Università di Pisa
Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis

Formal Modelling and Simulation of Biological Systems with Spatiality

Giovanni Pardini

Supervisor
Prof. Roberto Barbuti

Supervisor
Prof. Andrea Maggiolo-Schettini

June 2011

Abstract

In Systems Biology, spatial modelling allows an accurate description of phenomena whose behaviour is influenced by the spatial arrangement of the elements. In this thesis, we present various modelling formalisms with spatial features, each using a different abstraction level of the real space. From the formalisms with the most abstract notion of space, to the most concrete, we formally define the *MIM Calculus with compartments*, the *Spatial P systems*, and the *Spatial CLS*. Each formalism is suitable for the description of different kinds of systems, which call for the use of different space modelling abstractions. We present models of various real-world systems which benefit from the ability to precisely describe space-dependent behaviours.

We define the MIM Calculus, inspired by Molecular Interaction Maps, a graphical notation for bioregulatory networks. The MIM Calculus provides high-level operators with a direct biological meaning, which are used to describe the interaction capabilities of the elements of such systems. Its spatial extension includes the most abstract notion of space, namely it only allows the modelling of compartments. Such a feature allows distinguishing only the abstract position where an element is, identified by the name of the compartment.

Subsequently, we propose a spatial extension to the membrane computing formalism *P systems*. In this case, we follow a more precise approach to spatial modelling, by embedding membranes and objects in a two-dimensional discrete space. Some objects of a Spatial P system can be declared as mutually exclusive objects, with the constraint that each position can accommodate at most one of them. The distinction between ordinary and mutually exclusive objects can be thought of as an abstraction on the size of the objects. We study the computational complexity of the formalism and the problem of efficient simulation of some kinds of models.

Finally, we present the Spatial Calculus of Looping Sequences (Spatial CLS), which is an extension of the Calculus of Looping Sequences (CLS), a formalism geared towards the modelling of cellular systems. In this case, models are based on two/three dimensional continuous space, and allow an accurate description of the motion of the elements, and of their size. In particular, Spatial CLS allows the description of the space occupied by elements and membranes, which can change their sizes dynamically as the system evolves. Space conflicts which may occur can be resolved by performing a rearrangement of elements and membranes. As example applications of the calculus we present a model of cell proliferation, and a model of the quorum sensing process in Pseudomonas aeruginosa.

Contents

Chapter 1

Introduction

Understanding how a biological system works is a complex matter, that requires studying its internal behaviour at different levels. For a long time, mathematical tools, such as differential equations, have been used to describe and study the behaviour of complex systems. Computer Science is also involved in this field of study. For example, computer science tools are commonly used in Biology to help to reconstruct knowledge about the internal functioning of biological systems, such as the reverse engineering of metabolic networks from experimental data.

Another approach, which has seen a widespread development in recent years, concerns the use of *formal methods* for the modelling and analysis of biological systems. An important aim of formal models is to enable the modeller to tackle the complexity of the systems under study. This is supported by the fact that formal models are intrinsically unambiguous, thus providing important advantages over the informal descriptions commonly used in Biology. Moreover, having a computational model of a biological system allows *in-silico* experiments, that is simulations of the behaviour of biological processes, which can be used by biologists as a support to rapidly test new hypotheses without having to resort to expensive and time-consuming *in-vivo* experiments. Another approach to the study of biological systems is given by the possibility of using formal analysis methods from Computer Science, such as model checking.

Apart from some earlier approaches to the use of computing formalisms for biological modelling, the most influential approach has been proposed by Regev, Shapiro and others in [74, 75, 67, 72], where the π-calculus process algebra [58] is used to describe biomolecular processes. The soundness of the approach is mainly due to the similarity between systems of interacting components, as studied in Computer Science, and the biological entities in a cell. In fact, the mentioned authors showed how molecules can be represented as interacting *agents*, where the interaction is described by a *communication* between the agents, and which results in a modification of the involved elements. Afterwards, many other formalisms originally developed by computer scientists to model systems of interacting components have been applied to Biology, and extended to allow more precise descriptions of the biological behaviours. Some other formalisms have also been developed expressly for being used in Biology.

Different formalisms allow studying different aspects of biological systems, depending on the features that they possess. Basic versions of formalisms usually allow only the description of *qualitative* aspects of the systems, while extensions of them are often

developed to also allow the description of *quantitative* aspects. Qualitative aspects deal with abstract state properties, such as reachability of states, but without taking into account any form of "quantity", such as time and probability, which instead constitute the quantitative aspects of a system.

Quantitative aspects are often described by means of *stochastic* formalisms. Actually, since stochastic modelling is particularly important in this field [88], the feature of stochasticity is often provided by formalisms for Biology. In fact, biological processes cannot be studied thoroughly by only using deterministic models, and stochasticity is mainly needed to abstract over some details of processes that are not known, or that cannot be described adequately. Gillespie's algorithm [40] is a widely used simulation algorithm for stochastic models of biological systems. Many other simulation methods have been proposed [41]. Apart from simulations, other analysis methods originated from Computer Science can be used, such as model checking (in particular, its probabilistic variant [53]).

In this thesis, we are interested in *spatial modelling*, which allows the description of the position of the elements of a system, with respect to some more or less abstract notion of space. The simplest form of spatial modelling is given by the ability to represent *compartments*, which allow the representation of different locations for the elements of the system. They may be either entailed by biological membranes or simply used to represent different abstract locations in a spatially heterogeneous environment.

The *MIM Calculus*, also extended with compartments, is the first formalism that we define, and provides the most abstract form of spatial modelling among the formalisms proposed in the thesis. We also investigate the possibility of going a step further in the precision of modelling, by developing other two formalisms, the *Spatial P systems* and the *Spatial Calculus of Looping Sequences*, which provide more concrete representations of the real space. In particular, the former is based on discrete space, while the latter is based on continuous space.

Various formalisms proposed in the literature allow different levels of representation of the space. Due to the important role of membranes in many biological processes, most computer science formalisms for biology allow some form of compartmental modelling. As regards models allowing more concrete representations of the space, that have been used to describe biological systems, we mention cellular automata [44, 62, 82], based on discrete space, and models based on differential equations, such as those describing the reaction diffusion process and spatial pattern formation [59, 60]. More recently, various formalisms allowing more concrete spatial modelling than membranes have been proposed. An extensive discussion on related works is presented in Chapter 2.

1.1 Motivations

In this thesis we develop various modelling formalisms with *spatial* features, namely with the ability to keep track of the "position" of the elements of a system. Spatial modelling is required to describe accurately many biological phenomena whose behaviour is influenced by the spatial arrangement of the elements.

The notion of compartments allows distinguishing only the abstract position of an element, identified by the name of the compartment. In this field, compartments are intended as closed areas which may contain elements and other compartments. As in the case of biological membranes, biological entities (such as molecules) can, in some cases,

cross compartment bounds.

Membranes play an important role in many biological processes. For example, the compartmentalization of the elements inside a cell is fundamental for its correct functioning, since it allows different processes to occur independently from one another in physically separated regions, which can communicate only through the release of chemical substances which can cross membrane bounds. Furthermore, membranes can actually play an active role in such a communication, by modulating the passing of chemical substances and signals through them. The possibility of describing compartments is provided by most of the formalisms for modelling biological systems. This often consists in the ability to represent membranes directly in the syntax, by means of some form of containment operator which also allows the building of hierarchies of compartments.

Another example involving membranes is that of *signalling pathways*, namely the processes that allows a cell to react to external stimuli by regulating some internal processes. In this case, receptors attached to the external membrane surface receive the external stimuli and, as a result, can trigger some internal process by releasing signalling proteins inside the membrane. A system of this kind is brought as an example of application in Chapter 3.

According to the system under study, the simplest form of spatial modelling, namely by means of membranes, may be enough to obtain faithful description of the system and of its behaviour. However, there are cases in which the fundamental assumption of compartmental modelling, namely the *well-stirred* assumption of the content of compartments, is not valid, since it would introduce too much approximation of the real situation.

For example, consider a *reaction-diffusion system*, which is composed of small elements which can freely diffuse in the environment, and can interact with one another. In case the rate of reaction is much higher than the rate of diffusion, the elements will tend to react among themselves rather than diffuse in the environment. Thus the time needed for the system to reach a well-stirred state is, with respect to the reaction rate, non-negligible. Since the behaviour of the system in such a time span can be quite different than the behaviour in the well-stirred state, this shows the importance of increasing the precision of modelling by keeping track of the position of the involved elements [84]. A similar situation happens in the model of the quorum sensing process described in Chapter 6.

Molecular crowding [84] is another case involving diffusion systems in which the *well-stirred* assumption does not hold. This happens when the number of molecules in a limited space is very high, effectively preventing the molecules from freely diffusing, and therefore changing the behaviour of the system.

The above examples show that, in order to obtain a faithful description of some kinds of systems, a more concrete spatial representation is necessary. Beyond microbiological systems, *ecological systems* are another field in which a spatial representation is useful such as, for example, in the description of population dynamics. In this case, compartments can easily model spatially separated regions, in which different populations live, and where individuals can move from a region to another. In Chapter 4 we present an example of this kind of systems, which describes, in particular, the evolution of *ring species*.

1.2 Contributions

We define three formalisms with spatial features, namely the *MIM Calculus*, the *Spatial P systems*, and the *Spatial Calculus of Looping Sequences*, each providing a different level of abstraction for the modelling of the real space. In fact, since they are tailored to the modelling of a different kind of systems, the appropriate level of abstraction is different from one another.

The first formalism that we define is the *MIM Calculus*, which is inspired by the *Molecular Interaction Maps*, a graphical notation for bioregulatory networks used in Biology. The MIM Calculus is defined in the style of a process calculus, and provides high-level operators with a direct biological meaning, which are used to describe the interaction capabilities of the elements of a system. We initially define the basic version of MIM Calculus, without any notion of spatiality, then we propose the *MIM Calculus with compartments*. As regards the basic version of MIM Calculus, we propose a notion of *consistency* of terms of the calculus, with the aim of identifying which terms constitute a formal representation of a MIM diagram. The MIM Calculus with compartments provides the most abstract notion of space among the formalisms proposed in the thesis, namely it only allows the modelling of *compartments*. We present a model of the G protein signalling pathway, which shows that the modelling of compartments is well-suited for this kind of systems.

We then present a spatial extension of the *P systems* formalism. P systems [69, 70], also known as *membrane systems*, originated as a computing formalism in the field of Natural Computing, and are inspired by the functioning of living cells. Later, they have been applied to the modelling of biological and ecological systems, usually by means of extensions developed on purpose, such as with the inclusion of stochasticity. The basic class of P systems is made of a hierarchy of membranes, where each membrane contains *objects*, represented by symbols of an alphabet, and *evolution rules*, namely rewrite rules which describe how the *reactant* objects evolve into *products*. In particular, objects can be transformed into other objects, and can be sent into other membranes.

The spatial extension of P systems that we propose is based on discrete space. In particular, membranes and objects are embedded in a two-dimensional discrete space with integer coordinates. Evolution rules are associated with membranes, and are also extended to allow specifying the (relative) positions of reactants and products with respect to the concrete position in which the rule is to be applied. A distinctive feature of Spatial P systems are the *mutually exclusive* objects, which are subject to the constraint that there can be at most one of them in each position. There are no constraints on the other (*ordinary*) objects. The distinction between ordinary and mutually exclusive objects can be thought of as an abstraction on the size of the objects.

In order to increase the usefulness of the formalism for the modelling of systems, we include in Spatial P systems the features of promoters and priorities among evolution rules. Each rule can have a multiset of objects, called *promoters*, which are required to be present inside the membrane in order for the rule to be applied. Priorities among evolution rules instead mean that a rule can be applied in a position only if there is no other higher priority rule which can be applied in place of it. Priorities are useful for the description of object movements.

Since P systems were initially devised as a computing formalism, the computational power of the basic class and various extensions, using different features and different forms of evolution rules, has been studied in the literature [70, 68]. From this theoretical point of

view, the feature of mutually-exclusive objects provided by Spatial P systems is a powerful feature. In fact, in order to study the computational power of Spatial P systems, we show that Turing universality can be achieved, if mutually-exclusive objects are allowed, by using only *non-cooperating* evolution rules, namely rules in which there is only one object in their left-hand part. As a comparison, the basic form of P systems with only non-cooperating rules are not universal. Conversely, if rules can have more than one object among the reactants (i.e. if *cooperating rules* are allowed), then universality is achieved also for the basic form of P systems.

We also study the problem of efficient simulation of some kinds of Spatial P system models. In particular, the peculiar feature of mutually-exclusive objects, together with priorities among evolution rules, makes it difficult to devise an efficient simulation algorithm. We present various algorithms for the simulation of restricted models, and study their complexity.

We exemplify the use of the formalism with a model of the evolution of *ring species*, describing the spreading of a population around a geographical barrier, which uses a high-level view of the spatial locations in which the populations spread. Moreover, we show that the formalisms can be used as a lower-level tool, abstracting the continuous space, by means of a model which simulates the schooling behaviour of herrings.

Finally, we present the *Spatial Calculus of Looping Sequences* (Spatial CLS), an extension of the Calculus of Looping Sequences (CLS) based on continuous space. CLS [15, 57, 10, 4] is a calculus, based on term rewriting, developed specifically for the modelling of biological systems. Thus, a CLS model is composed of a *term*, which describes the structure and elements of a biological system, and a set of *rewrite rules* describing the possible evolutions. The fundamental concept of CLS is that of *sequence*, which comes in two forms: *simple sequences*, which are used to represent simple entities of biological systems, such as proteins and DNA strands, and *looping sequences*, which are meant to represent closed circular sequences of elements. Looping sequences, together with a containment operator provided by the calculus, allow a direct modelling of membranes and of the biological elements which are embedded in them. Therefore, membranes and elements inside them can be directly modelled in the syntax.

Spatial CLS extends CLS by allowing to keep track of the position of biological elements in a 2D or 3D continuous space, as time passes. The movement of elements in the space can be precisely described by means of *movement functions* associated with them. The application of rewrite rules to system elements can be constrained to the positions of the involved elements, in such a way, for example, to allow the interaction between two elements only when they are close enough.

An important feature of Spatial CLS is the ability to describe the space occupied by elements and membranes. Moreover, the semantics of the calculus ensures that the space occupied by different objects is always kept disjoint, namely different objects cannot overlap, and each object must be correctly contained within the bounds of its containing membrane, if any. In order to model specific behaviours, the modeller can provide different algorithms to rearrange the position of objects in case of a space conflict. We propose an algorithm for the resolution of space conflicts which is based on the assumption that conflicting objects push each other when they are too close. In order to increase the modelling power of the calculus, stochasticity is included in Spatial CLS by means of stochastic rates associated with rewrite rules.

As example applications of the calculus, we present a model of the development of a

biological tissue, which arises from the proliferation of cells performing the mitosis cycle. This model takes advantage of the possibility of representing the space occupied by the elements, in this case the cells, and demonstrates the use of a rearrangement algorithm. A model of the quorum sensing process performed by the *Pseudomonas aeruginosa* bacterium is also presented. In this model, bacteria diffuse small molecules in the environment, which enable them to determine their concentration in a colony of bacteria. In the presented model we keep track of the positions of the molecules diffusing in the environment, which is useful to obtain faithful simulations of the system behaviour.

1.3 Structure of the thesis

In Chapter 2 we survey some formalisms for the formal description and analysis of biological and ecological systems. In subsequent chapters, from the formalisms with the most abstract notion of space, to the most concrete, we formally define the *MIM Calculus*, the *Spatial P systems*, and the *Spatial CLS*.

In particular, the *MIM calculus*, and its extension for compartmental modelling are presented in Chapter 3. Then, in Chapter 4, we formally define the semantics of *Spatial P systems*, and study their computational power. In Chapter 5 we present the algorithms for the simulations of some restricted kinds of Spatial P system models, and prove their correctness with respect to the semantics. Lastly, in Chapter 6 we present the *Spatial Calculus of Looping Sequences*, its formal syntax and semantics, and an algorithm for the resolution of space conflicts.

For each formalism that we define, we develop models of real-world systems in order to exemplify their use and demonstrate their practical usefulness. Finally, in Chapter 7 we draw some conclusions on the work presented in the thesis, and discuss possible future works.

1.4 Published material

Part of the material presented in the thesis has been either published or submitted for publication, as detailed in the following.

- The definition of the basic version of the MIM calculus, from Chapter 3, appeared in [8].
- An earlier version of Spatial P systems, from Chapter 4, has been published in [9].
- The current definition of Spatial CLS, as presented in Chapter 6, has been published in [7]. An earlier version appeared in [6].

Chapter 2

Related works

In this chapter we survey some formalisms which have been used or proposed as modelling tools for biological and ecological systems. We start our survey from the π-calculus, and exemplify its use for the description of biomolecular processes as proposed in [74, 75, 67, 72].

2.1 The π-calculus for the modelling of biological systems

The π-calculus [58] is a process algebra originally developed for describing concurrent processes, that are able to communicate by using channels. The key feature of the calculus is its ability to represent *mobility*, i.e. the movement of an agent to a different location in a virtual space of processes. In the π-calculus, the concept of location is implicit, and is determined by the collection of links to other processes. In this view, a change of the location is represented by a change in the neighbours of a process, i.e. of the processes to which it can talk to by exchanging messages through the links.

An important notion of the π-calculus is that of *names*, which represent both the names of the links and the data which can be communicated (i.e. transmitted along a link). The set of names is denoted by $\mathcal{N}$; its metavariables are $a, b, \ldots, x, y, z$. π-calculus *processes* and *prefixes* are defined by the following grammar:

$$\begin{array}{lcl} P & ::= & \mathbf{0} \;\mid\; \alpha.P \;\mid\; (\mathbf{new}\ x)P \;\mid\; P_1 + P_2 \;\mid\; P_1 \mid P_2 \;\mid\; A(x_1, \ldots x_n) \\ \alpha & ::= & \overline{x}\langle y\rangle \;\mid\; x(y) \;\mid\; \tau \end{array}$$

$\mathbf{0}$ is the nil process, which cannot do anything. A process $\alpha.P$ can do an *action* α and then behaves as P. There are three kinds of actions. An *output action* $\overline{y}\langle x\rangle.P$ means that the name x has to be sent along channel y, and then the process behaves as P. An *input action* $y(x).P$, instead, means that a arbitrary name z can be received on channel y, and then the process behaves as P where each free occurrence of x is replaced by the received name z. An input prefix $y(x).P$ binds name x in P. Finally, τ represents the *hidden action*, meaning that the process can perform an unobservable (internal) action.

The construct $(\mathbf{new}\ x)P$ is used to declare *private names*. It binds the name x in P, and means that x is a private name P, i.e. it is different from any other name x appearing outside. Operator $+$ denotes *choice*, meaning that the process can behave either as P_1 or as P_2. Operator $\mid$ denotes the *parallel composition* of processes. Finally, given a set of *process identifiers* $\mathcal{K}$ ($A, B, \ldots \in \mathcal{K}$) and process definitions of the form $A(y_1, \ldots, y_n) = P$, the

construct $A(x_1, \ldots, x_n)$ can be used to refer to the process P associated with the identifier A. In this case, variables $y_1, \ldots, y_n$ in P are to be replaced by the actual names $x_1, \ldots, x_n$.

According to the semantics of the calculus, a process can evolve by performing actions. In particular, two processes can *synchronize* if they are able to perform two complementary actions (an output and an input) on the same channel. This models a communication between processes. Since channels and data are the same entity (they are names), it is possible to send (the name of) a channel to another process, thus enabling the receiving process to communicate through the channel. This models mobility, interpreted as a change in the interaction network of processes.

The basic idea used to represent biochemical processes using the π-calculus, proposed in [74, 75], is to represent molecules (and their domains) as processes, where chemical interaction corresponds to communication. Since changes in the structure of molecules determine a modification of their interaction capabilities, this is naturally modelled by using mobility.

Example 2.1.1. To highlight the use of the π-calculus for modelling biochemical processes, we report a part of the model of RTK-MAPK signal transduction pathway from [75]. A *pathway* describes a sequence of interactions among molecules. In this particular case, we consider a *signal transduction* pathway, namely a process by which an external stimulus, received by a *receptor* embedded in cell membrane, triggers a chain of reactions which, in the end, results in some internal effect. In the RTK-MAPK pathway, the receptor RTK receives an external stimulus which, in the end, causes the *activation* of the MAPK enzyme. The *enzyme* is a protein which is able to increase the rate of other reactions. Such a modification of the internal behaviour constitutes the effect in response to the external stimulus.

