency of alternative composition)

— (x + y); z = x; z + y; z (right distributivity of + over ;)

— (x; y); z = x; (y; z) (associativity of sequential composition)

— x — y = y — x (commutativity of parallel composition)

— (x — y) — z = x — (y — z) (associativity of parallel composition)

So, we can say that any mathematical structure with three binary operations satisfying these 7 laws is a process algebra.

The notion of equivalence studied is usually not language equivalence. Prominent among the equivalences studied is the notion of bisimulation. A bisimulation is a binary relation between state transition systems, associating systems which behave in the same way in the sense that one system simulates the other and vice-versa. Intuitively two systems are bisimilar if they match each other's moves. In this sense, each of the systems cannot be distinguished from the other by an observer.

Formally, given a labelled state transition system $(S, \lambda, \rightarrow)$, a bisimulation relation is a binary relation R over S (i.e., $R \subseteq S \times S$) such that both R^{-1} and R are simulations. Or equivalently R is a bisimulation if for every pair of elements $p, q \in S$ with $(p, q) \in R$, $\forall \alpha \in A$:

$$\forall p' \in S \ : \ p \xrightarrow{\alpha} p' \Rightarrow \exists q' \in S \ : \ q \xrightarrow{\alpha} q'$$

and symmetrically,

$$\forall q' \in S \ : \ q \xrightarrow{\alpha} q' \Rightarrow \exists p' \in S \ : \ p \xrightarrow{\alpha} p'$$

Given two states $p, q \in S$, p is bisimilar to q, written $p \sim q$, if there is a bisimulation R such that $(p, q) \in R$.

Chapter 2

Molecular Interaction Maps

Molecular Interaction Maps (MIMs) is a graphic formalism to model chemical and biological interactions between molecules in a schematic way.

A MIM is a powerful diagram convention that is capable of flexible representation of networks containing multi-protein complexes, protein modifications, and enzymes that are substrates of other enzymes. This graphical representation makes it possible to view all of the many interactions in which a given molecule may be involved, and it can portray competing interactions, which are common in bioregulatory networks [23].

MIMs, also known as Kohn maps, were firstly described by Kurt W. Kohn in a 1999 article (see [22]) in order to create a tool to organize the known interactions in the form of a diagram. The choice of a map was based on the idea that the eye could catch important features out of a graphical depiction more than from a list of equations. Hence, MIMs were born with the target of transmitting the important facts of a given process more than as formal, unambiguous language to describe it.

2.1 Needs and Motivations for MIMs

The complexity of the molecular interactions implicated in cell regulatory networks[1] challenges human comprehension. Before MIMs, diagrams of molecular interactions were often lacking clearness or incomplete. The preparation of more comprehensive regulatory network diagrams were both difficult and urgent. The difficulties were not due merely to the large number of reactions, already present in familiar metabolic diagrams, but rather to complexities

[1] a cell regulatory network is a collection of molecules or complexes in a cell which interact with each other and with other substances in the cell, governing the production and degradation of new complexes and molecules.

that rarely occur in classical pathway diagrams, such as multisubunit complexes, protein modifications, enzymes that are modified by other enzymes, and protein domains whose function is regulated by other domains of the same molecule.

Why do biologists need molecular interaction maps? First, it is often difficult to keep in mind all of the known interactions that may be pertinent to a particular experimental or theoretical question, and a molecular interaction map can be used in much the same way as a road map or electronic circuit diagram. Second, molecular interaction maps can suggest new interpretations or questions for experiment. Third, the act of preparing a molecular interaction map imposes a discipline of logic and critique to the formulation of functional models. Finally, the diagram convention provides a shorthand for recording complicated findings or hypotheses.

Another kind of difficulty in preparing useful maps is the incompleteness and uncertainty of knowledge, as well as the limited scope of applicability of some interactions. An important aspect of molecular interaction maps is that they are linked to an annotation list that summarizes current information relevant to particular interactions and provides references. A molecular interaction map can therefore function also as a review. The maps can be updated interactively via the Internet and thus can provide a current summary of an area.

2.2 MIMs explained

In a MIM, each named molecule is shown only once to facilitate tracing all the known interactions of any given molecular species. A variety of defined connecting lines serve to describe the interactions between the molecules. Because each molecular species in a diagram should appear only once, interactions must be indicated by several types of lines connecting the species. The different types of interaction lines are distinguished by different kinds of arrowheads or other line endings, as summarized in Figure 2.2. Multimolecular complexes or modified forms are depicted by "nodes" placed on the lines. A line may originate either at a named molecular species or at a node, and may terminate at a molecular species, a node, or at another line. Lines that cross do no imply an interaction.

A unique aspect of the MIM notation is that it can show all of the known interactions and allow the unknown contingencies (effects of one interaction on another) to be left unspecified until those details become available. In this sense, MIM diagrams are "heuristic". A heuristic MIM therefore may not provide all the information required for computer simulation. Particular

models for computer simulation can, however, be extracted from heuristic MIMs and formulated in "explicit" diagrams using a subset of the MIM symbols [23].

Heuristic MIMs are "canonical" in that they are not restricted to a particular cell type or cell state, and they do not indicate a particular sequence of events. Rather, they show the interactions that can occur if the relevant molecules are present in the same place at the same time.

Molecules within a cell can interact with each other in different ways: let's see a couple of examples of possible biochemical reactions using an intuitive equation notation:

$\mathbf{A + B \rightarrow A:B}$ Molecules A and B bind together, forming a new complex molecule, called a dimer, named A:B; this reaction is termed "complexation", the reverse reaction is termed "decomplexation"; if a molecule of a certain species binds to a molecule of the same species, the complex is named homodimer;

$\mathbf{A \rightarrow pA}$ A phosphate p is added to a molecule A, this modification is termed "phosphorylation". In an equivalent manner, other modifications can apply to a molecule, in particular to proteins, such as acetylation, ubiquitination, etc. A modification usually activates an otherwise metabolically inert molecule, that, once modified, becomes ready to interact with other molecules. Note that the phosphate p is not shown on the left side of the reaction: it is considered to be quite common and always available if needed.

2.3 MIM Syntax

Here below we will sketch MIM syntax as designed by Kohn. For a exhaustive explanation refer to [23].

2.3.1 Molecular Species and Interactions

As said, MIM maps are simple graphs whose nodes represents molecules and arcs interactions between them. Each molecule generally appears only once on a map, so that it can gather all the interactions involving such molecule in one place. As said, molecular species can be of two kinds: elementary or complex. Complex molecular species are combinations or modifications of elementary species. In Figure 2.1 A, B and C are elementary species while x,y and z are complex ones. Depending on the scale adopted, an elementary species can represent either a single atom or a group of bounded proteins:

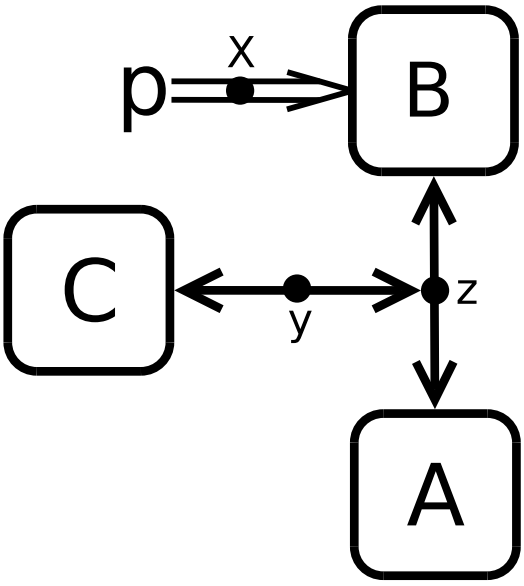


Figure 2.1: A simple Molecular Interaction Map

what it matters is that in that map, the species is considered "atomic", hence its internal structure is not important for the map at hand.

Interactions

Figure 2.2 lists all actual interactions, as last defined in [23]. Note that this can change in time, as biologists can decide to "merge" interactions now considered different, to split up one interaction in more specialized ones or extend the meaning of interactions now restricted to some kind of molecules to different ones (and changing the symbols in the process). The latter is what brought the creation of the covalent bond (Figure 2.2f) that was not present in [22].

Noncovalent (reversible) binding between molecular species is denoted by a line with barbed arrowheads at both ends (Figure 2.2a).

Covalent modification (phosphorylation, acetylation, myristoylation, ubiquitination, and so on) is represented by a line with a barbed arrowhead at one end pointing to the modification site (Figure 2.2b). The bond cleavage symbol (Figure 2.2f) is used to show dephosphorylation by a phosphatase. (The zig-zag symbol indicates a reaction that catalyzes bond cleavage.)