The pathway itself is modelled as the parallel composition of all the processes representing molecules. In the following, π-calculus names are written in lower case, while process identifiers are upper case. The pathway is modelled as a process formed by the parallel composition of all the molecules present.

$$RTK\_MAPK\_pathway = Free\_ligand \mid RTK \mid Ras \mid \cdots$$

In this case, for example, there are a free *ligand*, a receptor tyrosine kinase (RTK) molecule, and a Ras protein (the other molecules are omitted). A ligand is a biological element which is able to bind to the receptor.

The following process definitions allow us to illustrate some important features.

$$\begin{aligned} Free\_extracellular\_domain &::= ligand\_binding.rtk\_binding.P' + \\ &\qquad antagonist\_binding.\mathbf{0} \\ Ligand &::= \overline{ligand\_binding}.Bound\_ligand \\ Antagonist &::= \overline{antagonist\_binding}.Bound\_antagonist \end{aligned}$$

The receptor has an *extracellular domain* to which the ligand can bind to. The process $Free\_extracellular\_domain$ models the free extracellular domain, namely when the ligand is not bound to it. The extracellular domain can interact through either channel $ligand\_binding$, with the ligand, or channel $antagonist\_binding$, with the antagonist. This models mutually exclusive interactions, in which the ligand and the antagonist compete to

bind to the same domain. We can see that a ligand can become the process *Bound_ligand* (defined elsewhere), representing the bound ligand and its interaction capabilities. Similarly, the antagonist can become a *Bound_antagonist*.

It is interesting to note that if the extracellular domain binds to the ligand, then it can still perform other interactions, i.e. through channel *rtk_binding*, while if it binds to the antagonist, then it becomes the nil process **0**, which is not able to perform any other interaction. The ability of an antagonist to unbind from the extracellular domain is modelled inside process *Bound_antagonist*.

The ability to send and receive channel names is used to model modifications of molecules. For example, the following definition shows an active kinase and a binding domain for it.

$$\begin{aligned} Active\_kinase &::= \overline{phosph\_site}\langle p\_tyr\rangle.P'' \\ Binding\_domain &::= phosph\_site(tyr).tyr.P''' \end{aligned}$$

When they interact, the name *p_tyr* is sent to the binding domain, thus enabling it to interact through the channel *p_tyr* afterwards.

Lastly, the operator of scope restriction for names is used for modelling *complexes*, that is two or more molecule bound together. A complex is modelled by processes sharing a private name, thus the creation of a complex is obtained by exchanging private names. Intuitively, when different molecules are bound together, they can interact with each other through some private channel name, while external molecules cannot interact with them through such a private name. Such an abstraction is also used to model *compartments*, therefore the compartment containing a process is entailed implicitly by the (private) channels that the process can access.

For example, in the following definition:

$$\begin{aligned} Free\_ligand &::= (\textbf{new}\ backbone) \\ &\qquad (Free\_binding\_domain \mid Free\_binding\_domain) \\ Free\_binding\_domain &::= \overline{ligand\_binding}\langle backbone\rangle.Bound\_ligand \\ Extracellular\_domain &::= ligand\_binding(cross\_backbone). \\ &\qquad Bound\_Extracellular\_domain \end{aligned}$$

the ligand has two free domains identified by channel *ligand_binding*, that can be bound to the extracellular domains. When a free domain of the ligand binds to an extracellular domain, the scope of the private name *backbone* is extruded to also include process *Bound_Extracellular_domain*. Once both domains of the ligand are bound, the following process is obtained:

$$\begin{aligned} Free\_ligand ::= (\textbf{new}\ backbone)(&Bound\_ligand \mid Bound\_ligand \mid \\ &Bound\_Extracellular\_domain \mid Bound\_Extracellular\_domain) \end{aligned}$$

where the sharing of the private name *backbone* among the processes represents the formation of the complex.

These examples show that a π-calculus model of a biological system, while allowing direct representation of molecules as processes, and precise descriptions of their behaviours,

can be complicated and hard to read. Moreover, complex encodings may be needed for modelling some complex behaviours. However, for constructing a model, one should try to exploit the compositionality of the calculus, which can help to handle this complexity.

The basic π-calculus allows analysing only qualitative properties of systems. In order to also enable the analysis of quantitative properties, an extension of the π-calculus *for biomolecular processes* has been proposed in [67]. It is a variant of the *Stochastic π-calculus* [65], where rates of reactions are computed according to their *basal rates* and the concentrations (quantities) of reactants. In particular, as with other stochastic formalisms, for each reaction (represented by an action in the π-calculus syntax) in which a certain species can take part there is a *rate coefficient* describing how the its concentration changes when the reaction occurs. Such a rate corresponds to the parameter of a negative exponential random variable modelling the duration of the activity. The behaviour is driven by a *race condition*, where all the enabled activities compete for proceeding, but only the fastest one succeeds. Usually, the semantics of a stochastic calculus is given by a Continuous Time Markov Chains (CTMC), which can be used either for stochastic simulation, or to perform different kinds of analysis, such as (probabilistic) model checking.

Other calculi based on π-calculus are *Bioambients* [73, 21], *Beta Binders* [66, 34], and π@ [85, 87, 86]. As regards extensions of the π-calculus providing more concrete representation of the space, we cite *SpacePI* [46], and the 3π process algebra [24], which embed processes in a continuous space.

2.2 Spatial modelling

The spatial features of biological systems can be studied by using many different means. The most abstract spatial feature, provided by almost all computer science formalisms for biology, regards the modelling of compartments. We survey various formalisms which provide some form of compartmental modelling, then we present other formalisms based on more concrete notions of space.

2.2.1 Compartmental modelling

Compartments are closed areas which may contain elements and other compartments. They may be either entailed by biological membranes or simply used to represent different abstract locations in a spatially heterogeneous environment. As in the case of biological membranes, biological entities (such as molecules) can, in some cases, cross compartment bounds. More complex calculi also allow describing reactions occurring on the surface of membranes.

Bio-PEPA [31, 29, 30] is an extension of the PEPA process algebra, a stochastic process algebra originally developed for performance analysis of computer systems [43]. Bio-PEPA uses a different, more abstract, view of biological systems than the one used in the π-calculus. Instead of representing molecules as processes, each Bio-PEPA process models a *species*. It allows the use of general kinetic laws, namely functions for deriving the rate of reactions from varying parameters, such as the rate coefficients and concentration of reactants, or the size of compartments. In Bio-PEPA compartments are static and not represented explicitly in the syntax, therefore a species which can appear in two different

compartments is represented by two different species components in the model, having different names.

The ability to directly describe compartments is quite common among computer science formalisms for biology, since membranes often play an important role in many biological processes. The widespread availability, among calculi for biology, of operators for the direct description of compartments shows the usefulness of this feature.

For example, Brane Calculi [23] are process calculi devoted to the modelling of interactions of biological membranes, which are active entities driving the evolution of the system. Membranes can be nested, and their structure can change dynamically as the result of the *actions* performed. The different calculi proposed differ form one another in the kind of actions which can be associated with membranes. For instance, the simplest of such calculi, the PEP calculus, models the processes of *endocytosis*, where an internal membrane is created by capturing some other external material, and *exocytosis*, which is the reverse process of the endocytosis. The PEP calculus provides the following three kinds of actions: *phagocytosis*, in which a membrane engulfs one external membrane; *pinocytosis*, in which a internal membrane is created (*vesicle*) without engulfing any membrane; and *exocytosis*, in which the content of an internal membrane is released outside.

Another calculus for biological modelling providing a notion of compartments is BioAmbients [73], which is inspired by Mobile Ambients [25]. Mobile Ambients is a process calculus underlain by many π-calculus concepts, which allows the explicit representation of movement of processes through *ambients*, intended as abstract places in which computation is performed. Moreover, ambients can be nested, and can move themselves from a place to another. Compartments can be emulated in calculi which do not allow a direct representation of them, as it is the case, for example, of the π-calculus. The BioAmbients calculus has been developed to address such a problem of compartmental modelling in the π-calculus. In fact, as we have seen, compartmental modelling in the π-calculus is based on the scope restriction operator, and does not provide a direct representation of compartments in the syntax. Therefore complex encodings would be required to deal with them, which may hamper the readability of models [75, 73].

A different approach to compartmental modelling is the one followed by Beta Binders [66], which provides a notion of compartment by means of *boxes* enclosing π-calculus processes. Processes in boxes are able to communicate with the external environment through specific *interaction sites*. Boxes and their interaction sites can dynamically change. For example, there are operators for *hiding* (thus preventing it to be used for an interaction) and *unhiding* interaction sites, and for joining and splitting boxes. The calculus does not permit the nesting of boxes. An extension of Beta Binders for the modelling of static nested compartments, and of the movement of objects across them, is presented in [42].

The π@ calculus [85, 87] is an extension of the π-calculus, developed as a low-level language for the implementation and comparison of calculi for the compartmental modelling of biological systems. The π@ calculus can be used to formalize multi-compartment systems with a dynamic structure. The syntax of the calculus does not provide the ability to directly describe compartments (membranes), which are instead associated with channel names. In the stochastic extension Sπ@, proposed in [86], the volume of membranes can change as result of interactions, and their sizes are taken into account for the simulation, performed by a specialized (multi-compartmental) version of Gillespie stochastic simulation algorithm.

The *attributed π-calculus* [48] provides the ability to associate *attributes* to processes, and to constraint process communication (synchronization) on the basis of the values of the attributes. The calculus is parametric with respect to the functional language used to represent values and constraints. Static compartments can be easily modelled in the calculus, by using attributes to denote the abstract location in which the process is present. The attributed π-calculus has been extended *with priorities* in [49]. Such an extension has been shown to be also able to model dynamic compartments, namely compartments which can change their nesting structure dynamically, by means of an encoding of the $\pi@$ calculus.

A different approach to the modelling of dynamic compartments is proposed in the Imperative π-calculus [47], which is another extension of the attributed π-calculus. The Imperative π-calculus extends the attributed π-calculus with a global store. The language embedded by the calculus provides operations for accessing the global store, namely for reading and modifying the values of variables in the global store. By representing compartments with different names, and associating channel names with variables in the global store, it is possible to keep track of the volume of compartments, and use their values to compute reaction rates.

In the field of ecological modelling, PALPS [36] has been proposed as a domain-specific process calculus for the modelling of population systems. In PALPS, each individual is modelled by a process, each possessing a *species* and a *location*. Locations are related by means of a neighbourhood relation, and individuals can either move non-deterministically through nearby locations, or be involved in other actions, such as preying. The behaviour of individuals can be determined by local conditions, for example by taking into account the number of individuals present in a location.

Apart from process calculi, the Calculus of Looping Sequences (CLS) [15, 57, 4] is a formalism based on term rewriting which allows a direct representation of membranes in the syntax of the calculus. The evolution of a biological system is represented by means of rewrite rules, which can be used to model both biochemical reactions and more complex behaviours, such as rearrangements in the membrane structure of the system. For example, the creation, dissolution and fusion of membranes can be easily expressed in a CLS model.

In the field of Natural Computing, P systems [69, 70, 68] (also known as *membrane systems*) have been defined as a computational formalism based on the description of interactions inside membranes. A P system is composed of a hierarchy of membranes, each of them containing a multiset of *objects*, which are processed by evolution rules. Evolution rules allow the description of the behaviour of a model, for example by representing chemical reactions. A set of rules is associated with each membrane, which are to be applied to the objects contained in the membrane itself, and can transform objects in different objects, and send them into other membranes. In the basic class of P systems, membranes are static and cannot be created nor destroyed. Many extensions of the basic formalism have been developed [70, 68]. Moreover, in spite of being initially defined as a computing formalism inspired by biological behaviour, they have also been applied to the modelling of biological systems [71]. As regards quantitative modelling, which is particularly important for obtaining executable models of systems, various probabilistic/stochastic extensions of P systems have been proposed [64, 18, 28, 27, 78, 39].

Metabolic P systems [55, 54] are a deterministic variant of P systems, developed to model the evolution of chemical substances driven by the laws of biochemistry. Differently from basic P systems, metabolic P systems use a more abstract view of the system, in which

different substances, representing populations of identical molecules, are transformed by some reactions. In particular, the variation in time of the concentration of each substance involved in a certain reaction is described by a flux regulation map, giving the amount of substances which are consumed/produced by the reaction in the current state. Such a transformation may depend on the concentration of the substances in the system, and by parameters describing environmental factors such as temperature, pressure, pH, etc.

P systems have also been applied in the field of Ecology [18, 28, 27]. The extension of P systems proposed in [18], based on a probabilistic extension of P systems introduced in [64], is used to describe the behaviour of *metapopulations*. Metapopulations are local populations living in spatially separated areas (called *patches*), where populations can interact, and individuals can disperse from a patch to nearby patches. Models for metapopulations aim at discovering how the fragmented habitat influences local and global population behaviour. In the proposed model, objects are used to model different species (predators and preys), and patches are represented as elementary membranes in a flat membrane structure. Patches form the nodes of an undirected weighted graph, with edges describing a neighbourhood relation between patches, modelling spatial proximity. Costs associated with edges model the effort of individuals for migrating from a patch to another.

Another extension of P systems, called *multienvironment P systems* [78], has been applied to the modelling of ecosystems [26, 33]. A multienvironment P system model is composed of a set of *environments*, each containing a P system, from a particular class of probabilistic P systems. All the P systems in environments have the same *skeleton*, namely they use the same alphabet of symbols, membrane structure, and sets of evolution rules. Objects in an environment, i.e. outside the skin membrane of a P system, can move from an environment to another by means of the application of special communication rules. The movement of objects to other environments, as well as the internal behaviour of P systems, is driven by time-varying probabilistic values associated with the rules.

In *Tissue-like P systems* [56, 37] the membrane hierarchy is replaced by a graph whose nodes are elementary membranes, called *cells*, and the edges describe the communications channels between cells. Such a device is useful to describe systems composed of many entities living together, such as in tissues, organs and organisms. The structure of graph models the protein channels among nearby membranes, through which inter-cellular communication can happen. Tissue-like P systems have been extended to *Population P systems* [17], which provide mechanisms to dynamically modify the structure of the communication graph among cells.

Lattice Population P systems [79, 77] are another variant of population P systems in which space is represented as a finite regular lattice where each cell of the lattice contains a distinct stochastic P system describing the behaviour of an individual, such as a bacterium. Translocation rules describe the possible movements of objects from a cell to another, thus enabling different P systems to communicate by sending and receiving such objects.

2.2.2 Concrete spatial modelling

In order to faithfully describe some biological systems, less abstract representations of space than the one provided by compartments, are needed. Beyond computer science formalisms, one of the earliest approaches in this direction involved the use of differential equations models, which allow the description of many different spatially-aware biological processes, such as *reaction diffusion* systems. This kind of models and their extensions

has been used, for example, as the basis for models of insect dispersal and chemotaxis [59]. Other kinds of spatial models involve spatial pattern formation [59, 60]. Since these models are based on continuous variables, they may not be adequate to describe discrete biological entities.

In computer science, a formalism including spatial features are the Cellular automata (CA) [61], which has been initially devised as a computational formalisms inspired by biological behaviours. A cellular automaton is composed of a finite grid of *cells*, where each cell has an associated state taken from a finite set of different states. Time is discrete, and at each step of the evolution all the cells change state in accordance with a *rule*, which is characteristic of the particular cellular automaton model. The rule is deterministic and "local", in the sense that the new state of a cell is determined only on the basis of the previous states of cell itself and of nearby cells. Many variants of cellular automata have been defined.

CAs have been developed as a computational tool inspired by biological behaviours. However, even if their focus has not been in the modelling of biological systems, they have been later used for this purpose. In this view, CAs are particularly suitable for describing the evolution of populations of many similar entities, whose behaviour is based on local interactions. For example, they have been used to describe tumour growth [63], showing how a simple rule driving the evolution can lead to a complex behaviour of the whole population. Other extensions of cellular automata, for example including stochasticity, have also been used [63].

A formalism developed for the modelling of biological systems, which provides a more concrete representation of the space, is SpacePI [46], an extension of the π-calculus with (continuous) space and time. In SpacePI, positions in a continuous space, such as $\mathbb{R}^2$, are associated with processes, and processes can move autonomously according to a movement function. In SpacePI, processes move uniformly, namely on a constant direction with constant speed, during fixed time intervals, at the end of which they can change direction and speed according to their movement function. The calculus is deterministic and processes may, and are required to, communicate only when they are close enough, with respect to their euclidean distance. In particular, an *interaction radius* is associated with each action and co-action. Two processes communicate as soon as the sum of the interaction radii of their complementary actions are equal to the distance between processes. SpacePI does not allow the direct modelling of compartments and membranes.

Another extension of the π-calculus with spatial features is the 3π process algebra [24], where processes are embedded in a 3D space. The calculus is based on affine geometry, and processes can interact by exchanging channel names or geometric data. The movement of a process is realized by applying a frame shift operation (described by an affine map) to the process itself. A peculiarity of the calculus is that a process has no visibility of its location in the global space, but can nevertheless communicate its position to other processes which can be used, for example, to compute the distance between two processes. Membranes and compartments cannot be described explicitly.

Chapter 3

A process calculus for Molecular Interaction Maps

In this chapter, we present the formal definition of the *MIM calculus*, a modelling formalism with a strong biological basis, which provides biologically-meaningful operators for representing the interaction capabilities of molecular species. The operators of the calculus are inspired by the reaction symbols used in Molecular Interaction Maps (MIMs) [52], which are a graphical language for describing bioregulatory networks used in biology. MIM diagrams are composed of nodes, representing molecular species, and edges connecting nodes, which represent the possible reactions among species. Edges can express different kinds of reactions, according to the reaction symbol used.

The basic version of the MIM calculus does not provide any feature for spatial modelling. In order to improve its usefulness, we also present an extended version of the calculus, namely the *MIM calculus with compartments* (*MIMc-co*) which provides the ability to denote "abstract" locations for the elements.

The MIM calculus (MIMc) is defined in the style of *process calculi*, where each molecule appearing in the system is described by a term. However, unlike most of the previously proposed calculi for describing biological processes, which model reactions by means of process communication, MIMc provides high-level operators with a direct biological meaning. For example, there are operators for expressing the creation of a bond between two compounds (*complexation*), and other biologically interesting events. We provide both a basic version of the calculus, without any operator for space modelling, and an extension allowing the description of *compartments*, thus the ability to indicate the abstract location where an element occurs.

The calculus has a strong relationship with Molecular Interaction Maps. The presented approach has a twofold advantage. On the one hand, we can exploit the features of process calculi such as incremental definition of models, techniques for analysis and verification of properties, and easy development of simulators. On the other hand, the correspondence of the operators of the calculus with biological interactions allows an immediate translation of Molecular Interaction Maps into the MIM calculus. Less immediate translations of Molecular Interaction Maps into more general formalisms can be found in [5, 19, 32].

Since the MIM calculus is based on Molecular Interaction Maps, we introduce the MIM notation first. Then we give a formal definition of the syntax and semantics of the MIM calculus, and we study properties of the formalism. Finally, the extension of the MIM

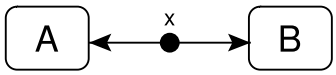


Figure 3.1: An example of MIM diagram.

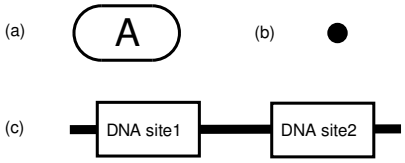


Figure 3.2: Species in MIMs.

calculus for compartment modelling is presented. Two case studies are also presented to show the use of the two versions of the calculus for modelling biomolecular networks.

3.1 Molecular Interaction Maps

MIM diagrams provide a static view of the molecular species in a system, and their possible interactions. Interactions are represented by lines connecting nodes representing species, and the meaning of each interaction depends on the symbol used to draw the line. Each molecular species usually appears only once in a diagram. Moreover, since the diagram is static, it does not contain any information about number of molecules (concentration) of the molecular species.

Three classes of molecular species can be represented: *elementary species* (fig. 3.2a), *complex species* (fig. 3.2b) and *DNA sites* (fig. 3.2c). Complex species represent either a combination of elementary species or a modified elementary species. Figure 3.1 shows a simple MIM diagram, containing the elementary species A and B which can interact. A named elementary species is drawn as a rounded box, containing its name. A complex molecular species, resulting from an interaction, is depicted as a small filled circle on the corresponding interaction line. For instance, in Figure 3.1, the complex species obtained by the binding of A and B is represented by the node x on the interaction line. To avoid cluttering the diagram, an interaction line may contain more than one of such nodes, all denoting the same complex species resulting from the depicted interaction.