Covalent binding between proteins or between sites within the same protein sometimes require a symmetrical symbol, for which purpose the double-

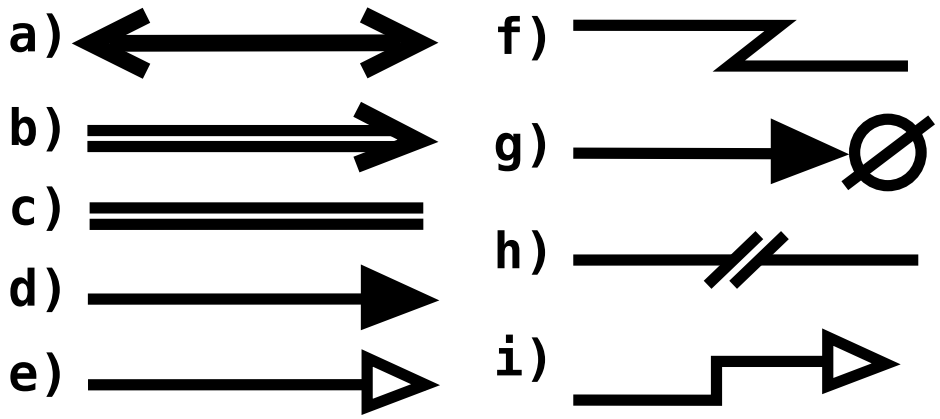


Figure 2.2: Molecular interaction symbols

line symbol shown in Figure 2.2c has been recently adopted.

Degradation is indicated by a filled triangle pointing to a null symbol (stoichiometric conversion to debris) as in Figure 2.2g.

Stoichiometric reaction (reactants converted to a corresponding number of product molecules) is indicated by a filled triangle arrowhead (Figure 2.2d). This symbol can be used also for translocation events, because molecules disappear from one location and reappear in another location, logically the equivalent of a stoichiometric reaction.

Any reaction that involves molecules on a different side of a membrane (cell membrane, nucleic membrane, etc. . .), called a reaction in-trans, make use of the splitted line (Figure 2.2h).

Production of a molecular species without loss of macromolecular reactants (as in transcription or translation) is indicated by an open triangle arrowhead pointing to a molecular species (small ubiquitous molecules, such as ATP or phosphate group, can usually be ignored)(Figure 2.2e). Production by transcription is indicated by an open triangle at the end of a hooked line (Figure 2.2i).

2.3.2 Contingencies

Figure 2.3 shows the four different contingencies that can be applied to any interaction. A contingency is a modification of the reaction and hence is an arrow that point from a molecular species to an interaction arrow. Fig. 2.3a shows the inhibition arrow: the interaction pointed is inhibited if the molecular species is present. Fig 2.3b shows the stimulation arrow: the

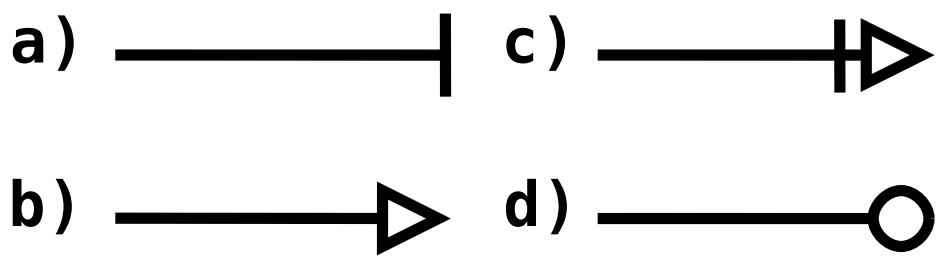


Figure 2.3: Contingency symbols

interaction is stimulated, that means that it's reaction rate is increased if the molecule connected by the arrow is present in the system. Fig. 2.3c depicts the necessity arrow: this is the dual of the inhibition and means that if the complex connected is not present, the reaction cannot happen. Last is the catalysis arrows (Fig. 2.3d) that describe a situation where a reaction has a different rate whether the molecular group connected by the arrow is present or not (typically, high when present and very low when not).

2.3.3 Interpretation of Maps

Kohn's maps are ambiguous. This, in biologists' view, is a feature more than a problem, as it allows a scientist to describe multiple interaction scenarios using a single map which can be read in different ways. This is a feature when you're using the map as a knowledge vehicle, whose captions are used to explain complex maps to students or peers. But it is a problem when you want to computer simulate the reactions described by the map. Let's see how and why maps are ambiguous, how biologists solve the problem when doing simulations and how we will tackle it in a different way.

Ambiguities

The most important ambiguities comes from different possible interpretations that the same map can describe. Figure 2.1 shows a very simple map but it hides some important choices that must be done before a simulation is possible:

1. can the phosphate group p link with B even if B is already in a bond with A?

2. can B modified by p (pB) still connect with A?

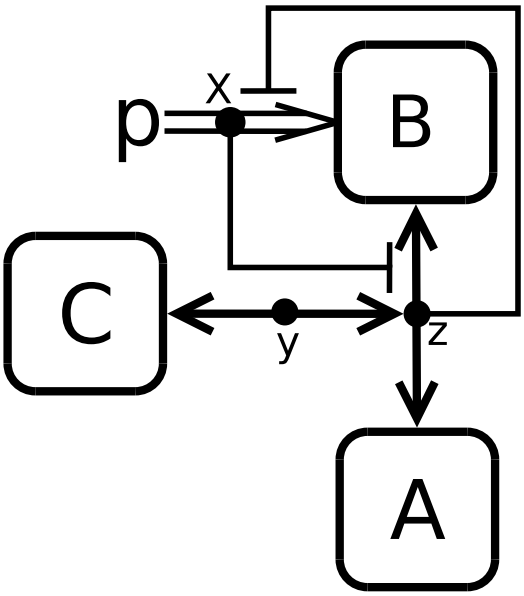


Figure 2.4: MIM of Figure 2.1 with mutual inhibitions to solve some ambiguities

3. can C connect either with $A{:}B$ or with $A{:}pB$?

These questions must be answered before any attempt to computer simulate the map can be done. To do that, it enough to add contingencies to enable or disable any spurious reaction. Figure 2.4 shows how we can solve problems 1) and 2) by adding the corresponding inhibitions. Actually, problem 3) will be solved by a syntax choice: if C was to complexate with $A{:}pB$, the arrow should have been placed against x. If the arrow points to z that means that C can only link with $A{:}B$.

Rewrite rules

In many cases, the method used to solve the ambiguities are "rewrite rules": those are rules, like the one used above, that modify the map to prohibit some interactions, to force an order to reactions' sequence or to eliminate some of the symbols from the map. This last method is suggested by Kohn himself in [23]: to prepare a computer simulation-ready map, he "preprocess" the diagram and remove all the interactions that are not allowed or forbidden (given the actual molecule quantities). In the thesis [17] we can find a way to rewrite the stimulation and the catalysis contingencies using only the other two (inhibition and necessity). We will use that result to include in out

semantics only those two contingencies. The thesis [24] instead uses reaction rate modifications to express contingencies, not much differently from what Kohn proposes. Our approach will be slightly different: we want a clear and unambiguous semantics, hence we will hold on to a strict interpretation of connections (like said before about solving problem 3) and we will require all the reduction rules to have a single interpretation, hence forcing slightly differences from the "canonical" maps.

Chapter 3

MIM-Calculus

In previous chapter we presented MIMs and all the basic formalism needed for our work. Here we will define formally our calculus, by giving definitions of its syntax and semantics..

3.1 Syntax and Semantics

We define any molecular kind (simple or complex) as a process.

3.1.1 Syntax

A process has a name and a (possibly empty) set of rules. The name describes the different elementary molecules that the process holds and the order and the interaction that were used to group them. The rules describe all (and every) interactions that the process can undertake. If a rule is not present in the set, than the process cannot enter in that reaction. Processes can be placed in parallel by the $\shortmid$ operator: processes in parallel can interact as long as there is a rule that allows them to.