MIM diagrams allow representing two kinds of interactions: *reactions*, which act on molecular species, and *contingencies*, which act on reactions or other contingencies. An interaction symbol represents a possible interaction that can happen if certain conditions on the state hold. Interactions can have a kinetic constant k associated with them, that describes its rate. Conceptually, a higher kinetic constant means that the interaction is more likely to happen than an interaction with a lower kinetic constant.

For defining the MIM calculus we consider the reaction symbols shown in Figure 3.3. Note that they are only a subset of the reaction symbols available for use in a MIM diagram.

- *Non-covalent binding* (Figure 3.3a): denotes the reversible binding of the two pointed species: a molecule of the first species can bind to a molecule of the second species, forming a compound. Two species joined by means of a non-covalent bond can eventually dissociate autonomously.

- *Covalent modification* (Figure 3.3b): denotes the covalent modification of the pointed species; the modification type (such as phosphorylation or acetylation) is written at the tail.

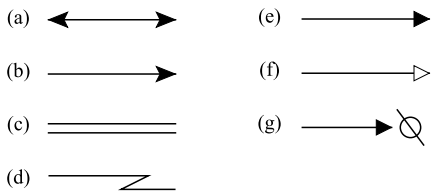


Figure 3.3: Reaction symbols.

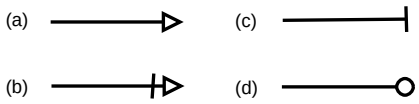


Figure 3.4: Contingency symbols.

- *Covalent binding* (Figure 3.3c): denotes the capability of the two connected species to form a covalent bond that, differently from the non-covalent bond, cannot dissociate autonomously but instead needs another molecular species which is able to break such a bond.
- *Cleavage of a covalent bond* (Figure 3.3d): denotes the possibility of a covalent bond at the head (right end) to be broken by the presence of the species at the tail (left end). This symbol points from a species to a reaction symbol representing covalent binding.
- *Stoichiometric conversion* (Figure 3.3e): denotes the conversion of the species at the tail of the arrow, called reactant, into a corresponding number of product species, i.e. the species written at the tail of the arrow disappears, while the pointed ones appear.
- *Lossless production* (Figure 3.3f): it is similar to the stoichiometric conversion, but without the loss of the reacting species.
- *Degradation* (Figure 3.3g): means that molecules of the species can disappear.

The following contingency symbols, shown in Figure 3.4, are provided by MIM diagrams:

- *Stimulation* (Figure 3.4a): means that the molecule of the species at the left end stimulates the pointed reaction;
- *Requirement* (Figure 3.4b): means that the molecule of the species at the left end is required in order for the pointed reaction to happen;
- *Inhibition* (Figure 3.4c): the presence of the species at the tail (left end) inhibits the possibility for the pointed interaction to happen;
- *Catalysis* (Figure 3.4d): means that the pointed reaction has a much higher reaction rate if the species is present than if it is not.

Since we will provide a qualitative definition of the MIM calculus, we will not deal with the contingencies of stimulation and catalysis, whose effect is to change the rate of the pointed reactions. The MIM calculus will allow describing, for each interaction, which species are needed for the reaction (requirements), and which species are inhibitors for the reaction.

A MIM diagram can be interpreted in different ways, namely the *explicit* interpretation, the *heuristic* interpretation, and the *combinatorial* interpretation ([52, 51]). Whenever a

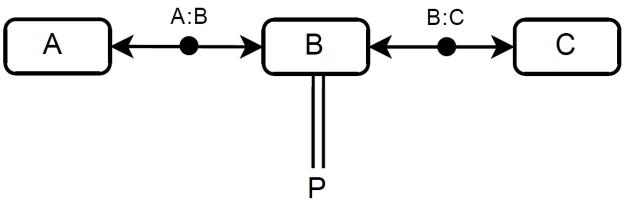


Figure 3.5: An example of MIM diagram.

MIM diagram is presented, it is necessary to specify which kind of interpretation must be used.

In the explicit interpretation, the interaction lines which are depicted represent the all and only possible interactions among elements. For example, consider the MIM diagram shown in Figure 3.5, in which a molecule B can either create a complex with an A, or create a complex a C, or be phosphorilated. In the explicit interpretation, the possible complex species which can be obtained are $A : B$, $B : C$, and pB. In particular, for example, it is not possible for the B molecule in the complex $B : C$ to become phosporilated, yielding $pB : C$, since such an interaction is not depicted. The same is for the complex $A : B : C$.

On the other hand, in the combinatorial interpretation each interaction line conceptually represents the interaction among some *sites* (or *domains*) of the involved elements, therefore an interaction on one site does not automatically preclude the possiblity of other sites to interact with other elements. Therefore, by using the combinatorial interpretation, the MIM diagram of Figure 3.5 means that, apart the species $A : B$, $B : C$, $\overline{pB}$, also the following species are possible: $A : B : C$, $A : pB$, $pB : C$ and $A : pB : C$. An advantage of the combinatorial interpretation is that a few depicted interactions can stand for a much greater number of explicit interactions.

Finally, the heuristic interpretation does not specify which complex species are possible, beyond those that are explicitly depicted. Therefore it is an intermediate interpretation between the explicit interpretation and the combinatorial one. Actually, a heuristic map needs some additional information, such as a textual description, to clarify which are the possible interactions. For this reason, a heuristic map, in constrast to the other interpretations, cannot be used directly as the input for a simulator. Nevertheless, they are often used, since they allow representing what is currently known about some system, leaving unspecified the interactions which are not known.

3.2 The MIM Calculus

In this section we formally introduce the syntax and semantics of the MIM calculus. MIM calculus is defined in the style of process calculi, where an agent represents a molecule of a certain named species. Names $A, B, C, \ldots$ are used to identify the different elementary species, and we denote by $\mathcal{E}$ the set of names of elementary species. We also assume a set $\mathcal{E}_c$ whose elements denote types of covalent modifications (such as phosphorylations).

Definition 3.2.1 (Syntax)**.** *Processes* P, *named species* S and capabilities μ of the *MIM*

calculus are defined by the following grammar:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \;\mid\; S \;\mid\; P \mid P \\ S & ::= & \mu.IS \\ IS & ::= & A \;\mid\; S:S \;\mid\; \overline{qS} \;\mid\; \overline{SS} \\ \mu & ::= & rec\ x.\mu \;\mid\; M \;\mid\; x \\ M & ::= & \varnothing \;\mid\; M+M \;\mid\; \gamma \end{array}$$

where simple interaction capabilities γ are defined as follows:

$$\begin{array}{rcll} \gamma & ::= & (\nu,\iota) \xrightarrow{N} \mu & \textit{(non-covalent binding)} \\ & \mid & (\nu,\iota) \overset{N}{=\!=} \mu & \textit{(covalent binding)} \\ & \mid & (\nu,\iota) \overset{q}{\Longrightarrow} \mu & \textit{(covalent modification)} \\ & \mid & (\nu,\iota) \overset{N}{\rightsquigarrow} & \textit{(cleavage)} \\ & \mid & (\nu,\iota) \longrightarrow P & \textit{(conversion)} \\ & \mid & (\nu,\iota) \longrightarrow\!\!\triangleright\, P & \textit{(lossless production)} \end{array}$$

where $\mathbf{0}$ is the empty process, $A \in \mathcal{E}$ denotes an elementary species, $q \in \mathcal{E}_c$ denotes the type of modification, $x \in \mathcal{X}$ is a variable. Moreover, N denotes a *species name*, namely a term in which each capability μ occurring in it is empty ($\mu = \varnothing$). The set of all species name, formally defined in the following, is denoted by $\mathcal{N}$, thus $N \in \mathcal{N}$. Finally, $\nu, \iota \subset \mathcal{N}$ are sets of species names, denoting *promoters* and *inhibitors*, respectively.

Terms P of the calculus are made of a composition of molecules S, by means of the parallel operator $\_ \mid \_$. Each molecule is of the form $\mu.IS$, where IS describes the structure of the molecule, and μ describes its interaction capabilities. In particular, IS denotes either an elementary molecule of species A, or a compound molecule. In the case of compound molecules, IS is made of the single molecules forming the compound, combined by means of different syntactical operators specifying the kind of bond that keeps the molecules together: either a *non-covalent bond* $S_1 : S_2$ between the species S_1 and S_2, or a *covalent modification* $\overline{qS}$ of species S, or a *covalent bond* $\overline{S_1 S_2}$ between S_1 and S_2. Note that the capabilities of each molecule forming a compound are retained in the compound description.

For example, term $\{\gamma\}.A$ models a molecule of species A, having a single interaction capability γ.[1] A complex formed of two simple molecules A and B is instead represented as $\mu_1.(\mu_2.A : \mu_3.B)$, where μ_1 are the capabilities of the compound, and μ_2, μ_3 are the capabilities of molecules A and B, respectively.

We denote the set of species by $\mathcal{S}$, and identify $\mathcal{N} \subset \mathcal{S}$ as its subset of named species without capabilities, i.e. where each μ is empty ($\mu = \varnothing$). We assume a function $\lfloor . \rfloor : \mathcal{S} \to \mathcal{N}$ that strips all the capabilities from a named species $S \in \mathcal{S}$. For example, $\lfloor \mu_1.(\mu_2.A : \mu_3.B) \rfloor = \varnothing.(\varnothing.A : \varnothing.B)$. Moreover, we often avoid writing empty capabilities when no ambiguities arise, therefore we simply write $A : B$ instead of $\varnothing.(\varnothing.A : \varnothing.B)$. This function is extended to processes $\lfloor . \rfloor : \mathcal{P} \to \mathscr{P}(\mathcal{N})$ as $\lfloor S_1 \mid \cdots \mid S_n \rfloor = \{\lfloor S_1 \rfloor, \ldots, \lfloor S_n \rfloor\}$.

[1] For the sake of readability, we have enclosed capabilities in curly brackets. From now on, we shall systematically use curly brackets around capabilities, as in the example.

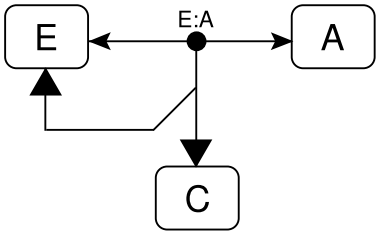


Figure 3.6: An example of recursive MIM

The calculus allows expressing different capabilities for molecules. Operator $\xrightarrow{N}\mu$ means that a species can form a non-covalent bond with a molecule of species name N. The result will be a compound of the form $\_:\_$ made of the two involved species, and with capabilities μ. Similarly, operator $\overset{N}{=\!=}\mu$ means that a species can form a covalent bond with a N, resulting in a compound of the form $\overline{S_1\ S_2}$ with capabilities μ. The operator for covalent modification $\overset{q}{\Longrightarrow}\mu$, similarly produces a compound $\overline{qS}$ with capabilities μ. The operator for cleavage $\overset{N}{\rightsquigarrow}$ means that a molecule can break the covalent bond specified by N, where N has to be either of the form of $\overline{N_1 N_2}$ or $\overline{qN}$. Finally, there are the operators $\longrightarrow P$, for expressing a conversion of a molecule into other molecules, and $\longrightarrow\!\triangleright P$ for a lossless production of molecules. In both cases, the resulting molecules are represented by a process P.

We allow recursive definitions of capabilities, by means of the *recursion* operator *rec*. As always, $rec\ x.\mu$ binds the free occurrences of the variable name x in μ. We assume a *substitution* function $\mu[\mu'/x]$ for replacing each free occurrence of x in μ with μ'. The substitution function is also extended to processes. We use the notation $rec\ \tilde{x}.\mu$ with $\tilde{x} = x_1, \ldots, x_n \in Var^*$ as an abbreviation for $rec\ x_1.\cdots .rec\ x_n.\mu$.

Names ν, ι are used to express contingencies on the application of an operator, depending on the species appearing in the environment. The former, ν, expresses the species that *must* be present *(promoters)*, while the latter, ι, expresses those that must be *absent (inhibitors)*. We omit writing contigencies when they are empty.

To give an example of a term in which recursive capabilities are used, let us consider a system in which substrate A is transformed into product C by the enzyme E. The MIM diagram in Figure 3.6 shows that enzyme E binds to A and the complex $E : A$ is subsequently transformed into C and E, thus recreating the enzyme. The enzyme E can be modeled in MIMc by the following term: $rec\ x.\{\xrightarrow{A}\{\longrightarrow\!\triangleright(x.E \mid \varnothing.C)\}\}.E$.

Definition 3.2.2 (Structural congruence)**.** The congruence relations $\equiv_x$ on the syntactical categories $x \in \{P, S, IS, \mu, M, \gamma\}$ of the calculus are the least equivalence relations closed under syntactical operators and such that the following laws hold:

1. $P_1 \mid P_2 \equiv_P P_2 \mid P_1, \quad P_1 \mid (P_2 \mid P_3) \equiv_P (P_1 \mid P_2) \mid P_3, \quad P \mid \mathbf{0} \equiv_P P$;
2. $S_1 : S_2 \equiv_{IS} S_2 : S_1, \quad \overline{S_1 S_2} \equiv_{IS} \overline{S_2 S_1}$;
3. $M_1 + M_2 \equiv_M M_2 + M_1,\ M_1 + (M_2 + M_3) \equiv_M (M_1 + M_2) + M_3,\ M + \varnothing \equiv_M M,$ $M + M \equiv_M M$;
4. (α-conversion) $\mu_1 \equiv_\mu \mu_2$ if they differ only on bound names;

5. $rec\ x.\mu \equiv_\mu \mu[rec\ x.\mu/x]$.

We omit the indication of x in $\equiv_x$ when no ambiguities arise.

We define a reduction semantics for the MIM calculus, given in terms of a Labelled Transition System (LTS) representing the possible evolutions of a term. The labels of the LTS are actions identifying the reactions that each single transition describes. The following possible actions are defined by the calculus:

- $N_1 \leftrightarrow N_2$: the creation of a non-covalent bond;
- $N_1 \nleftrightarrow N_2$: the cleavage of a non-covalent bond;
- $N \longrightarrow\!\!\!\!\bullet \{N_1, \ldots, N_k\}$: a conversion;
- $N \longrightarrow\!\!\!\!\triangleright \{N_1, \ldots, N_k\}$: a lossless production;
- $N_1 = N_2$: the creation of a covalent bond;
- $N \rightsquigarrow \overline{N_1 N_2}$: the cleavage of a covalent bond;
- $q \Longrightarrow N$: a covalent modification;
- $N \rightsquigarrow \overline{qN_1}$: the removal of a covalent modification;

where $N, N_1, N_2 \in \mathcal{N}$ denote species names. The set of all possible actions is denoted by Act.

Definition 3.2.3 (Reduction semantics)**.** The *reduction semantics of MIM calculus* is the relation $\xrightarrow{\alpha}$ on processes such that:

$$P \xrightarrow{\alpha} P' \iff \exists \iota \subset \mathcal{N}.\ P \xrightarrow{(\emptyset,\iota)\ \alpha} P' \tag{3.1}$$

where $\alpha \in Act$ is an action that represents the capability of P used for the reduction step, and $\xrightarrow{(\nu,\iota)\ \alpha}$, with $\nu, \iota \subset \mathcal{N}$, is the least relation on processes, closed under structural congruence $\equiv_P$, and satifying the inference rules shown in Figure 3.7.

Rule 3.2 deals with the creation of a non-covalent bond between molecules $\mu_1.S_1$ and $\mu_2.S_2$, thus producing a complex $\mu.(\mu_1.S_1 : \mu_2.S_2)$. Note that it is sufficient that only molecule $\mu_1.S_1$ has the capability of binding with a molecule with name $\lfloor S_2 \rfloor$; in fact, the symmetric capability is not required for $\mu_2.S_2$. Rule 3.3 deals with the cleavage of a non-covalent bond. There are no conditions for the cleavage, therefore it can occur at any time. Rule 3.4 deals with the conversion of a molecule $\mu.S$ into a number of other molecules when $\mu.S$ has the proper capability. Rule 3.5 deals with the lossless production, namely with the case in which $\mu.S$ produces a number of molecules without disappearing. Rules 3.6 and 3.7 are the analogs of rules 3.2 and 3.3 for the case of the covalent binding. The unbinding, expressed by rule 3.7, requires the presence of a molecule $\mu.S$ having the capability of breaking the bond. Rules 3.8 and 3.9 deal with molecule covalent modification of a type q and with the removal of the modification, respectively.

Rule 3.10 is used to apply a step of the reduction to the parallel composition of processes. Given the term Q, with which P is composed, the conditions of the rule ensure that the step is forbidden if any of the molecules present in Q are also inhibitors for the

$$\frac{\mu_1 = \{X + (\nu, \iota) \xrightarrow{\lfloor S_2 \rfloor} \mu\} \qquad \alpha = \lfloor S_1 \rfloor \leftrightarrow \lfloor S_2 \rfloor}{\mu_1.S_1 \mid \mu_2.S_2 \xrightarrow{(\nu,\iota)\ \alpha} \mu.(\mu_1.S_1 : \mu_2.S_2)} \tag{3.2}$$

$$\frac{\alpha = \lfloor S_1 \rfloor \nleftrightarrow \lfloor S_2 \rfloor}{\mu.(S_1 : S_2) \xrightarrow{(\emptyset,\emptyset)\ \alpha} S_1 \mid S_2} \tag{3.3}$$

$$\frac{\mu = \{X + (\nu, \iota) \longrightarrow P\} \qquad \alpha = \lfloor S \rfloor \longrightarrow \lfloor P \rfloor}{\mu.S \xrightarrow{(\nu,\iota)\ \alpha} P} \tag{3.4}$$

$$\frac{\mu = \{X + (\nu, \iota) \longrightarrow\!\!\!\!\triangleright\ P\} \qquad \alpha = \lfloor S \rfloor \longrightarrow\!\!\!\!\triangleright\ \lfloor P \rfloor}{\mu.S \xrightarrow{(\nu,\iota)\ \alpha} \mu.S \mid P} \tag{3.5}$$

$$\frac{\mu_1 = \{X + (\nu, \iota) \overset{\lfloor S_2 \rfloor}{=\!=\!=} \mu\} \qquad \alpha = \lfloor S_1 \rfloor =\!=\!= \lfloor S_2 \rfloor}{\mu_1.S_1 \mid \mu_2.S_2 \xrightarrow{(\nu,\iota)\ \alpha} \mu.\overline{(\mu_1.S_1)(\mu_2.S_2)}} \tag{3.6}$$

$$\frac{\mu = \{X + (\nu, \iota) \overset{\lfloor \overline{S_1 S_2} \rfloor}{\rightsquigarrow}\} \qquad \alpha = \lfloor S \rfloor \rightsquigarrow \lfloor \overline{S_1 S_2} \rfloor}{\mu.S \mid \mu'.\overline{S_1 S_2} \xrightarrow{(\nu,\iota)\ \alpha} \mu.S \mid S_1 \mid S_2} \tag{3.7}$$

$$\frac{\mu_1 = \{X + (\nu, \iota) \xRightarrow{q} \mu\} \qquad \alpha = q \Longrightarrow \lfloor S_1 \rfloor}{\mu_1.S_1 \xrightarrow{(\nu,\iota)\ \alpha} \mu.\overline{q(\mu_1.S_1)}} \tag{3.8}$$

$$\frac{\mu = \{X + (\nu, \iota) \overset{\lfloor \overline{qS_1} \rfloor}{\rightsquigarrow}\} \qquad \alpha = \lfloor S \rfloor \rightsquigarrow \lfloor \overline{qS_1} \rfloor}{\mu.S \mid \mu'.\overline{qS_1} \xrightarrow{(\nu,\iota)\ \alpha} \mu.S \mid S_1} \tag{3.9}$$

$$\frac{P \xrightarrow{(\nu,\iota)\ \alpha} P' \qquad \lfloor Q \rfloor \cap \iota = \emptyset \qquad \nu' = \nu \setminus \lfloor Q \rfloor}{P \mid Q \xrightarrow{(\nu',\iota)\ \alpha} P' \mid Q} \tag{3.10}$$

Figure 3.7: The inference rules defining the semantics of the MIM calculus.