Definition (Syntax)**.** MIM-Calculus' processes' syntax is:

$$\mathcal{P} \ ::= \ \mathcal{N} \mid \{r\}.\mathcal{N} \mid \mathcal{P}_1 \shortmid \mathcal{P}_2 \mid c \mid \varnothing_{\mathcal{P}}$$

where N is a complex name, whose syntax is

$$\mathcal{N} \ ::= \ A, B, \cdots \mid \varnothing \mid \mathcal{N}_1 : \mathcal{N}_2 \mid c\overline{\mathcal{N}} \mid \overline{\mathcal{N}_1 \mathcal{N}_2}$$

and where r is a rule, whose syntax is

$$r \ ::= \varepsilon \mid \xrightarrow[\{\mathcal{N}_n\},\{\mathcal{N}_i\}]{\mathcal{N}} r \mid \xleftarrow[\{\mathcal{N}_n\},\{\mathcal{N}_i\}]{r_1,r_2} \mid \xrightarrow[\{\mathcal{N}_n\},\{\mathcal{N}_i\}]{\{\{r\}.\mathcal{N}\}} \mid \underset{\{\mathcal{N}_n\},\{\mathcal{N}_i\}}{\overset{\mathcal{N}}{=\!=\!=}} r$$
$$\mid \underset{\{\mathcal{N}_n\},\{\mathcal{N}_i\}}{\overset{\mathcal{N}_1\mathcal{N}_2}{\leftrightsquigarrow}} \mid \underset{\{\mathcal{N}_n\},\{\mathcal{N}_i\}}{\overset{c\mathcal{N}}{\leftrightsquigarrow}} \mid \underset{\{\mathcal{N}_n\},\{\mathcal{N}_i\}}{\overset{r_1,r_2}{=\!\not=\!=}} \mid \xrightarrow[\{\mathcal{N}_n\},\{\mathcal{N}_i\}]{\{\{r\}.\mathcal{N}\}} \mid r_1 \shortmid r_2$$

and c is a small compound, like a phosphate or an ubiquitin, that cannot be considered an autonomous molecule but that sometime reacts with bigger processes:

$$c \ ::= p \mid u \mid \cdots$$

In this syntax, each term (or process) has a name that can be an elementary name $A, B, \ldots$ or a composition of them through the complexation operator : or the covalent operator $\overline{\quad}$ or the void name $\varnothing$ (the void name is sometimes used to model interactions that originate from the environment or from unseen or unknown processes).
The set of rules can be empty or it can contains any number of rules in parallel. All the rules in a given set have the same priority. Every rule must explicitly lists all the contingencies affecting it, expressed by the two sets on its lower part: $\mathcal{N}_n$ is the set of the necessities (the names that must be present in the system for the reaction to be allowed) and $\mathcal{N}_i$ is the set of the inhibitions (the names that must NOT be in system or else the reaction is forbidden).
Given the syntax definition, a string that describe the processes of Figure 2.4 is the following:

$$\{\xrightarrow[\varnothing,pB]{B}\mu\}.A \shortmid \{\xrightarrow[\varnothing,pB]{A}\mu, \underset{\varnothing,A:B}{\overset{p}{=\!=\!=}}\}.B \shortmid \{\xrightarrow[\varnothing,\varnothing]{A:B}\}.C$$

where $\mu = \{\xrightarrow[\varnothing,\varnothing]{C}\}$.
This is an unambiguous term, whose evolution can be computer simulated. Note that in absence of reaction rates, the probability of the complexation between A and B or of the phosphorylation of B is the same.

3.1.2 Structural Congruence

The structural congruence of the new semantics is as follow:

$$\begin{aligned} \mathcal{P} \mid \mathcal{Q} &\equiv \mathcal{Q} \mid \mathcal{P} \\ (\mathcal{P} \mid \mathcal{Q}) \mid \mathcal{R} &\equiv \mathcal{P} \mid (\mathcal{Q} \mid \mathcal{R}) \\ \mathcal{P} \mid \varnothing_{\mathcal{P}} &\equiv \mathcal{P} \\ \{\varepsilon\}.\mathcal{N} &\equiv \mathcal{N} \end{aligned} \tag{3.1}$$

$$\begin{aligned} \mathcal{N}_1 : \mathcal{N}_2 &\equiv \mathcal{N}_2 : \mathcal{N}_1 \\ \overline{\mathcal{N}_1 \mathcal{N}_2} &\equiv \overline{\mathcal{N}_2 \mathcal{N}_1} \end{aligned} \tag{3.2}$$

$$\begin{aligned} r_1 \mid r_2 &\equiv r_2 \mid r_1 \\ (r_1 \mid r_2) \mid r_3 &\equiv r_1 \mid (r_2 \mid r_3) \\ r \mid \varepsilon &\equiv r \end{aligned} \tag{3.3}$$

This congruence rules give us a way to manipulate terms so that their meaning stay unchanged. Besides the obvious ones, there are a couple that deserve a little explanation: equations 3.2 tell us that the complexation order or the covalent order of names are irrelevant to the meaning of the term. For example, in the term described above, $A:B$ is exactly the same as $B:A$.

3.1.3 Reduction semantics

This semantics gives the processes the rules with which they evolve. First we give meaning to all the symbols in the syntax then we introduce the reduction rules that show how processes are combined together through the interaction rules.

$$\frac{\mu_1 = \{X | \xrightarrow[N,I]{\mathcal{N}_2} \mu\}, \mu_2 = \{Y | \xrightarrow[N,I]{\mathcal{N}_1} \mu\}, \mu_3 = \{\mu | \xleftarrow[N,I]{\mathcal{N}_1.X, \mathcal{N}_2.Y}\}}{\mu_1.\mathcal{N}_1 \mid \mu_2.\mathcal{N}_2 \xrightarrow[N,I]{} \mu_3.\mathcal{N}_1 : \mathcal{N}_2} \qquad \text{(Complexation)}$$

$$\frac{\mu_0 = \{\mu | \xleftarrow[N,I]{\mathcal{N}_1.X, \mathcal{N}_2.Y}\}, \mu_1 = \{X | \xrightarrow[N,I]{\mathcal{N}_2} \mu\}, \mu_2 = \{Y | \xrightarrow[N,I]{\mathcal{N}_1} \mu\}}{\mu_0.\mathcal{N}_1 : \mathcal{N}_2 \xrightarrow[N,I]{} \mu_1.\mathcal{N}_1 \mid \mu_2.\mathcal{N}_2} \qquad \text{(Decomplexation)}$$

$$\frac{\mu_1 = \{X | \xlongequal[N,I]{c} \mu\}, \mu_2 = \{\mu | \xlongequal[N,I]{\not{X}}\}}{c | \mu_1.\mathcal{N} \xrightarrow[N,I]{} \mu_2.c\overline{\mathcal{N}}} \qquad \text{(Covalent Modification)}$$

$$\frac{\mu_1 = \{X|\overset{\mathcal{N}_2}{\underset{N,I}{=\!=}}\mu\}, \mu_2 = \{Y|\overset{\mathcal{N}_1}{\underset{N,I}{=\!=}}\mu\}, \mu_3 = \{\mu|\overset{X,Y}{\underset{N,I}{=\!\not=}}\}}{\mu_1.\mathcal{N}_1 \mid \mu_2.\mathcal{N}_2 \underset{N,I}{\longrightarrow} \mu_3.\overline{\mathcal{N}_1\mathcal{N}_2}} \quad \text{(Covalent Link)}$$

$$\frac{\mu_1 = \{X|\overset{c\mathcal{N}_2}{\underset{N,I}{\rightsquigarrow}}\}, \mu_2 = \{Y, \overset{W}{\underset{N_1,I_1}{=\!\not=}}\}, \mu_3 = \{W|\overset{c}{\underset{N_1,I_1}{=\!=}}\mu\}}{\mu_1.\mathcal{N}_1 \mid \mu_2.c\overline{\mathcal{N}_2} \underset{N,I}{\longrightarrow} c \mid \mu_1.\mathcal{N}_1 \mid \mu_3.\mathcal{N}_2} \quad \text{(cCleavage)}$$

$$\frac{\mu_1 = \{X|\overset{\mathcal{N}_2\mathcal{N}_3}{\underset{N,I}{\rightsquigarrow}}\}, \mu_2 = \{Y|\overset{\mathcal{N}_3}{\underset{N_1,I_1}{=\!=}}\mu\}, \mu_3 = \{W|\overset{\mathcal{N}_2}{\underset{N_1,I_1}{=\!=}}\mu\}, \mu_4 = \{\mu|\overset{Y,W}{\underset{N_1,I_1}{=\!\not=}}\}}{\mu_1.\mathcal{N}_1 \mid \mu_4.\overline{\mathcal{N}_2\mathcal{N}_3} \underset{N,I}{\longrightarrow} \mu_1.\mathcal{N}_1 \mid \mu_2.\mathcal{N}_2 \mid \mu_3.\mathcal{N}_3} \quad \text{(Cleavage)}$$

$$\frac{\mu_0 = \{X|\overset{\{(\mu.\mathcal{N})_i, i\geq 1\}}{\underset{N,I}{\longrightarrow}}\}}{\mu_0.\mathcal{N}_0 \underset{N,I}{\longrightarrow} \mu_1.\mathcal{N}_1 \mid \cdots \mid \mu_i.\mathcal{N}_i} \quad \text{(Stoichiometric Conversion)}$$

$$\frac{\mu_0 = \{X|\overset{\{(\mu.\mathcal{N})_i, i\geq 1\}}{\underset{N,I}{\longrightarrow\!\!\triangleright}}\}}{\mu_0.\mathcal{N}_0 \underset{N,I}{\longrightarrow} \mu_0.\mathcal{N}_0 \mid \mu_1.\mathcal{N}_1 \mid \cdots \mid \mu_i.\mathcal{N}_i} \quad \text{(LossLess Production)}$$

These rules give us a way to interpret all the symbols that where introduced in the syntax. As you can see, the semantics mimics the equivalent MIM interaction.