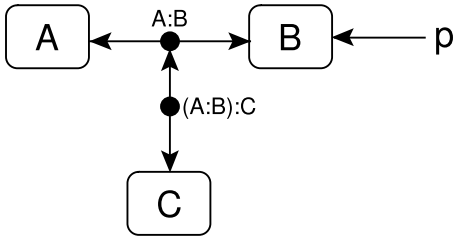


Figure 3.8: A MIM diagram.

interaction, namely $\lfloor Q \rfloor \cap \iota = \emptyset$. As regards the set of promoters ν, it is updated to remove those which are present in Q, since their constraint is satisfied. This last condition, together with the fact that the semantics requires that the set ν of promoters is empty to actually do the reduction step (Equation 3.1), ensures that all the promoters of the capability used for that step are present in the parallel composition of processes, among those elements which are not involved in the interaction. As usual, we define $\rightarrow^*$ as the reflexive and transitive closure of relation $\xrightarrow{\alpha}$.

Example 3.2.1. Consider an example of a MIM process which represents a molecular system described by the MIM diagram shown in Figure 3.8.

A MIM process, differently from a MIM diagram, represents both the possible interactions among the species and the number of molecules that are present in the system. For the sake of conciseness, let us denote by μ_A, μ_B, μ_C the capabilities of species A, B and C, respectively. These capabilities are defined as follows:

$$\mu_A = \{\xrightarrow{B}\{\xrightarrow{C} \varnothing\}\}$$
$$\mu_B = \{\xrightarrow{A}\{\xrightarrow{C} \varnothing\} + \xRightarrow{p} \varnothing\}$$
$$\mu_C = \{\xrightarrow{A:B} \varnothing\}.C$$

The following MIM process corresponds to a system with species A, B and C, with the interaction capabilities described by the diagram in Figure 3.8, and in which there are two molecules of A, two of B and one of C:

$$P = \mu_A.A \mid \mu_A.A \mid \mu_B.B \mid \mu_B.B \mid \mu_C.C$$

In the process P, the species A can complex with B, producing a molecule able to complex with C. Species B is also able to form a complex with A, and it can also be phosphorilated. Finally, species C can complex with $A : B$.

For example, species A and B can bind to form a complex, as described by the following transition:

$$\mu_A.A \mid \mu_B.B \mid P' \xrightarrow{(\emptyset,\emptyset)\ A \leftrightarrow B} \{\xrightarrow{C} \varnothing\}.(\mu_A.A : \mu_B.B) \mid P'$$

where $P' = \mu_A.A \mid \mu_B.B \mid \mu_C.C$. The term obtained, describing a complex species of name $A : B$, can in turn react with a C, as expressed by the capability $\xrightarrow{C} \varnothing$. Such a reaction is described by the following transition of the semantics:

$$\{\xrightarrow{C} \varnothing\}.(\mu_A.A : \mu_B.B) \mid \mu_C.C \mid P'' \xrightarrow{(\emptyset,\emptyset)\ A:B \leftrightarrow C} \varnothing.\left(\{\xrightarrow{C} \varnothing\}.(\mu_A.A : \mu_B.B) : \mu_C.C\right) \mid P''$$

where $P'' = \mu_A.A \mid \mu_B.B$. Recall that the non-covalent bonds of $A : B$ and $(A : B) : C$ can dissociate autonomously as described by rule 3.3 of the reduction semantics.

3.3 Consistency

In this section we investigate the relationship between Molecular Interaction Maps and the MIM Calculus. We propose three *consistency* definitions, with the aim of identifying the terms of the calculus which could be a formal representation of a MIM diagram. Recall that an important difference between MIM diagrams and the MIM calculus is that diagrams provide a static view of the species of a system, and of the interactions which can occur among the species, while MIM calculus allows representing single molecules and provides a semantics for deriving the evolution of the described system. In MIM diagrams, the capabilities of each molecule depend only on the species of the molecule, and are irrespective, for example, of the different reactions that might produce a molecule of that species. On the contrary, the MIM calculus allows representing single molecules $\mu.IS$, and different molecules of the same species might have different capabilities.

For example, term $P = \{\xrightarrow{B} \mu_1\}.A \mid \varnothing.A \mid P'$ contains two molecules of species A with different capabilities: the first one can bind to another molecule of species B, while the second one has no capabilities. Note that these two molecules of species A could have been obtained as a result of other reactions (for example, by transformation of other molecules), hence during any evolution of the system there may be some states in which all the molecules of the same species have the same capabilities while, in other states, this is not true.

It appears to be of particular interest to establish which terms of the MIM calculus represent MIM diagrams, in the sense that a MIM diagram can be associated with a term, and the term evolves in accordance with the behaviour intended by the diagram. One may also ask that in a term molecules of a certain species always have the same capabilities. This captures the constraint of uniqueness of species in MIM diagrams.

For this purpose, we present three different definitions of consistency of MIM calculus terms, namely *semantic consistency*, *(weak) syntactic consistency*, and *strong syntactic consistency*. Semantic consistency is the weakest form of consistency, and takes into account only terms that can be reached from the initial state. This form of consistency requires that, whenever a molecule of a certain species named N is produced, i.e. a molecule S, with $\lfloor S \rfloor = N$, appears in the top-level parallel composition, it always has the same capabilities.

Definition 3.3.1 (Semantic Consistency)**.** A term P is *semantically consistent* iff

$$\forall \mu_1.S_1, \mu_2.S_2, P', P''.$$
$$\lfloor S_1 \rfloor \equiv \lfloor S_2 \rfloor \wedge P \rightarrow^* \mu_1.S_1 \mid P' \wedge P \rightarrow^* \mu_2.S_2 \mid P'' \implies \mu_1 \equiv \mu_2$$

The definitions of syntactic consistencies deal instead with the species that syntactically appear in a term. Before defining formally the two forms of syntactic consistencies, namely the weak syntactic consistency and the strong syntactic consistency, we need to define *contexts*. Contexts allow identifying the position inside a term in which a molecule of a certain species (with its capabilities) appears. Formally, a context is a term with a *hole*, denoted as $\Box$, which occurs in the position of a capability μ. Therefore the hole corresponds to the collection of capabilities of a molecule.

Definition 3.3.2 (Context)**.** Contexts of MIM calculus are defined by the following grammar:

$$
\begin{array}{rcl}
C & ::= & P \mid S_c \\
S_c & ::= & \mu_c.IS \quad \mid \quad \mu.IS_c \\
IS_c & ::= & S : S_c \quad \mid \quad \overline{qS_c} \quad \mid \quad \overline{SS_c} \\
\mu_c & ::= & \Box \quad \mid \quad M + \gamma_c
\end{array}
$$

$$
\begin{array}{rcll}
\gamma_c & ::= & (\nu, \iota) \xrightarrow{N} \mu_c & \textit{(non-covalent binding)} \\
& \mid & (\nu, \iota) \overset{N}{=\!=} \mu_c & \textit{(covalent binding)} \\
& \mid & (\nu, \iota) \overset{q}{\Longrightarrow} \mu_c & \textit{(covalent modification)} \\
& \mid & (\nu, \iota) \longrightarrow C & \textit{(conversion)} \\
& \mid & (\nu, \iota) \longrightarrow\!\!\triangleright\, C & \textit{(lossless production)}
\end{array}
$$

The set of all contexts is denoted by $\mathcal{C}$.

The syntax ensures that exactly one hole is present in a context. Given S_c, the hole can occur either in the capabilities of S_c itself, when $S_c = \mu_c.IS$, or in the capabilities of one of the molecules of which the molecule is composed, $S_c = \mu.IS_c$. Given a capability with a hole μ_c, the hole can be either the capability itself $\Box$, or it can occur in one of the capabilities appearing in μ_c. In particular, if $\mu_c = \gamma_c$ (a basic capability) the hole can occur inside the capabilities of the species which can be produced by γ_c. Given a context C, its hole can be substituted with a capability μ, giving a process denoted $C[\mu]$.

For example, the context $C_1 = \mu_1.(\mu_2.A : \Box.B)$ represents a molecule complex $A : B$ in which the hole refers to the capability of B forming the complex. Context C_1 can be applied to a capability μ_3 obtaining $C_1[\mu_3] = \mu_1.(\mu_2.A : \mu_3.B)$. Note that in this case, the hole is relative to a species named B, and this is clearly visible from the syntax of the context. However, in other cases, the name of the species relative to a hole is not directly present in the syntax of the context. For example, the hole in context $C_2 = \{\xrightarrow{B} \Box + \longrightarrow\!\!\triangleright\, P\}.A$ is relative to the species obtained as a complexation between A and B, whose name is $A : B$, which is not directly present in the syntax of the context. In order to extract, from a given context, the name of the species relative to the hole, we use a function $name : \mathcal{C} \to \mathcal{N}$ defined as follows.

Definition 3.3.3. Function $name$, from contexts $\mathcal{C}$ to molecular names $\mathcal{N}$, and function $name'(\mu_c, N) = N$, from a context μ_c and a name $\mathcal{N}$ to a name $\mathcal{N}$, are recursively defined as follows:

$$name(P \mid S_c) = name(S_c) \tag{3.11}$$
$$name(\mu.IS_c) = name(IS_c) \tag{3.12}$$
$$name(S : S_c) = name(S_c) \tag{3.13}$$
$$name(\overline{qS_c}) = name(S_c) \tag{3.14}$$
$$name(\overline{SS_c}) = name(S_c) \tag{3.15}$$
$$name(\mu_c.IS) = name'(\mu_c, \lfloor IS \rfloor) \tag{3.16}$$
$$name'(\Box, N) = N \tag{3.17}$$

$$name'((\nu,\iota) \xrightarrow{N'} \mu_c, N) = name'(\mu_c, N : N') \tag{3.18}$$

$$name'((\nu,\iota) \overset{N'}{=\!=} \mu_c, N) = name'(\mu_c, \overline{N\,N'}) \tag{3.19}$$

$$name'((\nu,\iota) \xRightarrow{q} \mu_c, N) = name'(\mu_c, \overline{qN}) \tag{3.20}$$

$$name'((\nu,\iota) \longrightarrow C, N) = name(C) \tag{3.21}$$

$$name'((\nu,\iota) \longrightarrow\!\!\triangleright\, C, N) = name(C) \tag{3.22}$$

$$name'(M + \gamma_c, N) = name(\gamma_c, N) \tag{3.23}$$

Definition of function *name* is given by two mutually recursive functions *name* and *name′*. In particular, *name′* takes two parameters, a capability context μ_c and a name N, where N is the name of the species with which this capability is associated. The function *name′* is used in Equation 3.16, where extracting the name, relative to the hole, from a context $\mu_c.IS$ is reduced to extracting the name from μ_c, knowing that the capability μ_c is relative to a species named $\lfloor IS \rfloor$.

Given the definition of contexts, we can formally define the *weak syntactic consistency* and the *strong syntactic consistency*. The weak form requires that the capabilities of each molecule of a species appearing in the term, including those forming compound molecules and those that can be obtained as the result of reactions, always have the same capabilities.

Definition 3.3.4 ((Weak) Syntactic Consistency)**.** A term P is *(weakly) syntactic consistent* iff

$$\forall C_1, C_2, \mu_1, \mu_2. \quad name(C_1) \equiv name(C_2) \wedge P \equiv C_1[\mu_1] \equiv C_2[\mu_2] \implies \mu_1 \equiv \mu_2$$

Strong syntactic consistency adds a further constraint, by requiring also that, whenever a non-covalent bond ($\longrightarrow$) or a covalent bond ($=\!=$) can be created between two species, then both species have the corresponding capability. In the definition, we write $\gamma \in \mu$ as a shorthand for $\exists M.\ \mu \equiv \{M + \gamma\}$.

Definition 3.3.5 (Strong Syntactic Consistency)**.** A *weakly* syntactic consistent term P is *strongly syntactic consistent* iff

$$\begin{aligned}
&\forall C_1, C_2, \mu_1, \mu_2, N_1, N_2, \mu. \\
&\quad name(C_1) \equiv N_1 \wedge name(C_2) \equiv N_2 \wedge P \equiv C_1[\mu_1] \equiv C_2[\mu_2] \implies \\
&\qquad \left(\left(\xrightarrow{N_2} \mu\right) \in \mu_1 \iff \left(\xrightarrow{N_1} \mu\right) \in \mu_2\right) \wedge \\
&\qquad \left(\left(\overset{N_2}{=\!=} \mu\right) \in \mu_1 \iff \left(\overset{N_1}{=\!=} \mu\right) \in \mu_2\right)
\end{aligned}$$

For example, term $P_1 = \{\xrightarrow{B} \mu_1\}.A \mid \varnothing.B$ is weakly syntactic consistent, but not strongly syntactic consistent, since molecule B does not have the capability of binding (with a non-covalent bond) to A. The strongly syntactic consistent term corresponding to P_1 is $P_2 = \{\xrightarrow{B} \mu_1\}.A \mid \{\xrightarrow{A} \mu_1\}.B$.

The following proposition shows that syntactic consistency implies semantic consistency.

Proposition 3.3.1. *$\forall P$. P is syntactically consistent $\implies$ P is semantically consistent.*

Proof. It is sufficient to prove that $P \rightarrow^* \mu.S \mid P'$ implies $\exists C.\ C[\mu] \equiv P$ and $name(C) = \lfloor S \rfloor$. This proof is done by induction on the length of the sequence of transitions $P \rightarrow^* \mu.S \mid P'$. Let us assume that such a sequence has the following form $P \equiv P_0 \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} P_n \equiv \mu.S \mid P'$.
As regards the base case ($n = 0$), we have $P \equiv \mu.S \mid P'$. Then, context $C = \Box.S \mid P'$ is such that $C[\mu] \equiv P$ and $name(C) = \lfloor S \rfloor$.

As regards the induction step, let $n > 1$ and suppose that the property holds for all $m < n$. We have $P_n \equiv \mu.S \mid P'$ and there are two cases to be considered: either $\mu.S$ already appeared before, i.e. $\exists\, \overline{n} < n.\ P_{\overline{n}} \equiv \mu.S \mid P''$ for some P'', or not. In the first case, by induction hypothesis, there exists a context C such that $C[\mu] \equiv P$ and $name(C) = \lfloor S \rfloor$. In the second case, $\mu.S$ has been created in the last execution step $P_{n-1} \xrightarrow{\alpha_n} \mu.S \mid P'$, where $\mu.S$ does not occur in the top-level parallel composition, i.e. $\nexists Q.\ P_{n-1} \equiv \mu.S \mid Q$. According to the semantics, $P_{n-1} \xrightarrow{\alpha_n} \mu.S \mid P'$ iff $P_{n-1} \xrightarrow{(\emptyset,\iota)\ \alpha_n} \mu.S \mid P'$ for some $\iota \subset \mathcal{N}$. By rule induction on the rules 3.2–3.10 of the semantics, we prove that, for all transitions $Q \xrightarrow{(\nu,\iota)\alpha_n} Q'$, for any $\mu.S$ created in the transition there is a context C such that $C[\mu] \equiv P$ and $name(C) = \lfloor S \rfloor$.

- *Rule 3.2*: $\mu_1.S_1 \mid \mu_2.S_2 \xrightarrow{(\nu,\iota)\ \alpha} \mu.(\mu_1.S_1 : \mu_2.S_2)$ with $\mu_1 \equiv \{X + (\nu,\iota) \xrightarrow{\lfloor S_2 \rfloor} \mu\}$.
  By induction hypothesis, there exists a context $\overline{C}$ such that $\overline{C}[\mu_1] \equiv P$ and $name(\overline{C}) = \lfloor S_1 \rfloor$. Therefore, context $C = \overline{C}[\{X + (\nu,\iota) \xrightarrow{\lfloor S_2 \rfloor} \Box\}]$ is such that $C[\mu] \equiv P$ and $name(C) = \lfloor (\mu_1.S_1 : \mu_2.S_2) \rfloor$.

- *Rule 3.3*: $\mu.(\mu_1.S_1 : \mu_2.S_2) \xrightarrow{(\emptyset,\emptyset)\ \alpha} \mu_1.S_1 \mid \mu_2.S_2$.
  Let $\overline{C}$ be the context of $\mu.(\mu_1.S_1 : \mu_2.S_2)$ (by induction hypothesis). Context $\overline{C}$ must contain the portion $\Box.(\mu_1.S_1 : \mu_2.S_2)$. Suppose that at least one of $\mu_1.S_1$ and $\mu_2.S_2$ never appeared before in any P_m, $m < n$ (otherwise, by induction hypothesis, their contexts are already known). Hence, term $S = \mu.(\mu_1.S_1 : \mu_2.S_2)$ has not been obtained by applying rule 3.2, but by one of the rules 3.3,3.4,3.5, 3.7,3.9. This means that S appeared literally in the inital term P, thus contexts C_1 can be obtained from $\overline{C}$ by replacing $\Box.(\mu_1.S_1 : \mu_2.S_2)$ with $\mu.(\Box.S_1 : \mu_2.S_2)$ and contexts C_2 can be obtained from $\overline{C}$ by replacing $\Box.(\mu_1.S_1 : \mu_2.S_2)$ with $\mu.(\mu_1.S_1 : \Box.S_2)$. Contexts C_1 and C_2 are such that $C_1[\mu_1] \equiv P$ with $name(C_1) = \lfloor S_1 \rfloor$ and $C_2[\mu_2] \equiv P$ with $name(C_2) = \lfloor S_2 \rfloor$.

- *Rule 3.4*: $\mu_1.S_1 \xrightarrow{(\nu,\iota)\ \alpha} \mu.S \mid P'$ with $\mu_1 = \{X + (\nu,\iota) \longrightarrow (\mu.S \mid P')\}$.
  Let $\overline{C}$ be the context of $\mu_1.S_1$. Then the context for $\mu.S$ is $C = \overline{C}[\{X + (\nu,\iota) \longrightarrow (\Box.S \mid P')\}]$.

- *Rule 3.5*: analogous to rule 3.4.

- *Rule 3.6*: analogous to rule 3.2.

- *Rule 3.7*: $\mu_1.S_1 \mid \mu'.\overline{(\mu_2.S_2)(\mu_3.S_3)} \xrightarrow{(\nu,\iota)\ \alpha} \mu_1.S_1 \mid \mu_2.S_2 \mid \mu_3.S_3$.
  Similarly to rule 3.3, if either $\mu_2.S_2$ or $\mu_3.S_3$ did not appear before, their contexts can be obtained from context $\overline{C}$ of $\mu'.\overline{(\mu_2.S_2)(\mu_3.S_3)}$. Context $\overline{C}$ must contain $\Box.\overline{(\mu_2.S_2)(\mu_3.S_3)}$. We obtain context C by replacing $\Box.\overline{(\mu_2.S_2)(\mu_3.S_3)}$ with

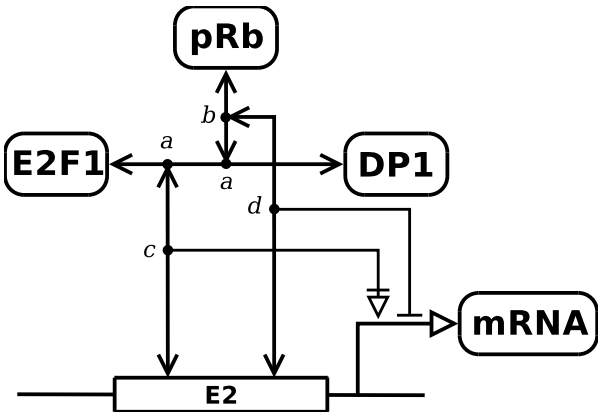


Figure 3.9: Molecular interaction map representing interactions among E2F1, DP1, and pRb.

$\mu'.\overline{(\Box.S_2)(\mu_3.S_3)})$ in $\overline{C}$ for $\mu_2.S_2$. Similarly, we obtain context C by replacing $\Box.\overline{(\mu_2.S_2)(\mu_3.S_3)}$ with $\mu'.\overline{(\mu_2.S_2)(\Box.S_3)})$ in $\overline{C}$ for $\mu_3.S_3$.

- *Rule 3.8*: $\mu_1.S_1 \xrightarrow{(\nu,\iota)\ \alpha} \mu.\overline{q(\mu_1.S_1)}$ with $\mu_1 = \{X + (\nu,\iota) \stackrel{q}{\Longrightarrow} \mu\}$.
  Let $\overline{C}$ be the context of $\mu_1.S_1$. Then the context for $\mu.\overline{q(\mu_1.S_1)}$ is $C = \overline{C}[\{X + (\nu,\iota) \stackrel{q}{\Longrightarrow} \Box\}]$.

- *Rule 3.9*: analogous to rules 3.3 and 3.7.

- *Rule 3.10*: $P \mid Q \xrightarrow{(\nu',\iota)\ \alpha} P' \mid Q$.
  Since Q is not modified by the transition, only terms $\mu.S$ in P' could have been created by the transition. The contexts of any $\mu.S$ in P' is given by the induction hypothesis on the rule. □

3.4 An example of modelling

In this section we show an example of a real molecular interaction map, taken from [50], and we show the corresponding term in the MIM calculus. Differently from Kohn maps, the MIM calculus can contain multiple molecules of a same molecular species, thus it can describe the evolution of the system starting from an initial configuration.