So the $\overset{-}{\rightarrow}$ arrow and its dual, the $\overset{-}{\leftarrow}$ arrow, describe the complexation-decomplexation interaction that in MIM is depicted by the double arrow line (Figure 2.2a). As you can see, rule (Complexation) describe a situation where two process are in parallel in the system and they both have a rule to complexate one with the other. Note that the process produced by the application of this rule has been given the complementary decomplexation rule to return to the original situation. For the same reason, in rule (Decomplexation) the products of the decomplexation have again the complexation rule.

Note that in both case, the sets of necessities and inhibitions have been inherited by the system evolution step: for these reactions to happen in this execution of the system, those contingencies must be respected by the whole environment. The other reactions are pretty much similar, with two notable exceptions. First, the covalent link and the covalent modification are not reversible. The unbonding must be executed by a third complex with that capability (cleavage). Second, the production and the conversion rules can have multiple molecules as their final products. This is somehow different

from canonical MIMs where in both cases a structural limit is posed on the quantities of products: the stoichiometric conversion must preserve the total atomic mass (hence the name) while the lossless production must preserve the total energy in the reaction. These limits comes from physical and chemical preservation laws that cannot be represented in our calculus. It's up to the user to write meaningful equations or not.

The last two rules of the semantics give us a way to interpret a system with multiple processes in parallel (rule (Conditions)) interacting with each other, with their sets of necessities and inhibitions that get checked and simplified at each step by the rule (Conditions) and that must be completely solved in order to apply the execution step to the system as a whole (rule (Final)). The function namesOf is used to extract the list of names from the current environment in order to be checked against the necessities and the inhibitions sets.

$$\frac{\begin{array}{c}\mathcal{P} \underset{N,I}{\longrightarrow} \mathcal{P}_1, \\ Q_n = \text{namesOf}(\mathcal{Q}), Q_n \cap I = \varnothing, \\ P_n = \text{namesOf}(\mathcal{P}), P_n \cap I = \varnothing, \\ N' = N \setminus Q_n\end{array}}{\mathcal{P} \mid \mathcal{Q} \underset{N',I}{\longrightarrow} \mathcal{P}_1 \mid \mathcal{Q}} \qquad \text{(Conditions)}$$

$$\frac{\mathcal{P} \underset{\varnothing,I}{\longrightarrow} \mathcal{P}_1}{\mathcal{P} \;\to\; \mathcal{P}_1} \qquad \text{(Final)}$$

where the function namesOf: $\mathcal{P} \to \{n \in \mathcal{N}\}$ is defined as follow:

$$\text{namesOf}(\mathcal{Q}) = \begin{cases} \varnothing & \text{if } \mathcal{Q} = \varnothing_{\mathcal{P}} \\ \mathcal{N}_1 \cup \text{namesOf}(\mathcal{P}_1) & \text{if } \mathcal{Q} = \mathcal{N}_1 \;|\; \mathcal{P}_1 \\ \mathcal{N}_1 \cup \text{namesOf}(\mathcal{P}_1) & \text{if } \mathcal{Q} = \{r\}.\mathcal{N}_1 \;|\; \mathcal{P}_1 \end{cases}$$

Chapter 4

Turing Equivalence of MIM-Calculus

In this chapter we want to show that MIM-Calculus is Turing equivalent by showing that the MIMs themselves are Turing equivalent.
To this end, we will show two results: first, MIMs without inhibition contingency have the same computational power of Petri Nets (which are NOT Turing equivalent). Second, if we add the inhibition contingency to MIMs and the inhibition arc to PN, the equivalence between them holds. And since inhibited Petri Nets are Turing equivalent (see [28]), this demonstrate that MIMs are actually Turing equivalent. In order to proceed, we will introduce Petri Nets formally and informally in a way convenient to our goal.

4.1 Petri Nets

A petri net is an abstract formal model of information flow. It can be described formally or via graphical intuitive representation.

Definition (Petri Nets)**.** Petri Nets are tuples $\{P, T, I, O\}$ where:
P is a set of Places
T is a set of Transitions
P and T must be disjoint: $P \cap T = \varnothing$
I is a function: $I \subseteq T \times \wp(P)$
O is a function: $O \subseteq T \times \wp(P)$

Such a description designs a static net, whose properties are equivalent to a graph with P and T nodes and I and O the arcs, respectively in input to and in output from a given transition. Marked petri nets (also known as

standard petri nets, or PN) are petri nets with "weights" associated to places via a marking function μ.

Definition (Marked Petri Nets)**.** Marked Petri Nets are tuples $\{P, T, I, O, \mu\}$ where P, T, I and O are defined exactly as for Petri Nets and where μ is the marking function for the places: $\mu \subseteq P \times \mathbf{N}$

Marked petri net are what normally is considered a Petri net and they can be "executed", that means that the μ function can be modified by choosing one of the "enabled" transition and "firing" it[1], hence augmenting or diminishing the marking value of the places specified by the I and O function.

Definition (Execution)**.** Executing one step of a standard petri net is obtained producing a new marking μ_1 from marking μ.

$$\mu_1(p) = \begin{cases} \mu(p) - 1 & \text{if } p \in I(t_0) \wedge p \notin O(t_0) \\ \mu(p) + 1 & \text{if } p \in O(t_0) \wedge p \notin I(t_0) \\ \mu(p) & \text{else} \end{cases} \qquad \text{(Execution)}$$

where t_0 is chosen from all the enabled[2] transition $t^e \in T$. The new marked petri net resulting from "firing" transition t_0 is then the tuple $\{P, T, I, O, \mu_1\}$.

Intuitively, a petri net is a graph. Figure 4.1 shows a simple Petri net. The pictorial representation of a Petri net as a graph used in this illustration is common practice in Petri net research. The Petri net graph models the static properties of a system, much as a flowchart represents the static properties of a computer program [28].

The graph contains two types of nodes: circles (called places) and bars (called transitions). These nodes, places and transitions, are connected by directed arcs from places to transitions and from transitions to places. If an arc is directed from node i to node j (either from a place to a transition or a transition to a place), then i is an input to j, and j is an output of i. In Figure 4.1, for example, place p_1 is an input to transition t_2, while places p_2 and p_3 are outputs of transition t_2.

In addition to the static properties represented by the graph, a Petri net has dynamic properties that result from its execution. Assume that the execution of a computer program represented by a flowchart is exhibited by

[1] this is not the most common definition, since choosing one transition to be fired takes away the possibility of parallel firings in mutually independent but enabled transitions. However, this doesn't interfere with the computational power issues at hand here (see [25] for reference).

[2] a transition t is enabled if "$\forall p \in I(t) \cdot \mu(p) \geq 1$".

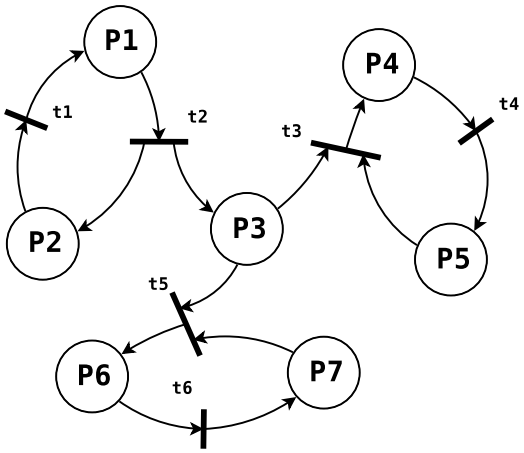


Figure 4.1: A simple graph representing a Petri Net

placing a marker on the flowchart to mark the instruction being executed, and that as the execution progresses, the marker moves around the flowchart. Similarly, the execution of a Petri net is controlled by the position and movement of markers (called tokens) in the Petri net. Tokens, indicated by black dots, reside in the circles representing the places of the net. A Petri net with tokens is a marked Petri net.

The use of the tokens rather resembles a board game. These are the rules: Tokens are moved by the firing of the transitions of the net. A transition must be enabled in order to fire. (A transition is enabled when all of its input places have a token in them.) The transition fires by removing the enabling tokens from their input places and generating new tokens which are deposited in the output places of the transition.

In the marked Petri net of Figure 4.2, for example, the transition t_2 is enabled since it has a token in its input place (p_1) Transition t_5, on the other hand, is not enabled since one of its inputs (p_3) does not have a token. If t_2 fires, the marked Petri net of Figure 4.3 results. The firing of transition t_2 removes the enabling token from place p_l and puts tokens in p_2 and p_3, the two outputs of t_2.

The distribution of tokens in a marked Petri net defines the state of the net and is called its marking. The marking may change as a result of the firing of transitions. In different markings, different transitions may be enabled. For example, in the marked net of Figure 4.3 three transitions are enabled: t_1, t_3, and t_5, none of which were enabled in the marking of Figure 4.2. In this situation, we have a choice as to which transition will fire next.

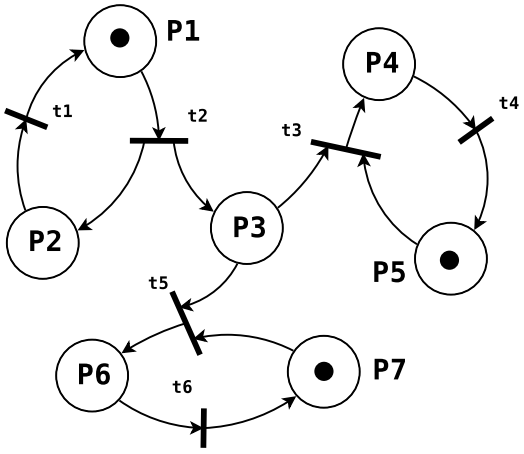


Figure 4.2: A marked Petri Net

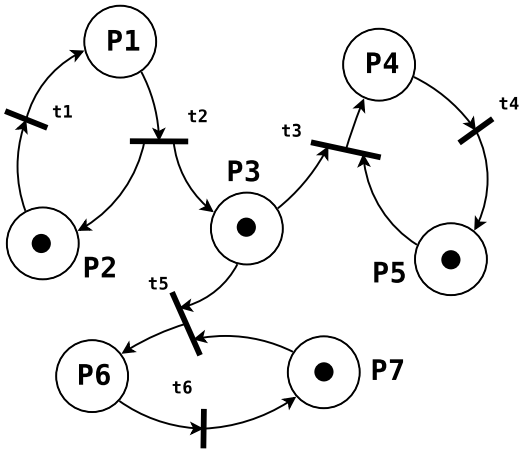


Figure 4.3: The marking resulting from firing transition t_2 in Figure 4.2. Note that the token in p_l was removed and tokens were added to p_2 and p_3

4.2 Turing Equivalence by Reduction to Petri Net

Establishing the computational power of a given formalism is often not an easy task. The easiest way to accomplish this is normally to "reduce" the formalism to another one whose computational power is already known.

Normally we "reduce" problems: $A \leq B$ (A is reducible to B) means that each and every solution to A can be found by solving B and translating the solution somehow. Hence, solving problem A cannot be harder than solving problem B.

Since a formalism is a method to model problems, we can extend the idea of reduction: to reduce a formalism A to another formalism B ($A \sqsubseteq B$) means that every problem written in formalism A can be modeled through formalism B, given a simple translation process. Hence, if we know the computational power of B, and $A \sqsubseteq B$, problems in A can be solved with at most the same effort.

A Turing-equivalent system is one that can simulate, and be simulated by, a universal Turing machine. So, if B is Turing-equivalent and $A \sqsubseteq B$, we can say that problems in A can be solved by a universal Turing machine. On the other hand, if we show that $B \sqsubseteq A$, we demonstrate that A can solve any problem solvable by a universal Turing machine. So, showing that B is Turing-equivalent and that both $A \sqsubseteq B$ and $B \sqsubseteq A$ relations hold, is enough to demonstrate that A is Turing-equivalent as well.

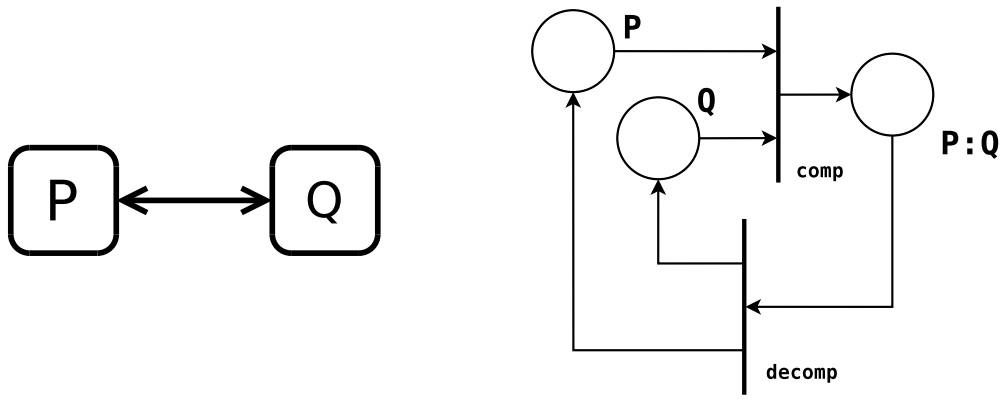
Petri nets, with their modeling properties and extensive literature, are a perfect candidate for this. Since their introduction in 1962, a large number of results about Petri Nets properties have been proved. Many concentrate on their computational power (and the computational power of PNs extended in a way or another). Here we will need only a couple of results:

- Standard Petri Nets are **not** Turing equivalent;
- Petri Nets extended with an inhibition arcs[3] are Turing equivalent (see [28]).

4.2.1 Equivalence of Non-Inhibited MIMs and Standard Petri Nets

We will reduce MIMs without the inhibition contingency to Petri Nets and vice-versa. We will be doing this by a structural approach: for each MIM and

[3]also called "negative context conditions" since they are read but not consumed by the transition, see [25]



(a) MIM Graphic Notation (b) Petri Net

Figure 4.4: Complexation and Decomplexation Interactions modeled through different notations

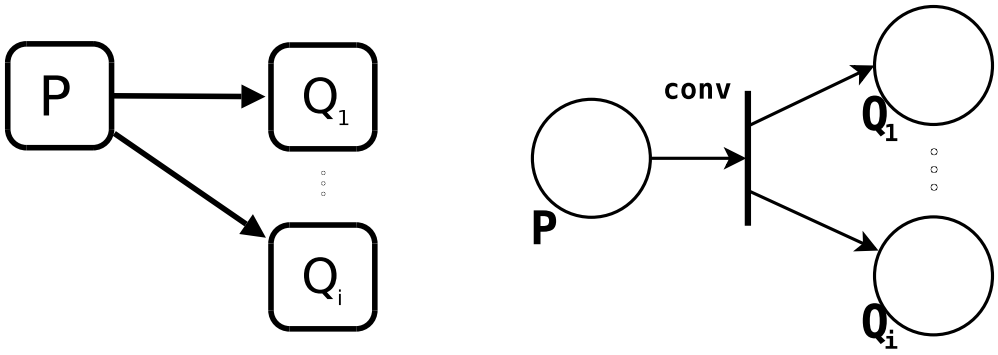
Petri Net primitive[4] we will give an equivalent model in the other formalism, hence demonstrating the equivalence by a loose type of structural induction on the rules (loose mainly because we don't bother to prove that all those pieces are necessary).

Reduction of MIMs to Standard Petri Nets

MIM primitives are elementary species, complex species and all the different interactions. Each of these can be mapped onto a petri net structure: elementary and complex species can be represented by labeled places while interactions can be modeled through transitions and their input and output arcs. By modeling all elements of MIM with equivalent Petri Net, we demonstrate that MIM $\sqsubseteq$ PN and so we show that solving a problem described in MIM notation cannot be harder than solving a problem described in PN notation. This is enough to assert that MIMs are NOT Turing Equivalent. So, here below you can find each MIM interaction both in MIM graphic notation and in its equivalent petri net.

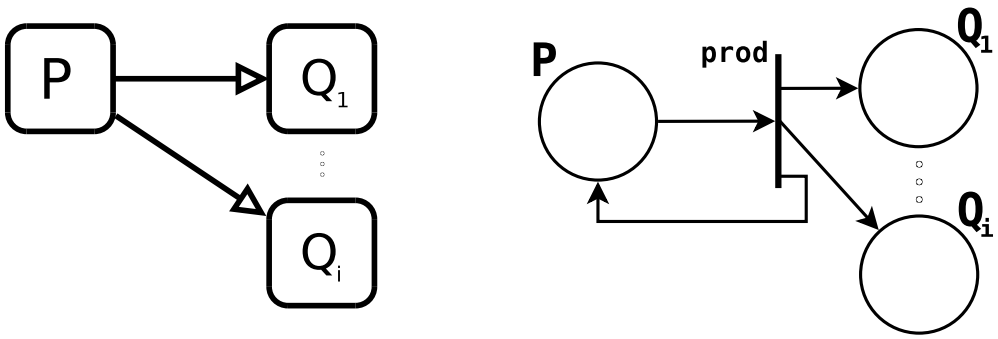
The complexation (and the decomplexation) of two elementary species A and B into the complex species $A{:}B$ in MIM can be seen in Figure 4.4a. In Figure 4.4b the equivalent petri net can be observed: note how the two transitions comp and decomp describe precisely the double interaction synthesized in the double arrow of MIMs. As you can see, we are only describing struc-

[4]we use a somehow intuitive concept of primitive: primitives are constituent elements who can be (or not) necessary to model the formalism, but that are sufficient to.