The example in [50] presents a comprehensive molecular interaction map of regulators of cell cycle and DNA repair processes. The presented map is limited to the events in the mammalian cell nucleus. We consider here only the interaction between a protein of the E2F family and a gene promoter E2. This interaction is an important part of the cell cycle. The transcription of the gene is activated or inhibited by the binding of different complexes with the promoter. The molecular interaction map representing the interactions among stimulatory and inhibitory complexes of E2F1, DP1 and pRb is shown in Figure 3.9.

The E2F1:DP1 dimer (indicated by nodes labelled (a) in Figure 3.9) and the (E2F1:DP1):pRb trimer (node (b)) can be bound to the promoter element E2. When the E2F1:DP1 dimer is bound to E2 the transcription activity is stimulated, while when the

(E2F1:DP1):pRb trimer is bound to E2 the transcription is inhibited. The stimulation is represented by the strongest constraint or requirement. We represent each species involved by a MIMc term with the capabilities of the species itself. In particular we use particular terms for representing the promoter E2 and the DNA. In the Kohn map the DNA is implicitly represented, but in a MIMc term DNA must be represented explicitly, for assigning to it the capability of producing the mRNA. Remark that we can have multiple copies of the species E2F1, DP1, and pRb in a MIMc term representing the system. Coherently with the cell system we can have only one copy of the DNA and of the gene promoter E2. Each basic element is identified by an elementary species $E2F1, DP1, pRb, E2, DNA, mRNA \in \mathcal{E}$.

The E2F1 species can be represented by the following term:

$$\{\xrightarrow{DP1}\{\xrightarrow{E2}\varnothing + \xrightarrow{pRb}\{\xrightarrow{E2}\varnothing\}\}\}.E2F1 = \mu_1.E2F1$$

which states that $E2F1$ can be bound to $DP1$, and then the dimer can either be bound to $E2$ or to pRb. When the $E2F1 : DP1$ dimer is bound to pRb, it can in turn bind to promoter $E2$ to inhibit the transcription. Note that the stimulation of DNA transcription by the trimer is not modeled among the capability of the species, which are just empty. Instead, this behaviour is captured by the DNA process, shown in the following. As for stimulation, inhibition is captured inside the definiton of the DNA. Species $DP1$ and pRb are represented by the following terms:

$$\{\xrightarrow{E2F1}\{\xrightarrow{E2}\varnothing + \xrightarrow{pRb}\{\xrightarrow{E2}\varnothing\}\}\}.DP1 = \mu_2.DP1$$
$$\{\xrightarrow{E2F1:DP1}\{\xrightarrow{E2}\varnothing\}\}.pRb = \mu_3.pRb$$

Finally, the promoter E2 and the DNA can be represented by the terms:

$$\{\xrightarrow{E2F1:DP1}\varnothing + \xrightarrow{(E2F1:DP1):pRb}\varnothing\}.E2 = \mu_4.E2$$
$$\{(\nu_{DNA}, \iota_{DNA}) \longrightarrow\!\triangleright\, mRNA\}.DNA$$

where

$$\nu_{DNA} = \{(E2F1 : DP1) : E2\}$$
$$\iota_{DNA} = \{((E2F1 : DP1) : pRb) : E2\}$$

The lossless production of $mRNA$ by the DNA is regulated by the presence/absence of the two complexes $(E2F1 : DP1) : E2$ and $((E2F1 : DP1) : pRb) : E2$. In particular, the former complex represents a promoter (triggering the reaction), while the latter represents an inhibitor for the reaction.

An initial configuration in which two molecules of species E2F1, DP1 and pRb are present is represented by the following MIMc term:

$$P_1 = \mu_1.E2F1 \mid \mu_1.E2F1 \mid \mu_2.DP1 \mid \mu_2.DP1 \mid \mu_3.pRb \mid \mu_3.pRb \mid \mu_4.E2$$
$$\mid \{(\nu_{DNA}, \iota_{DNA}) \longrightarrow\!\triangleright\, mRNA\}.DNA$$

The term can evolve towards different configurations. For example, after a complexation between $E2F1$ and $DP1$ occurs, the processes $\mu_1.E2F1$ and $\mu_2.DP1$ are replaced by the following term, representing a complex with name $E2F1 : DP1$:

$$\{\xrightarrow{E2}\varnothing + \xrightarrow{pRb}\{\xrightarrow{E2}\varnothing\}\}.(\mu_1.E2F1 : \mu_2.DP1)$$

Thus the whole term becomes:

$$\begin{aligned} P_2 = {} & \mu_1.E2F1 \mid \mu_2.DP1 \mid \mu_3.pRb \mid \mu_3.pRb \mid \mu_4.E2 \\ & \mid \{(\nu_{DNA}, \iota_{DNA}) \longrightarrow\!\triangleright\, mRNA\}.DNA \\ & \mid \{\xrightarrow{E2} \varnothing + \xrightarrow{pRb}\{\xrightarrow{E2} \varnothing\}\}.\,(\mu_1.E2F1 : \mu_2.DP1) \end{aligned}$$

As a further evolution step we may have the binding of the dimer $E2F1 : DP1$ to the promoter $E2$. The resulting term is:

$$\begin{aligned} P_3 = {} & \mu_1.E2F1 \mid \mu_2.DP1 \mid \mu_3.pRb \mid \mu_3.pRb \\ & \mid \{(\nu_{DNA}, \iota_{DNA}) \longrightarrow\!\triangleright\, mRNA\}.DNA \\ & \mid \varnothing.\left((\{\xrightarrow{E2} \varnothing + \xrightarrow{pRb}\{\xrightarrow{E2} \varnothing\}\}.(\mu_1.E2F1 : \mu_2.DP1)) : \mu_4.E2\right) \end{aligned}$$

As an example of derivation, we show how the semantics is applied to the term P_1 above obtaining the term P_2 in a single reduction step. For the sake of readability, we write the terms P_1, P_2 as:

$$\begin{aligned} P_1 &= \mu_1.E2F1 \mid \mu_2.DP1 \mid Q \\ P_2 &= \mu.\,(\mu_1.E2F1 : \mu_2.DP1) \mid Q \end{aligned}$$

where

$$\begin{aligned} Q = {} & \mu_1.E2F1 \mid \mu_2.DP1 \mid \mu_3.pRb \mid \mu_3.pRb \mid \mu_4.E2 \\ & \mid \{(\nu_{DNA}, \iota_{DNA}) \longrightarrow\!\triangleright\, mRNA\}.DNA \\ \mu = {} & \{\xrightarrow{E2} \varnothing + \xrightarrow{pRb}\{\xrightarrow{E2} \varnothing\}\}. \end{aligned}$$

The transition $P_1 \xrightarrow{(\nu',\iota)} P_2$, with $\nu' = \iota = \emptyset$, is obtained with the following derivation, by using the rules of the semantics:

$$\dfrac{\dfrac{\mu_1 = \{X + (\nu, \iota) \xrightarrow{DP1} \mu\} \qquad \nu = \iota = \emptyset}{\mu_1.E2F1 \mid \mu_2.DP1 \xrightarrow{(\nu,\iota)\quad E2F1 \leftrightarrow DP1} \mu.\,(\mu_1.E2F1 : \mu_2.DP1)} \qquad \lfloor Q \rfloor \cap \iota = \emptyset}{\mu_1.E2F1 \mid \mu_2.DP1 \mid Q \xrightarrow{(\nu',\iota)\quad E2F1 \leftrightarrow DP1} \mu.\,(\mu_1.E2F1 : \mu_2.DP1) \mid Q}$$

where $\nu' = \nu \setminus \lfloor Q \rfloor = \emptyset$. Finally, according to the definition (Equation 3.1), we have the transition $P_1 \xrightarrow{E2F1 \leftrightarrow DP1} P_2$.

3.5 Extension with compartments

In this section we extend the definition of the MIM calculus to allow specifying a position for each element of a system. We provide a quite abstract notion of space, by assuming a set of *names of compartments* $\mathcal{H}$, where each term of the calculus can be associated with a compartment name, for specifying its position. Therefore, also the interaction capabilities of processes need to take into account the position of the involved elements. For example,

it is possible to specify, for an element, the capability to bind to another element in a different compartment.

The syntax of the *MIM calculus with compartments* (*MIMc-co*) is a straightforward extension of the original syntax of the calculus, defined in Section 3.2. Each species $\mu.IS$ now needs to specify the compartment in which it occurs using a new spatial operator @. Formally, each species is of the form $\mu.IS@h$, where $h \in \mathcal{H}$ is the name of the compartment. Moreover, it is necessary to extend the operators of the calculus for describing the capabilities of creating bonds (both non-covalent $\longrightarrow$ and covalent $=\!=$), and for the cleavage $\rightsquigarrow$, to allow specifying the compartment for the involved molecular species. As regards the operators of conversion $\longrightarrow\!\!\blacktriangleright$ and lossless production $\longrightarrow\!\!\triangleright$, the compartment for each resulting species is specified inside the resulting term. The formal definition of the syntax of the calculus follows.

Definition 3.5.1 (Syntax)**.** *Processes* P, *named species* S and capabilities μ of the *MIM calculus with compartments* are defined by the following grammar:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \;\mid\; S \;\mid\; P \mid P \\ S & ::= & \mu.IS@h \\ IS & ::= & A \;\mid\; S : S \;\mid\; \overline{q\ S} \;\mid\; \overline{S\ S} \\ \mu & ::= & rec\ x.\mu \;\mid\; M \;\mid\; x \\ M & ::= & \varnothing \;\mid\; M + M \;\mid\; \gamma \end{array}$$

where simple interaction capabilities γ are defined as follows:

$$\begin{array}{rcll} \gamma & ::= & (\nu, \iota) \xrightarrow{N@h} \mu & \textit{(non-covalent binding)} \\ & \mid & (\nu, \iota) \overset{N@h}{=\!=} \mu & \textit{(covalent binding)} \\ & \mid & (\nu, \iota) \overset{q}{\Longrightarrow} \mu & \textit{(covalent modification)} \\ & \mid & (\nu, \iota) \overset{N@h}{\rightsquigarrow} & \textit{(cleavage)} \\ & \mid & (\nu, \iota) \longrightarrow\!\!\blacktriangleright P & \textit{(conversion)} \\ & \mid & (\nu, \iota) \longrightarrow\!\!\triangleright P & \textit{(lossless production)} \end{array}$$

As for the original definition, $\mathbf{0}$ denotes the empty process, $A \in \mathcal{E}$ denotes an elementary species name, and $q \in \mathcal{E}_c$ denotes the type of modification. As regards the capabilities, $x \in \mathcal{X}$ is a variable, N is the species name, while ν and ι are sets of species names. As before, the set of all species names is denoted by $\mathcal{N}$, whose elements are terms with empty capabilities.

We assume that the structural congruence relations, as defined in Definition 3.2.2, are extended to terms of the MIM calculus with compartments.

In MIMc-co, species names from the set $\mathcal{N}$ specify also the compartment of each molecular species appearing in it. For example, $N_1 = \varnothing.((\varnothing.A@out) : (\varnothing.B@in))@in$ is the name of a complex formed by a molecule A originating from compartment *out* and a molecule B originating from compartment *in*, and where the complex itself occurs in compartment *in*. For readability, we usually omit the indication of the compartments of subterms, when it does not introduce ambiguities; that is, name N_1 can be written as $N_1 = (A : B)@in$. Moreover, we assume functions $\lfloor . \rfloor$, which give the names of either a given named species $\mu.IS@h$, or a process P, to be extended to the syntax of MIMc-co.

3.5.1 The semantics of the binding operators

In the MIM calculus with compartments, an important distinction with respect to the original semantics of the MIM calculus (Definition 3.2.3) concerns the behaviour of the two binding operators, namely for the creation of non-covalent bonds $\longrightarrow$, and for covalent bonds $=\!=$. Recall that, in the basic version of the MIM calculus, in order to model two species A and B which can bind together, it is sufficient that one of the two terms describing them possesses the capability of creating a bond with the other species. Moreover, such a bond would create a complex with the same name, independently from the fact that only the one or the other have such a capability. For example, consider the following terms:

$$P_A = \{\}.A \qquad P'_A = \{\xrightarrow{B} \mu\}.A$$
$$P_B = \{\}.B \qquad P'_B = \{\xrightarrow{A} \mu\}.B$$

The only term reachable with one transition of the semantics from term $P_1 = P'_A \mid P_B$ is $P'_1 = \mu.(\{\xrightarrow{B} \mu\}.A : \{\}.B)$, labelled by the action of complexation $A \leftrightarrow B$, namely $P_1 \xrightarrow{A \leftrightarrow B} P'_1$. Moreover, the reverse transition is also possible, that is: $P'_1 \xrightarrow{A \nleftrightarrow B} P_1$. Note that P'_1 is composed of only one molecular species whose name is $A : B$, in fact $\lfloor P'_1 \rfloor = \{A : B\}$.

It is easy to see that also terms $P_2 = P_A \mid P'_B$ and $P_3 = P'_A \mid P'_B$, after a transition labelled $A \leftrightarrow B$, both yield a term containing just one molecular species whose name is $A : B$. That is, given $P_2 \xrightarrow{A \leftrightarrow B} P'_2$ and $P_3 \xrightarrow{A \leftrightarrow B} P'_3$, then P'_2 and P'_3 are such that $\lfloor P'_2 \rfloor = \lfloor P'_3 \rfloor = \{A : B\}$. Therefore, let us consider term $Q = \{\xrightarrow{A:B} \mu'\}.C$, in which there is another species C which can bind to the complex $A : B$. When term Q is put in parallel to either one of P_1, P_2, or P_3, then it is always possible to perform the following transitions:

$$P_i \mid Q \xrightarrow{A \leftrightarrow B} P'_i \mid Q \xrightarrow{A:B \leftrightarrow C} \mu'.(P'_i : Q) \qquad \forall\, i = 1, 2, 3$$

yielding molecules $\mu'.(P'_i : Q)$ for $i = 1, 2, 3$, each having the same name $(A : B) : C$.

However, when using compartments, the semantics of terms like those of the example, when A and B are in different compartments, is no longer the same. In fact, we assume that, whenever a species $\mu_1.S_1@h_1$ has either a capability $\xrightarrow{S_2@h_2}$ or $\xlongequal{S_2@h_2}$, then the resulting complex has to be positioned in the compartment of the other species, namely h_2. As the following example shows, this allows us to avoid the specification of the compartment for the resulting compound molecule, since it is determined by which of the two binding species has the capability of creating the bond.

For example, consider the term $Q_1 = \{\xrightarrow{B@in} \mu\}.A@out \mid \{\}.B@in$, in which there is an element A in compartment *out*, an element B in compartment *in*, and where only A has the capability to bind to B. Therefore, according to the informal semantics discussed previously, the interaction causes the resulting complex $A : B$ to be positioned in compartment *in*. Namely, term Q_1 can perform the following transition:

$$Q_1 \xrightarrow{A@out \leftrightarrow B@in} \mu.\Big((\{\xrightarrow{B@in} \mu\}.A@out) : (\{\}.B@in)\Big)\,@in$$

where the name of the resulting complex is $(A@out : B@in)@in$. Instead, if such a capability is given to B instead of A, then the resulting complex is put in compartment

out. That is, given $Q_2 = \{\}.A@out \mid \{\xrightarrow{A@out} \mu\}.B@in$, the following transition is possible:

$$Q_2 \xrightarrow{A@out \leftrightarrow B@in} \mu.\Big((\{\}.A@out) : (\{\xrightarrow{A@out} \mu\}.B@in)\Big)\ @out$$

In this case, the name of the resulting complex is $(A@out : B@in)@out$. Finally, note that if both $A@out$ and $B@in$ possess the capability of binding to the other element, then both the complexes with names $(A@out : B@in)@in$ and $(A@out : B@in)@out$ are possible.

Therefore, in the MIM calculus with compartments, as regards to the capabilities of creation of bonds, $\longrightarrow$ and $==$, the location of the resulting complex depends on which species has the capability to bind to the other.

3.5.2 Formal semantics

The formal definition of the semantics of the MIM calculus with compartments follows. We assume the set of actions *Act* to be extended to the new syntax.

Definition 3.5.2. The semantics of the *MIM calculus with compartments* (*MIMc-co*) is the relation $\xrightarrow{\alpha}$ on processes such that:

$$P \xrightarrow{\alpha} P' \iff \exists \iota \subset \mathcal{N}.\ P \xrightarrow{(\emptyset,\iota)\ \alpha} P' \tag{3.33}$$

where $\alpha \in Act$ is an action that represents the capability of P used for the reduction step, and $\xrightarrow{(\nu,\iota)\ \alpha}$, with $\nu, \iota \subset \mathcal{N}$, is the least relation on processes, closed under structural congruence $\equiv_P$, and satifying the inference rules shown in Figure 3.10.

The inference rules shown in Figure 3.10, defining the semantics of the calculus, are a straightforward extension of those defined for the basic calculus (Figure 3.7). Note that rules 3.24 and 3.28 put the resulting complex in the same compartment as the one containing the target of the binding, identified by the name h_2 in both cases.

3.5.3 Example of G protein signalling

In this section we develop a model of the G protein signalling pathway, which shows how the features for spatial modelling provided by the MIM calulus with compartments can be used. The G proteins are signal transducers, that enable a cell to react to external stimuli, such as hormones and neurotransmitters, by regulating some internal processes. Recall that the most important distinction with respect to the original definition of the MIM calculus is in the behaviour of the two binding operators, namely for the creation of non-covalent bonds $\longrightarrow$, and for covalent bonds $==$.

Figure 3.11 shows an explicit MIM diagram depicting the interactions among elements involved in the G protein signalling process, adapted from [52]. A G protein is composed of $G\alpha$ and $G\beta\gamma$ elements, which are attached to the internal surface of the cell membrane, and can be activated by *receptors* embedded in the membrane. The *G protein-coupled receptor* is represented in the diagram by the element GPR. The receptor GPR is composed of an extracellular (external) receptor domain, and a cytoplasmic (internal) domain, through which it can interact with both the external and internal elements. It receives external stimuli, and provoke some internal modification as a result.

$$\frac{\mu_1 = \{X + (\nu,\iota) \xrightarrow{\lfloor S_2@h_2 \rfloor} \mu\} \qquad \alpha = \lfloor S_1@h_1 \rfloor \leftrightarrow \lfloor S_2@h_2 \rfloor}{\mu_1.S_1@h_1 \mid \mu_2.S_2@h_2 \xrightarrow{(\nu,\iota)\ \alpha} \mu.(\mu_1.S_1@h_1 : \mu_2.S_2@h_2)@h_2} \tag{3.24}$$

$$\frac{\alpha = \lfloor S_1 \rfloor \nleftrightarrow \lfloor S_2 \rfloor}{\mu.(S_1 : S_2) \xrightarrow{(\emptyset,\emptyset)\ \alpha} S_1 \mid S_2} \tag{3.25}$$

$$\frac{\mu = \{X + (\nu,\iota) \longrightarrow P\} \qquad \alpha = \lfloor S_1@h_1 \rfloor \longrightarrow \lfloor P \rfloor}{\mu.S_1@h_1 \xrightarrow{(\nu,\iota)\ \alpha} P} \tag{3.26}$$

$$\frac{\mu = \{X + (\nu,\iota) \longrightarrow\!\triangleright P\} \qquad \alpha = \lfloor S_1@h_1 \rfloor \longrightarrow\!\triangleright \lfloor P \rfloor}{\mu.S_1@h_1 \xrightarrow{(\nu,\iota)\ \alpha} \mu.S_1@h_1 \mid P} \tag{3.27}$$

$$\frac{\mu_1 = \{X + (\nu,\iota) \xlongequal{\lfloor S_2@h_2 \rfloor} \mu\} \qquad \alpha = \lfloor S_1@h_1 \rfloor == \lfloor S_2@h_2 \rfloor}{\mu_1.S_1@h_1 \mid \mu_2.S_2@h_2 \xrightarrow{(\nu,\iota)\ \alpha} \mu.\overline{(\mu_1.S_1@h_1)(\mu_2.S_2@h_2)}@h_2} \tag{3.28}$$

$$\frac{\mu = \{X + (\nu,\iota) \overset{\lfloor \overline{S_1 S_2}@h_3 \rfloor}{\rightsquigarrow}\} \qquad \alpha = \lfloor S@h \rfloor \rightsquigarrow \lfloor \overline{S_1 S_2}@h_3 \rfloor}{\mu.S@h \mid \mu'.\overline{S_1 S_2}@h_3 \xrightarrow{(\nu,\iota)\ \alpha} \mu.S@h \mid S_1 \mid S_2} \tag{3.29}$$

$$\frac{\mu_1 = \{X + (\nu,\iota) \xRightarrow{q} \mu\} \qquad \alpha = q \Longrightarrow \lfloor S_1@h_1 \rfloor}{\mu_1.S_1@h_1 \xrightarrow{(\nu,\iota)\ \alpha} \mu.\overline{q(\mu_1.S_1@h_1)}@h_1} \tag{3.30}$$

$$\frac{\mu = \{X + (\nu,\iota) \overset{\lfloor \overline{q\ S_2}@h_3 \rfloor}{\rightsquigarrow}\} \qquad \alpha = \lfloor S_1@h_1 \rfloor \rightsquigarrow \lfloor \overline{q\ S_2}@h_3 \rfloor}{\mu.S_1@h_1 \mid \mu'.\overline{q\ S_2}@h_3 \xrightarrow{(\nu,\iota)\ \alpha} \mu.S_1@h_1 \mid S_2} \tag{3.31}$$

$$\frac{P \xrightarrow{(\nu,\iota)\ \alpha} P' \qquad \lfloor Q \rfloor \cap \iota = \emptyset \qquad \nu' = \nu \setminus \lfloor Q \rfloor}{P \mid Q \xrightarrow{(\nu',\iota)\ \alpha} P' \mid Q} \tag{3.32}$$

Figure 3.10: The inference rules defining the semantics of the MIM calculus with compartments.

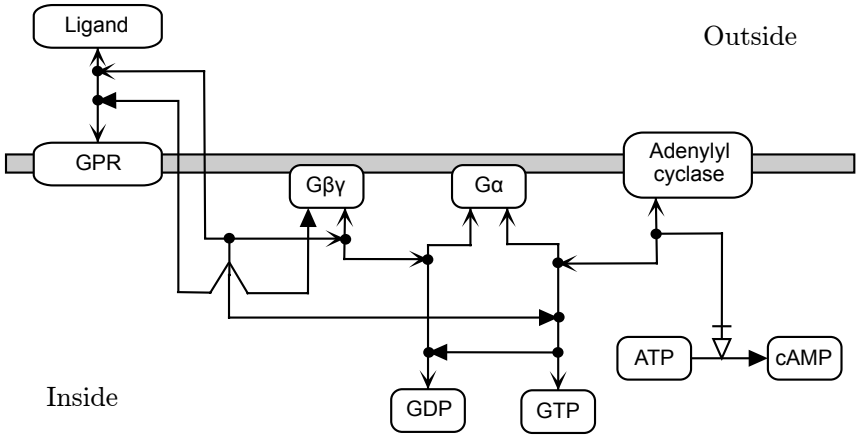


Figure 3.11: Molecular Interaction Map of the G protein signaling pathway.

A G protein is composed of three subunits, identified by the symbols α, β and γ. The α subunit is represented in the MIM diagram by the element named $G\alpha$. The β and γ subunits of the G protein form tight bonds which cannot dissociate under normal conditions, therefore the dimer $G\beta\gamma$ is considered as an elementary molecule in the model.

The α subunit of the G protein can bind to either GDP or GTP. When the $G\alpha$ is bound to the GTP, it is the active state, while when bound to GDP is inactive. On the one hand, the active G-protein complex $G\alpha : GTP$ can bind to the *adenylyl cyclase* protein on the membrane, which in turn causes the stimulation of the conversion of ATP into *cyclical-AMP*, denoted by $cAMP$. Actually, we model such a stimulation using the strongest constraint of *requirement*, as depicted by the arrow symbol shown in Figure 3.4b. On the other hand, $G\alpha : GTP$ complex can slowly convert to the inactive complex $G\alpha : GDP$. The $G\alpha$ cannot normally bind GTP (not shown in the diagram), therefore the activated form $G\alpha : GTP$ can be obtained only when an inactive complex $G\alpha : GDP$ exchanges the GDP for GTP.

In resting state, the inactive complex $G\alpha : GDP$ binds to the other subunits $G\beta\gamma$, thus obtaining the complex $(G\alpha : GDP) : G\beta\gamma$. Unless the receptor GPR is activated by the binding of a ligand, such as a hormone, to its the extracellular domain, the inactive G protein complex $(G\alpha : GDP) : G\beta\gamma$ cannot bind to the receptor (actually, to its internal domain). On the other hand, when GPR is activated, it binds to $(G\alpha : GDP) : G\beta\gamma$ which, in turn, stimulates the conversion of $G\alpha : GDP$ into $G\alpha : GTP$. This last transformation is depicted by a forked conversion line, which points from the complex $((G\alpha : GDP) : G\beta\gamma) : (GPR : Ligand)$ to both $G\alpha : GTP$, and the other elements GPR, $G\beta\gamma$.

The model The various elements involved in the G protein signalling are described in the following model of the MIM calculus with compartments. There are four compartments defined, i.e. $\mathcal{H} = \{out, m, cyt, in\}$, which represent the following compartments:

- *out*: denoting the outside of the cell, which contains only the element *Ligand*;
- m: denoting the membrane of the cell, which contains the two element which are able to interact both with the inner space and with the outer space, namely the

receptor GPR and the adenylyl cyclase;

- *cyt*: which denotes the cytoplasmic (internal) surface of the membrane, to which the G protein is attached, i.e. the elements $G\alpha$ and $G\beta\gamma$;
- *in*: denoting the internal space of the cell, which contains the other elements GDP, GTP, ATP, and $cAMP$.

The ligand is described by the term $\mu_{Ligand}.Ligand@out$, having capabilities $\mu_{Ligand} = \{\xrightarrow{GPR@m} \varnothing\}$, while the receptor is described by $\varnothing.GPR@m$. Note that, according to the semantics of the binding operators (Definition 3.5.2), since only the ligand has the capability to create a complex with the receptor, then the resulting complex will be positioned in compartment m. In fact, the semantics allows deriving the following transition:

$$\mu_{Ligand}.Ligand@out \mid \varnothing.GPR@m \xrightarrow{(\emptyset,\emptyset)\ \alpha} ((\mu_{Ligand}.Ligand@out) : (\varnothing.GPR@m))@m$$

with action $\alpha = Ligand@out \leftrightarrow GPR@m$.

The G protein is attached to the internal surface of the cell membrane, therefore the subunits are described by terms $\varnothing.G\alpha@cyt$ and $\varnothing.G\beta\gamma@cyt$. Their capabilities are empty since the complexes resulting form interactions with other elements remain attached to the membrane surface, i.e. in compartment *cyt*.

As regards GDP and GTP, they are described by terms $\mu_{GDP}.GDP@in$ and $\varnothing.GTP@in$. The capabilities of $GTP@cyt$ are empty, since $G\alpha$ cannot normally bind GTP.

As regards GDP, it is defined by the term $\mu_{GDP}.GDP@in$, where its capabilities are as follows:

$$\begin{aligned}
\mu_{GDP} &= \left\{ \xrightarrow{G\alpha@cyt} \mu_{G\alpha GDP} \right\} \\
\mu_{G\alpha GDP} &= \left\{ \xrightarrow{G\beta\gamma@cyt} \left\{ \xrightarrow{((GPR@m):(Ligand@out))@m} \mu_{GPRact} \right\} \right\} \\
\mu_{GPRact} &= \Big\{ \longrightarrow \varnothing.((\varnothing.GPR@m) : (\mu_{Ligand}.Ligand@out))@m \mid \varnothing.G\beta\gamma@cyt \mid \\
&\qquad \mu_{G\alpha GTP}.((\varnothing.G\alpha@cyt) : (\varnothing.GTP@in))@cyt \Big\}
\end{aligned}$$

$GDP@in$ can bind to G protein subunit $G\alpha@cyt$, which in turns binds to $G\beta\gamma@cyt$ and then to the activated receptor, described by the complex $(GPR : Ligand)@m$. The position of the complexes are precisely determined by which element has the capability of binding to the other, between each pair of elements which can bind. In fact, complexes $G\alpha : GDP$ and $(G\alpha : GDP) : G\beta\gamma$ end up into the *cyt* compartment, namely they are attached to the internal membrane surface. Instead, when complex $((G\alpha : GDP) : G\beta\gamma)@cyt$ binds to the activated receptor, the resulting complex is put in compartment m, since we assume that the GPR receptor is kept in its place.

The resulting complex, whose capabilities are denoted by μ_{GPRact}, can be converted back to (i) the complex $(GPR : Ligand)@m$, (ii) the G protein subunits $G\beta\gamma@cyt$, and (iii) the activated G protein subunit $(G\alpha : GTP)@cyt$. This models the expected behaviour of the exchange of GDP for GTP, performed by the G protein once activated by the receptor.

Despite that $G\alpha$ cannot normally bind GTP, we have seen that the activated G protein subunit $(G\alpha : GTP)@cyt$ can nevertheless be obtained once the receptor is activated. The capabilities of such a complex are the following:

$$\mu_{G\alpha GTP} = \left\{ \longrightarrow \mu_{G\alpha GDP}.(\varnothing.G\alpha@cyt : \mu_{GDP}.GDP@in)@cyt + \xrightarrow{AdCyc@m} \varnothing \right\}$$

Complex $(G\alpha : GTP)@cyt$ can either convert back to the inactive form $(G\alpha : GDP)@cyt$ or bind to the adenylyl cyclase $AdCyc@m$. When it binds to the adenylyl cyclase, it also enables the conversion of $ATP@in$ to $cAMP@in$, as formally defined by the following term:

$$\left\{ (\nu_1, \emptyset) \longrightarrow \varnothing.cAMP@in \right\}.ATP@in$$

where the set of promoters is $\nu_1 = \{((G\alpha@cyt : GTP@in) : AdCyc@m)@m\}$.

Chapter 4

Spatial P systems

In the field of Membrane Computing, the *P systems* formalism [69] has been proposed as a distributed and parallel computing model inspired by the structure and the functioning of the living cell. A P system is composed of a hierarchy of membranes, each of them containing a multiset of *objects*, which are processed by evolution rules. Evolution rules allow the description of the behaviour of a model, for example by representing chemical reactions. The basic P systems model associates a set of rules with each membrane, which are to be applied to the (only) objects contained in the same membrane. A rule specifies *reactants* and *products*: when a rule is applied, the reactants are removed from the membrane and the products are either (i) added inside the membrane, (ii) sent into an inner membrane, or (iii) sent outside the membrane.

Many variants and extensions of P systems exist that include features which increase their expressiveness and which are based on different evolution strategies. In particular, we cite P systems with promoters and inhibitors [20], and P systems with priority among evolution rules [69], since Spatial P systems allow the specification of promoters, and provide a form of priority among rules. In P systems with promoters and inhibitors, the applicability of each rule can be constrained by the presence or absence of other objects in the membrane. In particular, the multiset of promoters represent the objects which are needed for the rule to be applicable. On the contrary, the multiset of inhibitors represent those objects whose presence effectively blocks the application of the rule. As regards P systems with priority among evolution rules, a rule is applicable in a membrane only if no higher priority rule is applicable, namely only if each higher priority rule requires a multiset of reactants which are not present in the membrane. See [70] for the definition of variants of P systems, and [68] for a complete bibliography.

In this chapter, we present an extension of P systems, in which objects and membranes are embedded into a two-dimensional discrete space. We call this extension Spatial P systems. Objects are associated with precise positions inside membranes, and evolve by means of the application of evolution rules. As for standard P systems, rules specify the objects which are consumed and the ones which are produced. Moreover, promoters can be specified in the rules. In addition, rules specify the relative positions of the various objects involved, with respect to the position in which the rule is applied. A feature of Spatial P systems is the distinction between *ordinary* objects and *mutually exclusive* objects. Every position inside a membrane can accommodate an arbitrary number of ordinary objects, but at most one mutually exclusive object. Finally, the calculus also allows a priority

relation among evolution rules in a membrane, described by a partial order.

Spatial P systems use a representation of the space analogous to that of Cellular Automata (CA) [61]. However, in Spatial P systems, cells contain objects, and interactions occur between objects. For this reason, each position in a Spatial P system can accommodate any number of objects, thus representing a possibly infinite number of different "states".

In this chapter we study the computational universality of the calculus when using different features. In particular, we study the computational power of Spatial P systems when using only non-cooperating rules, namely when evolution rules are restricted to have only *one* reactant. This is a severe limitation, which makes standard P systems as powerful as context free grammars, therefore not universal. We prove that the feature of mutually exclusive objects is sufficient to achieve computational universality even when only non-cooperating rules are allowed. In order to illustrate the use of the calculus we present a model of the evolution of populations in the presence of geographical separations.

In the next chapter, we are going to investigate the problem of simulating some restricted kinds of Spatial P system models. This is an interesting problem, which enables a practical use of the formalism for the analysis of spatial models.

4.1 Background

In this section we recall the definition of standard P systems [70], along with the extension with promoters and inhibitors, and the extension with priority among evolution rules. These two extensions are particularly interesting because such features are also provided by Spatial P systems. Moreover, we recall some known results on the computational power of the different versions of P systems presented. Finally, we also recall the definition and computational results of matrix grammars, a computational device that we use to prove the universality of Spatial P systems. Formal semantics of different versions of P systems are presented in [22, 1, 12, 11, 13, 14].

In the following, we often denote multisets over a finite alphabet as strings. More precisely, let V^* be the set of all strings over an alphabet V, including the empty one, denoted by λ. For $a \in V$ and $x \in V^*$ we denote by $|x|_a$ the number of occurrences of a in x. Given an alphabet $V = \{a_1, \ldots, a_n\}$, where the ordering is important, the *Parikh mapping* of a string $x \in V^*$ is defined as $\Psi_V(x) = (|x|_{a_1}, \ldots, |x|_{a_n})$, called the *Parikh vector*. That is, given a string $x \in V^*$, representing a multiset over V, the Parikh vector $\Psi_V(x)$ gives the multiplicities in x of each symbol. The definition is also extended to languages, namely, given a language $L \subseteq V^*$, the *Parikh image* of L is defined as $\Psi_V(L) = \{\Psi_V(x) \mid x \in L\}$. Given a family of languages X, the family of Parikh images of languages in X is denoted as PsX.

Finally, given a multiset $\mathcal{M}$, the multiplicity of an element x in $\mathcal{M}$ is denoted by $\mathbf{m}(\mathcal{M}, x)$. We use the following notations for operations on multisets, where the multiplicity is significant. The union operator between multisets is denoted by $\uplus$, while the difference operator is denoted by $\setminus\!\setminus$. Inclusion between two multisets $\mathcal{A}, \mathcal{B}$ is denoted by $\mathcal{A} \Subset \mathcal{B}$, meaning that $\mathcal{A}$ is a *submultiset* of $\mathcal{B}$. Proper inclusion, namely if $\mathcal{A} \Subset \mathcal{B}$ and $\mathcal{A} \neq \mathcal{B}$, is denoted by $\Subset$.

4.1.1 P systems

A P system is composed of a hierarchy of membranes, each of them containing a multiset of *objects*, which are processed by evolution rules. We assume membranes to be labelled by natural numbers. Evolution rules describe how the objects of the system evolve, for example they can be used to describe chemical reactions, that is rules in which some objects interact and, as a result, they are transformed into some other objects. A set of rules is associated with each membrane. Given a membrane m, its evolution rules in the set R_m can be applied only to the objects contained in the same membrane, and not in any other membrane. In particular, a rule in a membrane m cannot be applied to objects contained in any child membrane.

A rule is of the form $u \to v$, where u denote the multisets the *reactants*, and v denote the *products*. When a rule is applied, the reactants are removed from the membrane and the products are added to the target membrane, which could be a different membrane than the one in which the rule is applied. Formally, given a membrane m, the products of a rule associated with m are described by a multiset of tuples of the following forms:

- $(a, here)$, usually written as a_{here}, meaning that the object a is added to the same membrane m;
- (a, out), usually written as a_{out}, meaning that the object a is to be sent out of the membrane;
- (a, in_x), usually written as a_{in_x}, meaning that the object a is to be sent into the child membrane labelled by x.

The target indication *here* is often omitted, therefore a rule of the form $u \to v$ is to be interpreted as if all the objects v have target indication *here*.

A characteristic of P systems is the way in which rules are applied in each step, namely with *maximal parallelism*. In each step, evolution rules are applied in a maximal non-deterministic way in all membranes, that is, in each membrane, a multiset of rules is selected non-deterministically to consume the membrane objects, in such a way that no other rule can be applied to the objects which are not involved in any rule application.

Let TAR be the set of object targets $\{here, out\} \cup \{in_i \mid i \in \mathbb{N}\}$, and let $V_{tar} = V \times TAR$. Formally, an evolution rule $u \to v$ is such that $u \in V^*$ and $v \in V_{tar}^*$. The number of reactants of a rule, that is the length of u, is called the *radius* of such a rule. An evolution rule is said to be *cooperating* if its radius is greater than 1, otherwise the rule is called *non-cooperating*. This naming is also extended to P system models, that is a non-cooperating P system is such that all its rules are non-cooperating, otherwise it is a *cooperating* P system.

A particular form of cooperating systems is that of *catalytic systems*, which are meant to capture the biological notion of catalysts, namely biological elements which are needed for a reaction to happen and directly participate in it, but they are neither consumed nor modified by the reaction. Formally, in catalytic P systems, a subset of objects $K \subseteq V$ denotes the set of *catalysts*, whose use in the rules is constrained. Namely, there can be two kinds of rules, either non-cooperating rules of the form $a \to v$, or rules with *catalysts* of the form $ca \to cv$; where $c \in K$, $a \in V \setminus K$, and $v \in (V \setminus K)_{tar}^*$.

Definition 4.1.1. A *P system* is a tuple $\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ where:

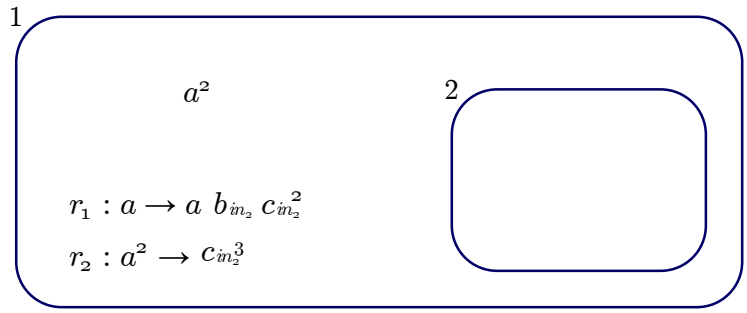


Figure 4.1: An example of P system model.

- V is a finite *alphabet* whose elements are called *objects*;
- $\mu \subset \mathbb{N} \times \mathbb{N}$ describes the tree-structure of membranes, where $(i, j) \in \mu$ denotes that the membrane labelled by j is contained in the membrane labelled by i;
- w_i, with $1 \leq i \leq n$, are strings from V^* representing multisets over V associated with membranes $1, 2, \ldots, n$ of μ;
- R_i, with $1 \leq i \leq n$, are finite sets of *evolution rules* associated with membranes $1, 2, \ldots, n$ of μ.

A sequence of transitions between configurations of a given P system Π is called a *computation*. A computation is *successful* if and only if it reaches a configuration in which no rule is applicable. The result of a successful computation can be defined in different ways. There are two common approaches, which correspond either to consider the multiset of objects sent out of the skin membrane during the computation, or the multiset of objects present in a particular membrane, denoted as the *output* membrane. Unsuccessful computations are those computations which never halt, thus yielding no result.

Given a P system Π whose set of objects is V, let $U \subseteq V$ be the set of objects that can be sent out of the skin membrane, namely objects that appear in the right-end side of a rule in R_1 with target *out*. The result $x \in U^*$ of a computation of Π can be mapped into a vector of natural numbers by the Parikh mapping $\Psi_U(x)$. The set of all vectors of natural numbers computed by Π is denoted $Ps(\Pi)$.

Following [70], let us denote by $P_n(\alpha)$ the class of P Systems with at most $n \geq 1$ membranes and using rules of type α, where $\alpha = coo$ indicates that cooperating rules are allowed, $\alpha = cat$ that only catalytic and non-cooperating rules are allowed, and $\alpha = ncoo$ that only non-cooperating rules are used. When the number of membranes is not bounded we replace n with $*$. Finally, $PsP_n(\alpha)$ denotes the family of sets of vectors of natural numbers computed by P systems of class $P_n(\alpha)$.