(a) MIM Graphic Notation (b) Petri Net

Figure 4.5: Stoichiometric Conversion modeled through different notations



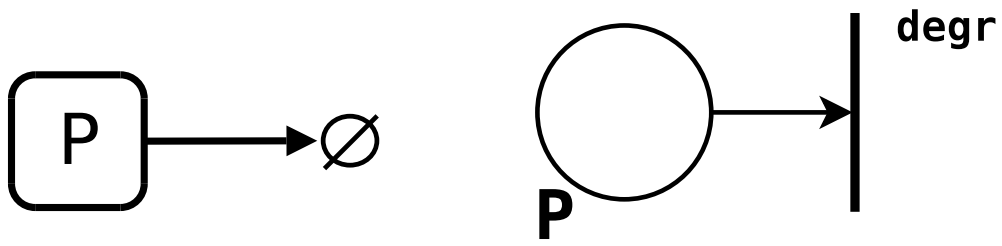
(a) MIM Graphic Notation (b) Petri Net

Figure 4.6: Lossless Production modeled through different notations

tural properties: the petri net is not marked and so it cannot be "executed" in the same way as the MIM of figure 4.4a cannot be simulated without knowing the exact quantities of reactants A and B in the environment.

Stoichiometric Conversion rule can be seen in Figure 4.5a. The rule determines how a species P converts to different species Q_i. The petri net in Fig.4.5b is pretty self-explanatory.

Lossless Production rule can be seen in Figure 4.6a. The rule determines how a species P produces different species Q_i without being consumed. Figure 4.6b depicts the equivalent petri net, where transition "prod" consume and reproduce a token in the place P. As explained in [25], this is not exactly the same as if P would simply stay put and produce the Q_i. However, we are not interested in parallel computation properties here and so we can ignore the difference. Even more, with the petri net rule Execution we decided that our petri net would fire only one transition at a time hence avoid-



(a) MIM Graphic Notation (b) Petri Net

Figure 4.7: Product Degradation modeled through different notations

ing the problem altogether. Note that we still can have concurrency and non-determinism, as multiple transitions (or multiple interactions, in case of MIM) can be enabled at a given time, and which one will fire is totally unknowable beforehand.

Figure 4.7 shows both MIM and petri net notation for the Product Degradation rule. Note that this could be seen as a particular case of Stoichiometric Conversion, where molecule P transforms in the null complex $\varnothing$.

Figure 4.8 describes the covalent link and modification interactions along with the respective cleavages and their equivalent PNs.

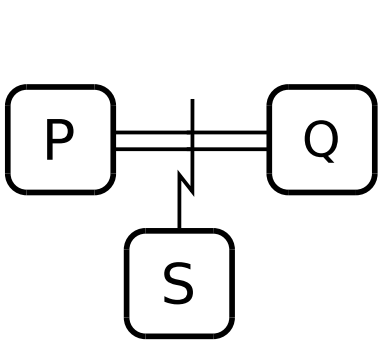
Reduction of Standard Petri Nets to MIMs

Now we want to model Standard Petri Nets with MIMs. We are not interested in the efficiency of the simulated model hence we can introduce artifacts that will help in the translation.

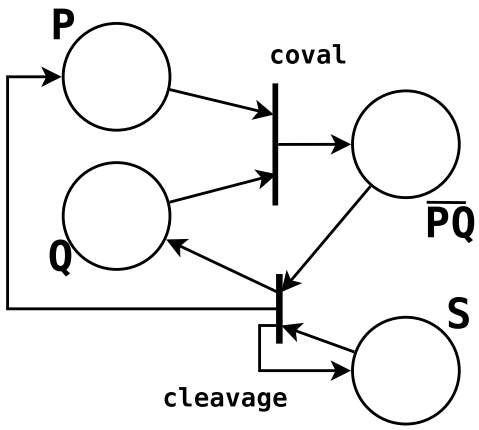
We take into consideration only two PN primitive structures [30], Synchronization (Figure 4.9a) and Concurrency (Figure 4.9b), since the others are transformed into MIM by reversing the MIM modeling we presented earlier (see page 23).

Since we are modeling PNs with MIMs, we are forced to model transitions that can have any number of inputting and outputting arcs with rules that can take at most two input (the covalent link, see rule (Covalent Link)) and any number of output (the stoichiometric conversion, see rule (Stoichiometric Conversion)). So while transition with multiple output (the Concurrency primitive) will be easily drawn with a MIM (see Figure 4.13), transition with multiple input (the Synchronization primitive) will require some modifications. In particular, we will need to consume all the inputs by "eating" at most two at a time.

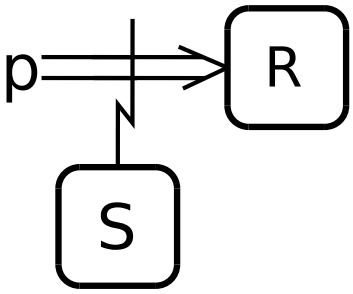
To do so preserving the semantics, we will have to introduce a global



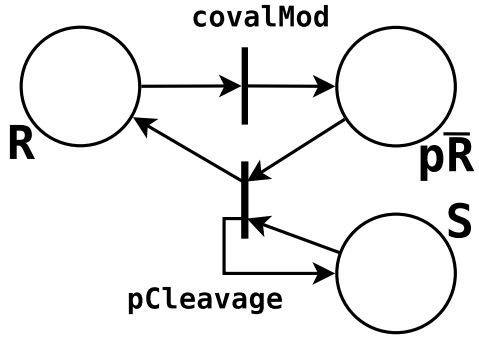
(a) Link with MIM



(b) Link with Petri Net



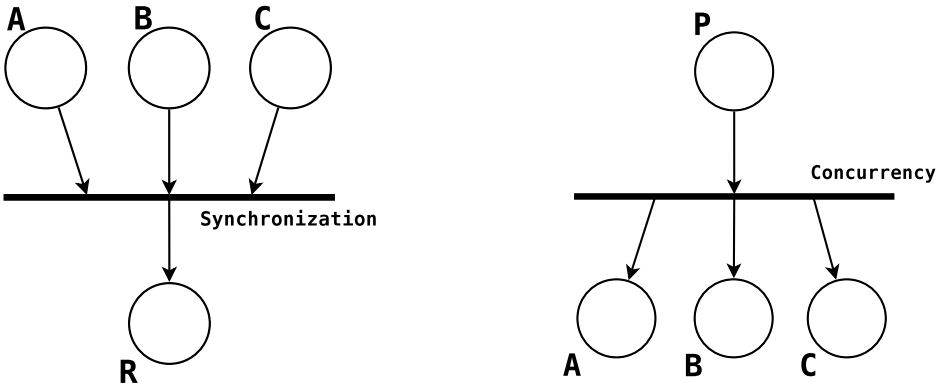
(c) Modification with MIM



(d) Modification with Petri Net

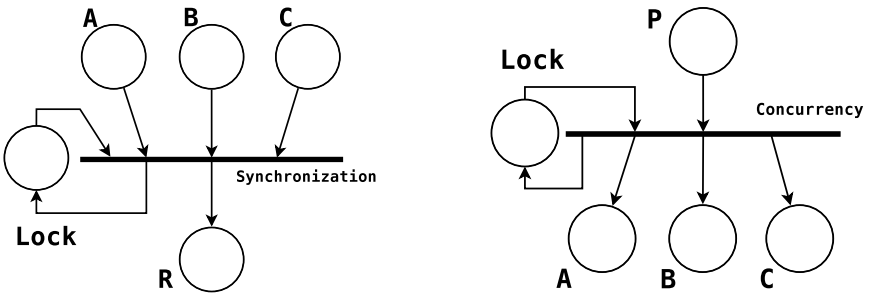
Figure 4.8: Covalent Link, Modification and Cleavage modeled through different notations

"lock", a predefined place with an initial marking of 1 that is in input to all PN's transitions (see Figure 4.10a). Since each transition must consume the lock to fire (and it must replace it afterward, obviously), this simple trick is able to serialize the firings. While the lock has the token, every transition that would be enabled without the lock is still enabled and while the lock has no token, only the transition that took it can execute. This is precisely what always happens in a Petri Net with the definition of execution we gave in page 20 but we will need the locking mechanism when some transitions will be expanded in more than one step. Taking the lock means disabling all the other transitions: this way the transition being fired can complete all the steps it needs to do to simulate the original transition it models (a kind of "atomic" action, a protected sequence of steps that cannot be interrupted) thus preserving the semantics of the original PN.