Example 4.1.1. Figure 4.1 depicts a P system with two membranes 1 and 2. The rules r_1 and r_2 are associated with membrane 1, while membrane 2 has no rules associated with it. An application of rule $r_1 = a \rightarrow a\, b_{in_2}\, c_{in_2}^2$ causes a copy of object b, and two copies of object c, to be sent into the inner membrane 2. Note that we have used an exponential notation for denoting multiple copies of the same tuple c_{in_2} in a compact way. The object a is still present after the application, since it appears in the right-hand part of the rule.

Rule $r_2 = a^2 \to c^3_{in_2}$, instead, can be applied to a pair of objects a, and results in sending three copies of the object c into membrane 2. The initial state, as depicted, contains two copies of object a in membrane 1, and no objects in membrane 2.

At the first step, either rule r_1 or r_2 is applied. In fact, both the rules are enabled, since their reactants are present in the membrane. Actually, if r_1 is applied to an object a, then the maximality requires it to be applied also to the other copy of s, since in that case rule r_2 could no be applied anymore. This application sends the objects bcc into membrane 2. The objects contained in membrane 1 remain aa after the application, therefore the double application of rule r_1 can be repeated in the subsequent step. Whenever rule r_2 is applied, it causes the two copies of a in membrane 1 to disappear, thus terminating the computation. In such a case, the objects ccc are sent into membrane 2. Therefore, any computation of this P system is composed of a sequences of steps in which only r_1 is applied (twice per step), followed by a last step in which rule r_2 is applied once. Therefore, whenever the P system terminates, membrane 2 contains a multiset of objects $b^k\, c^{2k+3}$, for some $k \geq 0$.

4.1.2 Computational power

We recall some fundamental results on the computational power of the basic class of P systems, by comparing it with languages in the Chomsky hierarchy. We denote by CF, CS, and RE, the families of languages generated by *context free*, *context sensitive*, and *arbitrary* grammars, respectively. Recall that $CF \subset CS \subset RE$, which is a part of the *Chomsky hierarchy*. Note that RE exactly corresponds to the family of *recursively enumerable* languages, namely the family of languages recognized by Turing machines, hence the name.

Technically, the output of a P system, namely a multiset of symbols, is not directly comparable with a language, namely a set of strings, generated by a Chomsky grammar. One approach to compare them is consider, on the one hand, the cardinalities of the multisets of symbols generated by a class of P systems and, on the other hand, the length sets of languages of a given a family of languages. Given a family of languages X, the family of length sets of languages in X is denoted as NX. In this case, it holds that $NCF \subset NCS \subset NRE$ [70]. Another approach is to consider Parikh vectors, which means considering the strings of a language as representing multisets. Clearly, $PsCF \subset PsCS \subset PsRE$.

The first two results that we report, without the proofs, concern the power of cooperation (for the proofs, see [70]). The first result shows that P systems with non-cooperating rules are not universal, namely they are equivalent to context free grammars. Moreover, they are also equivalent to P systems with only one membrane, namely the membrane hierarchy collapses to only one membrane.

Theorem 4.1.1 ([70]). *$PsP_*(ncoo) = PsP_1(ncoo) = PsCF$*

In case of cooperating rules, the following theorem shows that when rules of such a form are allowed, then computational universality is achieved.

Theorem 4.1.2 ([70]). *$PsP_*(coo) = PsP_m(coo) = PsRE$, for all $m \geq 1$*

Actually, allowing only the use of catalysts in rules is sufficient to obtain computational universality, as shown by the following theorem.

Theorem 4.1.3 ([83]). *$PsP_m(cat) = PsRE$, for all $m \geq 1$*

4.1.3 P systems with priorities among evolution rules

P systems can be equipped with a priority relation among evolution rules. The priority among rules is described by the partial order relations $\rho_i \subseteq R_i \times R_i$, for each $i \in \{1, \ldots, n\}$. Given the partial order ρ, for some membrane, a pair $(r_1, r_2) \in \rho$, with $r_1 \neq r_2$, means that the rule r_2 has higher priority than r_1, and is usually written as $r_1 < r_2$. The applicability of rules is constrained as follows. A rule r_1 can be applied only if no other higher priority rule $r_2 > r_1$ is enabled in the same step, that is if the reactants for r_2 are not available at the beginning of the step. Note that, even if the reactants of rule r_2 are present, it can happen that such a rule is not applied, for example because there is another non-comparable rule r_3 (i.e. neither $r_2 < r_3$ nor $r_3 < r_2$) competing with r_2 for the same reactants. Even in such a case, any lower priority rule $r_1 < r_2$ still cannot be applied.

Example 4.1.2. Consider the rules $r_1 = ab \to c$, $r_2 = a \to d$, $r_3 = b \to e$, with relative priorities $r_1 > r_3$, namely $\rho = \{(r_1, r_1), (r_2, r_2), (r_3, r_3), (r_3, r_1)\}$. Given a multiset of objects bb, the only possible state that can be obtained after one step is ee, resulting from the application of rule r_3 twice. Note that r_3 could be used since rule r_1 could not, as the reactants ab are not present in the given state. Instead, given the objects aab, the following states can be obtained: cd, by applying both r_1 and r_2 once; and ddb, by applying r_2 twice. Note that rule r_3 cannot be applied, even if the object b is not used by any rule, since the higher priority rule r_1 could be applied in the given initial state aab.

Computational power

For completeness, we report the following result on the computational power of P systems with priorities, proved in [81]. In this case, $PsET0L$ denotes the Parikh images of $ET0L$ languages, a kind of Lindenmayer systems (see [70]), which is known not to be universal [70]. Precisely, $PsET0L \subset PsCS \subset PsRE$, where $PsCS$ denote the Parikh images of context sensitive languages.

Theorem 4.1.4 ([81]). *$PsP_m(ncoo, pri) = PsET0L$, for all $m \geq 1$*

4.1.4 P systems with promoters and inhibitors

P systems with promoters and inhibitors can be formalized in different ways. We consider P systems with promoters and inhibitors at the level of sets of rules, as defined in [20, 70]. In such a model, evolution rules are extended to specify, for each rule, two multisets of objects denoted as *promoters* and *inhibitors*. They are used to constrain the applicability of a rule on the basis of the objects which are present in the membrane. Precisely, a rule can be applied only if *all* the promoters, with their multiplicities, are present in the membrane, and *not all* of the inhibitors are present in the membrane [20]. A rule with promoters and inhibitors is usually denoted as $u \to v|_{x_1,\ldots,x_k,\neg y_1,\ldots,\neg y_h}$, where $u \in V^*$ and $v \in V^*_{tar}$ denote the reactants and products, respectively, while each $x_i \in V$ denote a promoter, and each y_j denote an inhibitor. Formally, given a multiset of objects w contained in a membrane, the rule $u \to v|_{x_1,\ldots,x_k,\neg y_1,\ldots,\neg y_h}$ can be applied only if $\forall i = 1, \ldots, k.\ x_i \in w \setminus\setminus u$ and $\forall j = 1, \ldots, h.\ y_j \notin w \setminus\setminus u$. Note that objects corresponding to promoters and inhibitors can evolve independently, as they can be consumed by the application of other rules.

For example, a rule $a \to b|_c$ means that an object a can be transformed to b only if there is at least one object c present in the membrane. Conversely, the rule $aa \to b|_{\neg b}$ means that two copies of a can be transformed to b only if no b is present.

Computational power

As regards the computational power of P systems with promoters and inhibitors at the level of sets of rules, we report the following results from [20]. In particular, each of both inhibitors and promoters are sufficient by themselves to obtain universality even with only non-cooperating rules. In the theorems, $proS$ and $inhS$ denote the use of promoters and inhibitors, at the level of sets of rules, respectively. Moreover, in denotes the use of a non-deterministic form of the in target. Namely target in_j is not allowed, while there can be a target in, whose meaning is that the associated object is to be sent into an child membrane chosen *non-deterministically*. Finer results are presented in [20].

Theorem 4.1.5. *([20])* $PsP_3(ncoo, in, inhS) = PsRE$

Theorem 4.1.6. *([20])* $PsP_1(ncoo, proS) = PsRE$

4.1.5 Matrix grammars with appearance checking

We recall from [70] the definition of matrix grammars with appearance checking, a popular tool which has been used to prove the computational universality of many types of P systems. We use matrix grammars with appearance checking to prove the computational universality of Spatial P systems without cooperating rules.

A *(context-free) matrix grammar with appearance checking* is defined as a tuple $G = (N, T, S, M, F)$, where N and T are disjoint alphabets of non–terminals and terminals, respectively, $S \in N$ is the axiom, M is a finite set of matrices, namely sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$ of context–free rules over $N \cup T$ with $n \geq 1$, and F is a set of occurrences of rules in the matrices of M. For a string w, a matrix $m : (r_1, \ldots, r_n)$ can be executed by applying its rules to w sequentially in the order in which they appear in m. Rules of a matrix occurring in F can be skipped during the execution of the matrix if they cannot be applied, namely if the symbol in their left–hand side is not present in the string.

Formally, given $w, z \in (N \cup T)^*$, we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*$ with $1 \leq i \leq n+1$ such that $w = w_1$, $z = w_{n+1}$ and, for all $1 \leq i \leq n$, either (1) $w_i = w'_i A_i w''_i$ and $w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or (2) $w_i = w_{i+1}$, A_i does *not* appear in w_i and the rule $A_i \to x_i$ appears in F. Thus, in case (2) a matrix can be applied even if some of its rules are not applicable, provided that these rules are listed in F. We remark that F consists of *occurrences* of rules in M, that is, if the same rule appears several times in the matrices, it is possible that only some of these occurrences are contained in F.

The language generated by a matrix grammar G is $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$, where $\Longrightarrow^* w$ is the reflexive and transitive closure of $\Longrightarrow$. As regards the computational power, matrix grammars with appearance checking are universal [70].

Let $|x|$ denote the length of the string x. A matrix grammar with appearance checking $G = (N, T, S, M, F)$ is said to be in *binary normal form* if $N = N_1 \cup N_2 \cup \{S, \#\}$, with these sets mutually disjoint, and the matrices in M are of the forms:

1. $(S \to XA)$, with $X \in N_1, A \in N_2$;
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$;
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$;
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2, x \in T^*, |x| \leq 2$.

Moreover, there is only one matrix of type 1 and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3. We remark that # is a trap symbol, namely once introduced it cannot be removed, and a matrix of type 4 is used only once, in the last step of a derivation.

For each matrix grammar (with or without appearance checking) there exists an equivalent matrix grammar in binary normal form. A matrix grammar with appearance checking in binary normal form is always given as $G = (N, T, S, M, F)$, with $N = N_1 \cup N_2 \cup \{S, \#\}$ and with $n + 1$ matrices in M, injectively labelled with $m_0, m_1, \ldots, m_n$. The matrix $m_0 : (S \to X_{init} A_{init})$ is the initial one, with X_{init} a given symbol from N_1 and A_{init} a given symbol from N_2; the next k matrices are without appearance checking rules, $m_i : (X \to \alpha, A \to x)$, with $1 \leq i \leq k$, where $X \in N_1, \alpha \in N_1 \cup \{\lambda\}, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$ (if $\alpha = \lambda$, then $x \in T^*$); the last $n - k$ matrices have rules to be applied in the appearance checking mode, $m_i : (X \to Y, A \to \#)$, with $k + 1 \leq i \leq n, X, Y \in N_1$, and $A \in N_2$.

We remark that in matrix grammars in binary normal form we can assume that all symbols $X \in N_1$ and $A \in N_2$ appear as the left-hand side of a rule from a matrix: otherwise, the derivation is blocked after introducing such symbols, hence we can remove these symbols and the matrices involving them.

4.2 Spatial P systems

Spatial P systems extend standard P systems by embedding membranes and objects in the two-dimensional discrete space with natural coordinates $\mathbb{N}^2$. Each object in a Spatial P system model is associated with a position inside a membrane. Membranes can have any shape, as long as they are defined as simple closed curves composed only of horizontal and vertical edges, and need to be properly nested. Sibling membranes still must not overlap, and membranes cannot exceed the bounds of their parent membrane. A membrane is described by the circular list of positions through which it passes, where each position has to be adjacent to both the previous and the next position in the list. Positions forming membrane edges are considered to be inside the membrane. There is always a distinguished skin membrane, which contains all other membranes and objects. We assume the skin membrane to be labelled with 1.

Figure 4.2 shows the spatial structure of membranes and objects of Spatial P system, composed of only rectangular membranes. The square cells of the grid correspond to positions, and evolution rules associated with membranes are not shown. Membrane 1, having width 8 and height 5, contains two membranes labelled 2 and 3. Assuming that the bottom-left position of membrane 1 corresponds to position $(0, 0)$, then membrane 1 is described by the sequence of positions $\langle(0,0),(0,1),\ldots,(0,4),(1,4),\ldots,(7,4),(7,3),\ldots,(7,0),(6,0),\ldots,(1,0)\rangle$. Membranes 2 and 3 are similarly described. As regards the objects, there are three objects a contained in membrane 2, at positions $(2, 2)$,$(3, 2)$

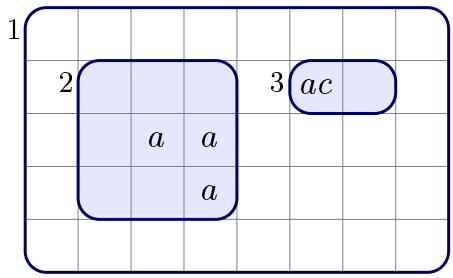


Figure 4.2: The spatial structure of membranes and objects of a Spatial P system.

and $(3, 1)$. Membrane 3 contains an object a and an object c in position $(5, 3)$. All other positions, in all membranes, are empty.

In order to increase the usefulness of the formalism, we include into Spatial P systems two features: promoters and priorities. Promoters are useful for describing rules in which the presence of some object enables the rule to be applied. A notion of priority among the rules, instead, allows a proper description of the movement.

Priority among evolution rules has a different meaning with respect to the usual semantics of P systems. Recall that, in standard P systems, a rule can be applied in a membrane only if there is no higher priority rule which is enabled, that is the reactants of each higher priority rule are not present in the membrane region at the beginning of the step. Instead, in Spatial P systems we permit lower priority rules to be applied even if there are higher priority rules whose reactants are present. This difference is motivated by the fact that, in Spatial P systems, because of the presence of mutually-exclusive objects, it can happen that a rule for which the reactants are present cannot be actually applied, since that would create a conflict among mutually-exclusive objects. Precisely, the semantics of Spatial P systems requires that there does not exist a set of positions in which it is possible to replace the multisets of rules selected in each of them with "greater" multisets, and still obtain a selection of rules which is applicable with respect to the reactants, and which does not create conflicts among mutually-exclusive objects. In this context, a greater multiset means increasing the multiplicity of application of any rule r, given the possibility to cancel the application of any lower priority rule $r' < r$.

The formal description of membranes in a Spatial P system model is composed of the following elements, where we assume that membranes are labelled by $\{1, \ldots, n\}$:

- a *tree structure* describing the containment hierarchy among membranes, represented by the set $\mu \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$, where $(i, j) \in \mu$ denotes that the membrane labelled by j is contained in the membrane labelled by i;

- a function $\sigma : \{1, \ldots, n\} \to \langle \mathbb{N}^2 \rangle$, which describes membrane bounds by associating to each membrane $i \in \{1, \ldots, n\}$ a sequence of *distinct* positions $\sigma(i) = \langle p_1, \ldots, p_s \rangle$ through which the membrane edges passes.

In order to ensure that the membrane is composed only of horizontal and vertical edges, each position p_i must be adjacent to the next one p_{i+1}, and also the last one p_s and the first one p_1 must be so. The condition that all positions must be distinct ensures that the membrane defines a simple closed curve. We also require each membrane to be properly contained in their parent membrane (except for the outer membrane), and

that positions forming membrane bounds of different membranes are distinct: $\forall i_1, i_2.\ i_1 \neq i_2 \implies \sigma(i_1) \cap \sigma(i_2) = \emptyset$. This last constraints ensures that different membranes do not overlap.

The structure of membranes can be seen as a partition of the space bounded by the skin membrane, where a position belongs to the region of a membrane if and only if it is contained within its bounds and not contained in any other child membrane. The set of all positions belonging to a membrane is called a *region*.

Let $Extents(i)$ denote the set of positions inside the closed curve denoted by $\sigma(i)$, with $\sigma(i) \subseteq Extents(i)$ for all $i \in \{1, \ldots, n\}$. For simplicity, we also define the $Extents$ for the special value 0, representing all the position outside the skin membrane, as $Extents(0) = \mathbb{N}^2 \setminus Extents(1)$. The set of positions forming the region of a membrane is defined by the function $Region : \{1, \ldots, n\} \to \mathscr{P}(\mathbb{N}^2)$ as $Region(i) = Extents(i) \setminus \bigcup_{(i,j)\in\mu} Extents(j)$. Note that $\forall i.\ \sigma(i) \subseteq Region(i)$. The constraint that each membrane is properly contained in its parent membrane formally corresponds to require that $\forall (i, j) \in \mu.\ Extents(j) \subset Extents(i)$.

Each object in a Spatial P system model is associated with a position in the region of a membrane. The two kinds of objects, *ordinary* objects and *mutually exclusive* (*ME*) objects, are represented by two disjoint sets V and E, respectively.

A set of evolution rules is associated with each membrane, where each rule is of the following form:

$$(u_1)_{p_1} \ldots (u_k)_{p_k}[(\pi_1)_{q_1} \ldots (\pi_\gamma)_{q_\gamma}] \to (v_1)_{t_1} \ldots (v_h)_{t_h}$$

where:

- $(u_1)_{p_1} \ldots (u_k)_{p_k}$ denote the *reactants* as strings of objects u_i, each of them having a *relative* position $p_i \in \mathbb{Z}^2$; the reactants are consumed by the application of the rule; (we assume $p_1, \ldots, p_k$ to be distinct);

- $(\pi_1)_{q_1} \ldots (\pi_\gamma)_{q_\gamma}$ denote the *promoters* of the rule, whose presence is necessary in order for the rule to be applicable, but are not consumed by the application; similarly to the previous case, each π_i is a string of objects with relative position $q_i \in \mathbb{Z}^2$; (we assume $q_1, \ldots, q_\gamma$ to be distinct);

- $(v_1)_{t_1} \ldots (v_h)_{t_h}$ represent the *products* of the rule, where each v_i is a string of objects, and each t_i is either a relative position in $\mathbb{Z}^2$, or a special symbol from the set $\{out_d\} \cup \{in_{j,d} | 1 \leq j \leq n\}$.
  The symbol out_d means that the object is to be sent outside of the current membrane, along a direction $d \in D = \{(0, 1), (0, -1), (1, 0), (-1, 0)\}$. The objects sent out of the skin membrane disappear from the system. Symbol $in_{j,d}$ means that the object is to be sent into the inner membrane labelled j, along the direction $d \in D$. Position $(0, 0)$ can be denoted by *here*. (We assume $t_1, \ldots, t_h$ to be distinct.)

A generic list of reactants $(u_1)_{p_1} \ldots (u_k)_{p_k}$, promoters $(\pi_1)_{q_1} \ldots (\pi_\gamma)_{q_\gamma}$, and products $(v_1)_{t_1} \ldots (v_h)_{t_h}$ may be denoted by a symbol $\overline{u}$, $\overline{\pi}$, or $\overline{v}$, respectively. The *radius* of a rule is defined as the number of reactants, namely $\sum_{i=1}^{k} |u_i|$. An evolution rule *without promoters* is said to be *cooperating* if its radius is greater than 1, otherwise the rule is called *non-cooperating*.

Each rule describes relative positions for the involved objects and, in order to be applied, the relative positions appearing in the rule have to be instantiated to concrete positions inside membrane region. For example, consider the objects $V = \{a, b, c\}$. A rule $r = a \to (b)_{(-1,0)}\,(c)_{out_{(0,1)}}$ can be applied to an object a in a position p, and would result in an object b being put in position $p + (-1, 0)$, and an object c being sent out of the membrane, along the direction $(0, 1)$. With respect to the Spatial P system configuration depicted in Figure 4.2, assume that r belongs to membrane 2. Then rule r can be applied only to the a objects in position $(3, 1)$ and $(3, 2)$, causing the two a objects to disappear, an object b being put into both $(2, 1)$ and $(2, 2)$, and an object c being sent out of the membrane, to both the positions $(3, 2)$ and $(3, 3)$. Rule r cannot be applied to the a object in $(2, 2)$, since it is not possible to send out of the membrane the c object from that position. In fact, such an application would put a c object in position $(2, 3)$, which is inside the region of membrane 2.