(a) Synchronization (b) Concurrency

Figure 4.9: Two Petri Nets primitives



(a) Synchronization (b) Concurrency

Figure 4.10: Synchronization and Concurrency with explicit lock

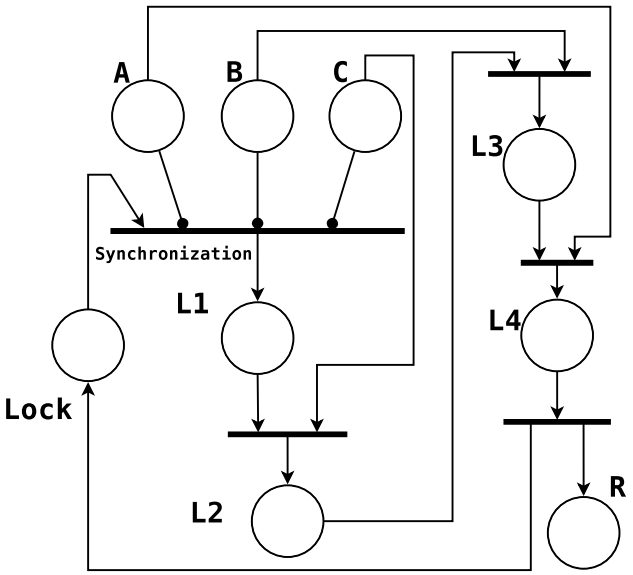


Figure 4.11: Synchronization through multiple steps and lock

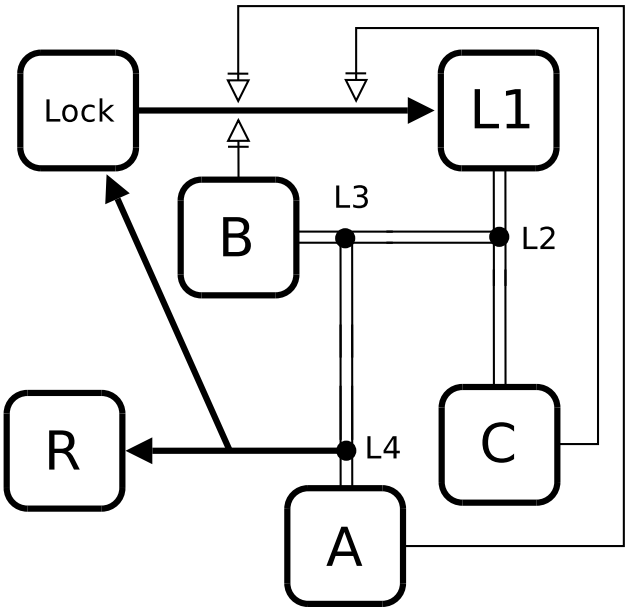


Figure 4.12: Synchronization modeled with MIM

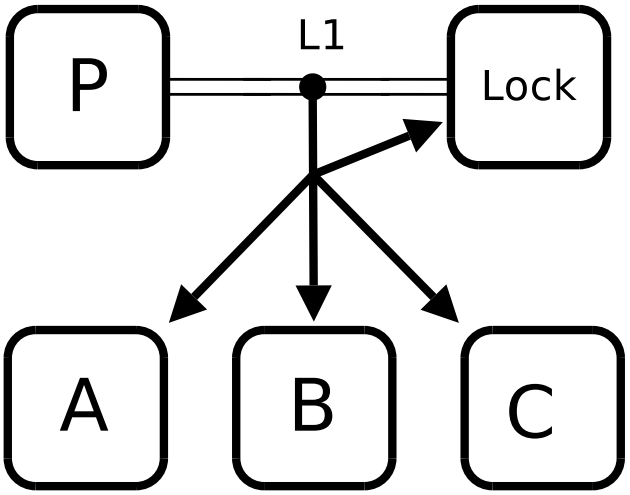


Figure 4.13: Concurrency modeled with MIM

To be able to correctly simulate the synchronization primitive, we introduce a new special kind of arc, dual to the inhibitor arc, the necessity arc (an arc with a filled circle as head): this arc when inputting in a transition, disables the transition if the source place is empty and enables it when is not empty (just like normal arcs) but then firing the transition does not consume any token off the place.

With this new arc, we can model our synchronization primitive through multiple binary steps, the first being the most important (see Figure 4.11): the transition *synchronization* is enabled if and only if places A, B, C and **lock** have each at least a token, but only the lock is consumed, thus disabling all other transitions and guaranteeing the atomicity of the whole process. Next steps will each consume one of the input tokens, whose presence is assured by the necessity arcs and the fact that all other transitions are disabled, while the last one produces the end result and returns the lock, exiting the protected section and re-enabling the rest of the transitions. Figure 4.12 shows the equivalent MIM.

As said above, the concurrency primitive is modeled naturally from the original PN (Figure 4.10b): the equivalent MIM can be seen in Figure 4.13.

4.2.2 Equivalence of Inhibited MIMs and Inhibited Petri Nets

This last part of our demonstration is actually pretty easy.

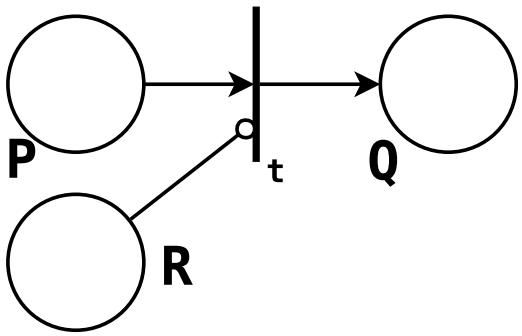


Figure 4.14: An inhibited Petri Net: transition t cannot fire if place R has a token

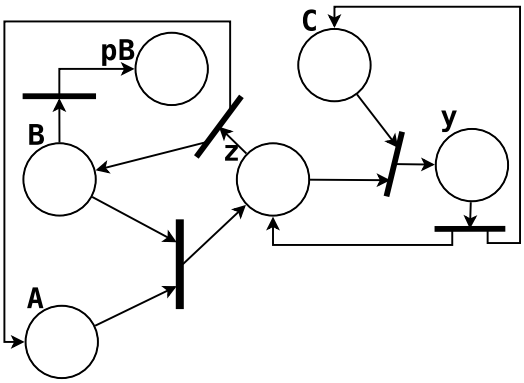


Figure 4.15: An simple Petri Net equivalent to MIM of Figure 2.1

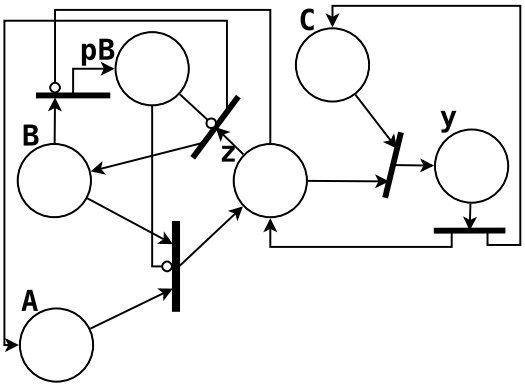


Figure 4.16: An simple Petri Net equivalent to MIM of Figure 2.4

PN to MIM: For any inhibition arc in the PN (an inhibition arc goes from a place to a transition, see Figure 4.14), in the equivalent MIM we add an inhibition contingency arrow from the correspondent complex to the correspondent interaction. More formally, ∀ inhibition arc from a place P inputting in a transition T, where in the resulting MIM T is modeled by interaction R and P by complex A, we add to the MIM an inhibition contingency arrow starting from A and pointing to R.

MIM to PN: For any inhibition contingency arrow in the MIM, in the equivalent PN we add an inhibition arc from the correspondent place to the correspondent transition. More formally, ∀ inhibition contingency arrow from complex A to an interaction R, where in the resulting PN R is modeled by transition T and A by place P, we introduce in the PN an inhibitory arc starting from P and inputting in T.

To make it even clearer, let's use a previous example: Figure 2.1 shows a simple MIM and Figure 2.4 shows that same MIM with explicit mutual inhibition. In figure 4.15 you will find a PN equivalent to the former example. In the figure 4.16 we added the inhibition arcs, in the same way the inhibition contingencies were added in the latter example, obtaining an equivalent petri net.

Chapter 5

Example

5.1 Example of translation

We took this example from [5](Figure 3c) because it is listed as an example of an explicit MIM, which can be used for computer simulation: In Kohn's words: "The model encoded in this MIM can be used for simulations of the phosphorylation process". This MIM depicts the phosphorylation of the

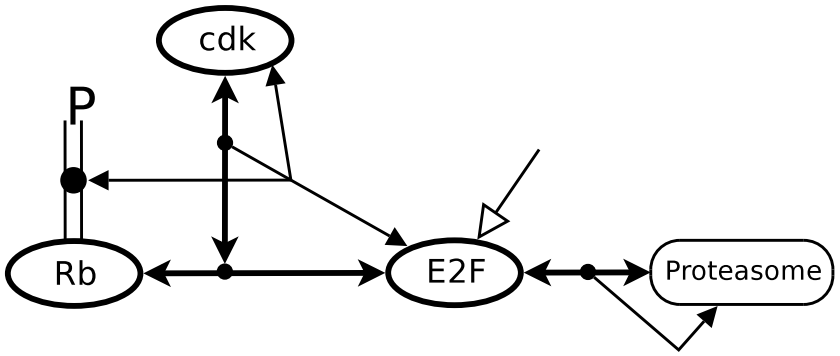


Figure 5.1: The phosphorylation of the retinoblastoma protein (Rb) by a cyclin-dependent kinase (cdk)

retinoblastoma protein (Rb) by a cyclin-dependent kinase (cdk). Rb forms a complex with E2F to create the Rb-E2F complex. This complex can be bound by cdk to create a complex Rb-E2F-cdk, which can then dissociate into E2F, cdk, and the phosphorylated form of Rb.

The concentration of E2F is determined by its rate of synthesis (line leading to E2F) and its rate of degradation in the proteasome; the explicit stages of the latter process are binding of E2F to the proteasome and creation

of an E2F-proteasome complex, followed by the disappearance of E2F from the complex.

Here below, you find the processes of the system: in the long line the initial agents (corresponding to the elementary molecules in the MIM), on their side the complex molecules that can be created at some stage during the simulation (corresponding to complex molecules, indicated on the MIM by a small black circle).

$$\mu_0.\varnothing \;\mid\; \mu_1.Rb \;\mid\; \mu_2.E_2F \;\mid\; \mu_3.cdk \;\mid\; \mu_4.Proteasome$$

$$\mu_5.Rb{:}E_2F$$
$$\mu_2.E_2F{:}Proteasome$$
$$\mu_3.cdk{:}(Rb{:}E_2F)$$

Here below you will find the rule set for each process in the system. Note that most of the complexity of MIM-Calculus lies in these set of rules.

$$\mu_0 = \{\xrightarrow{\mu_2.E_2F}\triangleright\} \qquad \mu_4 = \{\xrightarrow{E_2F}\mu_6\}$$
$$\mu_1 = \{\xrightarrow{E_2F}\mu_5, \overset{p}{=\!=}\} \qquad \mu_5 = \{\xrightarrow{cdk}\mu_7\}$$
$$\mu_2 = \{\xrightarrow{Rb}\mu_5\} \qquad \mu_6 = \{\xrightarrow{\mu_4.Proteasome}\bullet\}$$
$$\mu_3 = \{\xrightarrow{Rb:E_2F}\mu_7\} \qquad \mu_7 = \{\xrightarrow{p\overline{Rb},\mu_3.cdk,\mu_2.E_2F}\bullet\}$$

Chapter 6

Conclusions

In this work we described a graphic formalism called Molecular Interaction Maps, used by biologists and chemists to describe large, complex system of interacting molecules, proteins and cells. MIMs can be easily used to give lots of details about such a complex system in a intuitive manner, so that students and colleagues can concentrate on the meaning more than on some difficult syntax.

Later we developed a non-graphical equivalent language for MIMs, based on a Process Algebra approach and called MIM-Calculus. To MIM-Calculus we gave a strict syntax, a structural equivalence and an operational semantics to simulate MIM's behavior.

Finally, we demonstrated that MIMs and hence MIM-Calculus are Turing equivalent by using Petri Nets as a graphic formalism whose computability power is well known. To do so, we constructively simulated MIMs with Petri Nets and vice-versa, thus proving that MIMs with only the necessity contingency are equivalent to Petri Net (hence not Turing equivalent) and by adding the inhibition contingency to MIM (and the inhibition arc to Petri Net) they reach the Turing equivalence.

Bibliography

[1] Luca Aceto. Some of my favourite results in classic process algebra. In *In Bulletin of the EATCS*, pages 89–108, 2003.

[2] Luca Aceto, Wan Fokkink, and I Chris Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. Elsevier, 2001.

[3] T. Agerwala. Special feature: Putting petri nets to work. *Computer*, 12(12):85–94, Dec. 1979.

[4] T. Agerwala and Y.-C. Choed-Amphai. A synthesis rule for concurrent systems. *Design Automation, 1978. 15th Conference on*, pages 305–311, June 1978.

[5] Mirit I. Aladjem, Stefania Pasa, Silvio Parodi, John N. Weinstein, Yves Pommier, and Kurt W. Kohn. Molecular Interaction Maps–A Diagrammatic Graphical Language for Bioregulatory Networks. *Sci. STKE*, 2004(222):pe8–, 2004.

[6] Jos C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3):131–146, 2005.

[7] Roberto Barbuti, Giulio Caravagna, Andrea Maggiolo-Schettini, and Paolo Milazzo. An intermediate language for the simulation of biological systems. *Electr. Notes Theor. Comput. Sci.*, 194(3):19–34, 2008.

[8] Roberto Barbuti, Giulio Caravagna, Andrea Maggiolo-Schettini, Paolo Milazzo, and Giovanni Pardini. The calculus of looping sequences. In Marco Bernardo, Pierpaolo Degano, and Gianluigi Zavattaro, editors, *SFM*, volume 5016 of *Lecture Notes in Computer Science*, pages 387–423. Springer, 2008.

[9] Roberto Barbuti, Stefano Cataudella, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. A probabilistic model for molecular systems. *Fundam. Inform.*, 67(1-3):13–27, 2005.

[10] Roberto Barbuti, Andrea Maggiolo-Schettini, and Paolo Milazzo. Extending the calculus of looping sequences to model protein interaction at the domain level. In Ion I. Mandoiu and Alexander Zelikovsky, editors, *ISBRA*, volume 4463 of *Lecture Notes in Computer Science*, pages 638–649. Springer, 2007.

[11] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Simone Tini. Compositional semantics and behavioral equivalences for p systems. *Theor. Comput. Sci.*, 395(1):77–100, 2008.

[12] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. Bisimulation congruences in the calculus of looping sequences. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *ICTAC*, volume 4281 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2006.

[13] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. A calculus of looping sequences for modelling microbiological systems. *Fundam. Inform.*, 72(1-3):21–35, 2006.

[14] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. The calculus of looping sequences for modeling biological membranes. In George Eleftherakis, Petros Kefalas, Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa, editors, *Workshop on Membrane Computing*, volume 4860 of *Lecture Notes in Computer Science*, pages 54–76. Springer, 2007.

[15] C. Costea, D. Costea, V. Groza, and B. Groza. Petri net reduction. *Electrical and Computer Engineering, 2007. CCECE 2007. Canadian Conference on*, pages 1527–1530, April 2007.

[16] Michael J. Flynn and Tilak Agerwala. Comments on capabilities, limitations and correctness of petri nets. In *ISCA*, pages 81–86, 1973.

[17] Simone Fonda. Simulazione combinatoria a vincoli di mappe di interazione molecolare. Master's thesis, University of Pisa, 2007.

[18] C. Fournet and M. Abadi. Hiding names: Private authentication in the applied pi calculus, 2003.

[19] Jan Friso Groote and Marc Voorhoeve. Operational semantics for petri net components. *Theor. Comput. Sci.*, 379(1-2):1–19, 2007.

[20] C. A. R. Hoare and C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1985.

[21] Andrés Iglesias and Sinan Kapcak. Symbolic computation of petri nets. In Yong Shi, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *International Conference on Computational Science (2)*, volume 4488 of *Lecture Notes in Computer Science*, pages 235–242. Springer, 2007.

[22] Kurt W. Kohn. Molecular Interaction Map of the Mammalian Cell Cycle Control and DNA Repair Systems. *Mol. Biol. Cell*, 10(8):2703–2734, 1999.

[23] Kurt W. Kohn, Mirit I. Aladjem, John N. Weinstein, and Yves Pommier. Molecular Interaction Maps of Bioregulatory Networks: A General Rubric for Systems Biology. *Mol. Biol. Cell*, 17(1):1–13, 2006.

[24] Daniela Lepri. A formal semantics for molecular interaction maps. Master's thesis, University of Pisa, 2008.

[25] Ugo Montanari and Francesca Rossi. Contextual nets. *Acta Inf.*, 32(6):545–596, 1995.

[26] Ugo Montanari and Vladimiro Sassone. Dynamic congruence vs. progressing bisimulation for ccs. *Fundamenta Informaticae*, 16:171–196, 1992.

[27] MohammadReza Mousavi. *Structuring Structural Operational Semantics*. PhD thesis, Department of Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 2005.

[28] James L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, 1977.

[29] Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.

[30] Devinder Thapa, Suraj Dangol, and Gi-Nam Wang. Transformation from petri nets model to programmable logic controller using one-to-one mapping technique. In *CIMCA/IAWTIC*, pages 228–233. IEEE Computer Society, 2005.

[31] Hsu-Chun Yen. Introduction to petri net theory. In Zoltán Ésik, Carlos Martín-Vide, and Victor Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 343–373. Springer, 2006.