Definition 4.2.1. A *Spatial P system* is described by a tuple

$$(V, E, \mu, \sigma, W_0, (R_1, \leq_1), \ldots, (R_n, \leq_n))$$

where:

- V and E are disjoint *alphabets*, denoting *ordinary objects* and *mutually exclusive objects* respectively;
- $\mu \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$ describes the tree-structure of membranes, as explained before;
- $\sigma : \{1, \ldots, n\} \to \langle \mathbb{N}^2 \rangle$ describes the shape and position of each membrane $i \in \{1, \ldots, n\}$;
- $W_0 = \{w^{(0)}_{x,y}\}_{(x,y)}$, is a set of strings $w^{(0)}_{x,y} \in (V \cup E)^*$, with $(x, y) \in Extents(1)$ being a position in the extents of the skin membrane, where each $w_{x,y}$ represents the multiset of objects associated with position (x, y) in the initial configuration of the system; a set W of this kind is called *configuration*, and must be such that each position contains at most one mutually exclusive object, namely $\forall p \in Extents(1).\ \sum_{x \in E} \mathbf{m}(w_p, x) \leq 1$.
- $(R_i, \leq_i)$, for $i \in \{1, \ldots, n\}$, describe the finite sets of *evolution rules* R_i associated with the membranes, together with a partial order relation $\leq_i$ on R_i describing the priorities among the rules; in case the priorities are not used, namely if, for every membrane i, the partial order relation is the identity relation I_{R_i}, then its indication can be omitted.

4.2.1 Semantics

In order to define the semantics, some auxiliary definitions are needed. We define the notion of *p-enabled rule*, which allows us to determine if a given rule can be considered for application in a given position p. In particular, it takes into account the resulting positions of the objects, and ensures that they would be put in the correct membrane regions with respect to their target positions.

Definition 4.2.2. Given a membrane m, and a position $p = (x, y) \in Region(m)$, an evolution rule $r = (u_1)_{p_1} \dots (u_k)_{p_k}[(\pi_1)_{q_1} \dots (\pi_\gamma)_{q_\gamma}] \to (v_1)_{t_1} \dots (v_h)_{t_h} \in R_m$ is *p-enabled* iff $\forall i = 1, \dots, h$:

- for any position $p_i \in \mathbb{Z}^2$ for reactants, and any position $q_j \in \mathbb{Z}^2$ for promoters, the relative position with respect to the current position p is contained in membrane region:

$$\forall i \in \{1, \dots, k\}.\ p + p_i \in Region(m)$$
$$\forall j \in \{1, \dots, \gamma\}.\ p + q_j \in Region(m)$$

- for any target position $t_i \in \mathbb{Z}^2$ appearing in the rule, the resulting position $p' = p + t_i$ with respect to the current position p is contained in membrane region, i.e. p' is inside membrane bounds and does not overlap with any inner membrane:

$$t_i \in \mathbb{Z}^2 \implies p + t_i \in Region(m)$$

- if the rule specifies an out_d target, then p is on the edge of the membrane, and the resulting position $p + d$ is in the region of the parent membrane:

$$t_i = out_d \implies p \in \sigma(m) \wedge p + d \in Region(father(m))$$

  where $father(j) = i$, if $\exists (i, j) \in \mu$; and $father(1) = 0$.

- if the rule specifies a target $in_{m',d}$, with m' a child membrane of m, then p is adjacent to it:

$$t_i = in_{m',d} \implies p + d \in \sigma(m')$$

The set of all p-enabled rules of a set of rules R is denoted by $Enabled(R, p)$.

In each step of the evolution of a Spatial P system, some evolution rules are chosen and applied to the system state, by removing all reactant objects and adding all the products. In particular, in each step, for every membrane m and for every position p in membrane region, a multiset of p-enabled evolution rules is chosen non-deterministically, with some constraints needed to ensure that the mutual exclusivity between objects, the priority, and the maximality are handled correctly.

The multisets of rules selected for application, for each position of a Spatial P system model, are described by a *selection*, defined in the following.

Definition 4.2.3. A *selection* of rules is described by a function S from positions to multisets of rules, such that for each position $p \in Extents(1)$, $S(p) = \mathcal{R}$ is a multiset of p-enabled rules from R_m, for the membrane m such that $p \in Region(m)$. We assume that $\forall p \in Extents(1).\ \exists \mathcal{R}.\ S(p) = \mathcal{R}$. Finally, an *empty selection* S is a selection such that $\forall (p, \mathcal{R}) \in S.\ \mathcal{R} = \emptyset$.

We introduce some auxiliary functions, which deal with the reactants, promoters and products, of the rules described by a selection. Given a rule r and a position p, let

$\eta(r,p)$, $\Pi(r,p)$, and $T(r,p)$ represent the multisets of reactants, promoters, and products, respectively, appearing in the rule in the relative position p. Formally:

$$\eta((u_1)_{p_1}\dots(u_k)_{p_k}[\overline{\pi}]\to\overline{v},p)=\begin{cases}u_i & \text{if } \exists i.\ p=p_i\\ \emptyset & \text{otherwise}\end{cases}$$

$$\Pi(\overline{u}[(\pi_1)_{p_1}\dots(\pi_\gamma)_{p_\gamma}]\to\overline{v},p)=\begin{cases}\pi_i & \text{if } \exists i.\ p=p_i\\ \emptyset & \text{otherwise}\end{cases}$$

$$T(\overline{u}[\overline{\pi}]\to(v_1)_{t_1}\dots(v_h)_{t_h},p)=\begin{cases}v_i & \text{if } \exists i.\ t_i=p\vee t_i=out_p\vee t_i=in_{j,p}\\ \emptyset & \text{otherwise}\end{cases}$$

Moreover, these functions are extended to multisets of rules, as follows: $\eta(\mathcal{R},p)=\biguplus_{r\in\mathcal{R}}\eta(r,p)$, $\Pi(\mathcal{R},p)=\biguplus_{r\in\mathcal{R}}\Pi(r,p)$, and $T(\mathcal{R},p)=\biguplus_{r\in\mathcal{R}}T(r,p)$.

Given a selection S, we define the following functions to determine, for any position $q\in Extents(1)$, which multisets of objects are: (i) required as reactants in q; (ii) required as promoters in q; and (iii) produced in q by the application of the rules. These functions are respectively denoted by $\eta(S,q)$, $\Pi(S,q)$, and $T(S,q)$, and are defined as follows:

$$\eta(S,q)=\biguplus_{(p,\mathcal{R})\in S}\eta(\mathcal{R},q-p)$$

$$\Pi(S,q)=\biguplus_{(p,\mathcal{R})\in S}\Pi(\mathcal{R},q-p)$$

$$T(S,q)=\biguplus_{(p,\mathcal{R})\in S}T(\mathcal{R},q-p)$$

Definition 4.2.4 (Valid selection)**.** Let W be a configuration, consider the following properties:

a. *applicability*: the reactants and promoters required by each rule are present in the system:

$$\forall p\in Extents(1).\ \eta(S,p)\uplus w_p\wedge\Pi(S,p)\uplus w_p$$

b. *mutual exclusivity of objects*: two mutually exclusive objects cannot occupy the same position p' in the configuration resulting from the application of the selection; note that, that during the step a ME object can disappear from a position and another one can take its place;

$$\forall p\in Extents(1).\ \sum_{x\in E}\mathbf{m}(w_p,x)-\mathbf{m}(\eta(S,p),x)+\mathbf{m}(T(S,p),x)\leq 1$$

Let us denote by $\mathcal{S}_W^{(\epsilon)}$ the set of all selections, satisfying a set of properties $\epsilon\subseteq\{a,b\}$, for a given configuration W. A selection $S\in\mathcal{S}^{(a,b)}$ is *valid* for W iff the following property hold:

c. *priority and maximality*: let us define the following priority relation on multisets of rules, and its extension to selections:

$$\begin{aligned}\mathcal{R} \preceq \mathcal{R}' \iff \forall r \in \mathcal{R}'.\\ \mathbf{m}(\mathcal{R}', r) < \mathbf{m}(\mathcal{R}, r) \implies \exists r' > r.\ \mathbf{m}(\mathcal{R}', r') > \mathbf{m}(\mathcal{R}, r');\end{aligned} \tag{4.1}$$

$$S \preceq S' \iff \forall (p, \mathcal{R}) \in S.\ \exists (p, \mathcal{R}') \in S'.\ \mathcal{R} \preceq \mathcal{R}' \tag{4.2}$$

A selection $S \in \mathcal{S}^{(a,b)}$ is *valid* iff $\forall S' \succ S.\ S \notin \mathcal{S}^{(a,b)}$.

The set of all valid selections is denoted by $\mathcal{S}^{\text{valid}}_W$. In the notations $\mathcal{S}^{(\epsilon)}_W$ and $\mathcal{S}^{\text{valid}}_W$, the indication of the configuration W can be omitted if there is no risk of ambiguity.

Consider the priority relation on multisets of rules, defined by Equation 4.1. Given the multisets $\mathcal{R}, \mathcal{R}'$, $\mathcal{R} \preceq \mathcal{R}'$ means that, for each rule $r \in \mathcal{R}$, the multiplicity of application of r can be decreased in $\mathcal{R}'$ with respect to $\mathcal{R}$ only if there is some other rule r', with priority greater than r, whose multiplicity is increased in $\mathcal{R}'$. The priority relation is extended to selections in Equation 4.2, by considering the multisets of rules selected for each position. Finally, a *valid* selection S is such that there does not exist a greater selection $S' \succ S$ which satisfies the properties (a) of applicability and (b) of mutual exclusivity.

Definition 4.2.5. Let $P = (V, E, \mu, \sigma, W_1, (R_1, \rho_1), \ldots, (R_n, \rho_n))$ be a Spatial P system model. Let $apply(W, S) = W'$ be such that $\forall p \in Extents(1).\ w'_p = w_p \setminus\setminus \eta(S, p) \uplus T(S, p)$. Then the semantics of P is defined as the least relation $W \xrightarrow{S} W'$, with W, W' sets indexed by positions in $Extents(1)$, $S \in \mathcal{S}^{\text{valid}}_W$, and where $W' = apply(W, S)$.

A *computation* of a Spatial P system, from the initial configuration W_1, is a sequence of transitions $W_1 \xrightarrow{S_1} W_2 \xrightarrow{S_2} \cdots \xrightarrow{S_{k-1}} W_k$, for some valid selections $S_i \in \mathcal{S}^{\text{valid}}_{W_i}$, $i = 1, \ldots, k-1$. A *successful computation* is a computation such that no transition are possible in the last configuration.

Note that, since we assume that the initial configuration W_1 does not contain conflicts among mutually exclusive rule, also any reachable W' does not contain conflicts, namely any W' such that $W_1 \xrightarrow{S_1} W_2 \xrightarrow{S_2} \cdots \xrightarrow{S_{k-1}} W_k = W'$ for some valid selections $S_i \in \mathcal{S}^{\text{valid}}_{W_i}$, $i = 1, \ldots, k-1$, is such that $\forall p \in Extents(1).\ \sum_{x \in E} \mathbf{m}(w'_p, x) \leq 1$.

The result of a successful computation is represented by the multiset of objects sent out of the skin membrane during the evolution, which are described by a Parikh vector. The set of all vectors computed by a Spatial P system Π is denoted $Ps(\Pi)$. Let us denote by $SP_n(\alpha)$ the class of Spatial P Systems with at most $n \geq 1$ membranes and using rules described by the set of labels α. Label *coo* indicates that cooperating rule are allowed, while *ncoo* that they are not. Similarly, we indicate with *me* that ME objects can be used, and with *nme* that their use is not allowed. As regards promoters, we use label *pro* to indicate that promoters can be used, while for priorities we use label *pri* if they can be used (that is, if the partial order relation is *not* the identity relation). For the sake of readability, we assume that the absence of label *pro* or *pri* denotes that the corresponding feature is not allowed. Analogously to what done for P systems, when the number of membranes is not bounded we replace n with $*$. Let $PsSP_n(\alpha)$ be the family of sets of vectors of natural numbers computed by the Spatial P systems of class $SP_n(\alpha)$.

4.3 Universality of Spatial P systems

In this section we study the universality of Spatial P systems without promoters and priorities among evolution rules. We first prove that Spatial P systems are not universal when only non-cooperating rules are used and ME objects are not allowed. Then we prove that universality can be reached if ME objects are allowed.

Theorem 4.3.1. $PsSP_*(ncoo, nme) \subseteq PsP_1(ncoo)$.

Proof. We show how to translate a Spatial P system with only non-cooperating rules and no ME objects $(V, \emptyset, \mu, \sigma, W_0, R_1, \ldots, R_n)$ into an equivalent classical P system $(\hat{V}, \mu, \hat{w_1}, \hat{R_1})$ with one membrane and only non-cooperating rules.

The idea is to use the spatial information to translate each evolution rule into a set of rules which take into account the position of objects and the spatial membrane structure. We discard the spatial description of membranes σ, but maintain the membrane structure μ. As regards the objects, the position of each object becomes part of the object name as a superscript. In particular, the set of objects of the translated P system is the following:

$$\hat{V} = V \cup \{\, \overline{x^{\mathrm{p}}} \mid x \in V \wedge p \in Extents(1)\}.$$

An object $x \in V$ contained in a position p is mapped into the object $\overline{x^{\mathrm{p}}} \in \hat{V}$. Therefore the initial configuration $\hat{w_1}$ of the translated P system is defined as follows:

$$\hat{w_1} = \biguplus_{p \in Extents(1)} \left\{ (\overline{x^{\mathrm{p}}}, n) \,\middle|\, (x, n) \in w_p^{(0)} \right\}$$

For any membrane i, each rule $r \in R_i$ of the Spatial P system is translated into at most $|Region(i)|$ rules, by instantiating the rule r for each position $q \in Region(i)$. Formally, the set of rules $\hat{R_1}$ of the translated P system is defined as the smallest set satisfying the following inference rule:

$$\frac{\begin{array}{c} i \in \{1, \ldots, n\} \qquad q \in Region(i) \\ (x)_p \to (y_1)_{t_1}, \ldots, (y_h)_{t_h} \in Enabled(R_i, q) \qquad s = p + q \qquad \forall j.\ z_j = enc(q, i, (y_j)_{t_j}) \end{array}}{\overline{x^{\mathrm{s}}} \to z_1 \ldots z_h}$$

where

$$enc(p, i, (y)_t) = \begin{cases} \overline{y^{\mathrm{q}}} & \text{if } t \in \mathbb{Z}^2,\ q = p + t; \\ \overline{y^{\mathrm{q}}} & \text{if } (t = out_d \wedge i \neq 1) \vee t = in_{k,d},\ q = p + d; \\ y & \text{if } t = out_d \wedge i = 1. \end{cases}$$

By definition, a rule r is translated for a position $q \in Extents(1)$ if and only if the rule is q-enabled. Finally, the rules sending objects out of the skin membrane drop their superscript, making the output exactly equal to that of the original Spatial P system. □

Since P systems with only non-cooperating rules are not universal [70], Theorem 4.3.1 implies that also Spatial P systems with only non-cooperating rules and, in particular, no ME objects are not universal. In the following we show that we can reach universality by allowing ME objects. In the proof of this result we show that any matrix grammar

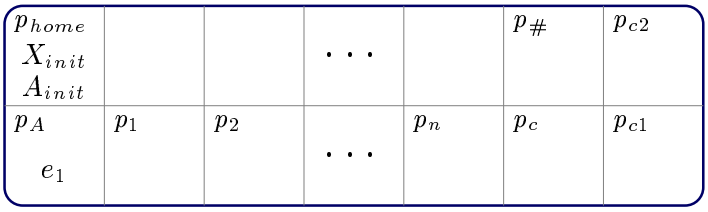


Figure 4.3: Initial step of the simulation.

with appearance checking (see Section 4.1.5) can be simulated by a Spatial P system with non-cooperating rules. Precisely, the following theorem shows that Spatial P systems with ME objects and using only non–cooperating rules are universal. In particular, note that neither promoters nor priorities are allowed.

Theorem 4.3.2. $PsSP_1(ncoo, me) = PsRE$.

Proof. It is enough to show that for a grammar G in binary normal form there is a Spatial P system $\Pi_G = (V, E, \mu, \sigma, W_0, R_1)$ with one membrane, which uses ME objects and only non-cooperating evolution rules, such that $Ps(\Pi_G) = \Psi_T(L(G))$. We build Π_G as a system with only a root membrane, whose geometry is depicted in Figure 4.3.

The set of ordinary objects of Π_G contains the symbols of G plus some control objects: $V = N \cup T \cup \{c_1, c_1', c_2, c_2', c_\#, c_\#', \#\}$. All objects corresponding to grammar symbols will reside in position p_{home}, as shown in Figure 4.3. The other positions can be seen as control positions to simulate the application of the matrices of the grammar. The ME objects $E = \{e_0, e_1, e_2, e_3\}$ do not move, and they expire after the given time, i.e. e_i is cancelled after i time steps. Formally, the rules for e_i objects and trap symbol # are the following:

$$e_3 \to e_2 \qquad e_2 \to e_1 \qquad e_1 \to e_0 \qquad e_0 \to \lambda$$

As regards the trap symbol #, once it is produced it is never removed, and there is a rule $\# \to \#$ which ensures that the P system never halts. The other rules for the membrane are defined in the following.

The execution consists of a repetition of cycles of four time steps. Each cycle selects non-deterministically and then applies one of the matrices $m_1, \ldots, m_n$. If during the cycle something goes wrong, i.e. either the application of the selected matrix is not possible or a matrix $(X \to Y, A \to \#)$ of type 3 is selected when a symbol A is present, then the trap symbol # is introduced yielding a non-terminating system, which corresponds to aborting the computation.

Let us define how matrices are mapped into rules. A matrix $m_i : (X \to Y, A \to x)$, $1 \leq i \leq k$, i.e. without appearance checking, is mapped into the following two rules:

$$(1) \qquad (X)_{(0,0)} \to (Y)_{(0,0)} \, (e_3)_{p_1} \cdots (e_3)_{p_{i-1}} \, (e_3)_{p_{i+1}} \cdots (e_3)_{p_n} \, (c_1)_{p_c} \, (e_2)_{p_{c1}}$$

$$(2) \qquad (A)_{(0,0)} \to (x)_{(0,0)} \, (e_3)_{p_A} \, (e_2)_{p_i} \, (c_2)_{p_c} \, (e_1)_{p_{c2}}$$

Note that, for readability, we denote by p_j the vector from position p_{home} to the position p_j itself, and similarly for the other positions. Rule (1) is used to select the matrix to be applied: since it puts a ME object in all positions p_j, with $j \neq i$, only one can be applied among the n which are available. Note that the occupied positions will be free exactly

after 3 time steps (the object e_3 is used) respecting the duration of the cycle. A control object c_1 is placed in position p_c and position p_{c1} is occupied for two time steps (object e_2). Moreover, there is a rule $c_1 \to c_1'$ to let object c_1 pass from step 1 to step 2. Rule (2) cannot be applied at step 1 of the cycle because position p_A is not initially free (it contains e_1) and the rule is trying to put an ME object there. Thus, the rule can be applied only at the second step of the cycle. It transforms the non-terminal A and puts an ME object e_2 in position p_i, the only one still free after the first step. Note that other rules without appearance checking cannot be applied at the second step because the positions in which they try to put the ME object is occupied. Note also that object e_3 is put in p_A, thus freeing the position only at the second step of the next cycle. A control object c_2 is put in position p_c and position p_{c2} is occupied for one time step.

A matrix $m_i : (X \to Y, A \to \#)$, $k+1 \leq i \leq n$, i.e. with appearance checking, is also mapped into two rules:

$$(1') \qquad (X)_{(0,0)} \to (Y)_{(0,0)} \, (e_3)_{p_1} \cdots (e_3)_{p_{i-1}} \, (e_3)_{p_{i+1}} \cdots (e_3)_{p_n} \, (c_\#)_{p_\#}$$

$$(2') \qquad (A)_{(0,0)} \to (\#)_{(0,0)} \, (e_3)_{p_A} \, (e_2)_{p_i}$$

The differences in rule $(1')$ w.r.t. rule (1) are that control object c_1 is not needed and that an object $c_\#$ is put in the control position $p_\#$. For this object we add rules $c_\# \to c_\#'$ and $c_\#' \to (e_2)_{p_A}$, i.e., we ensure the correct continuation of the cycle if the trap symbol is not generated at the second step of the cycle. By rule $(2')$, if a symbol A is present in p_{home} at the second step of the cycle, after matrix i was selected, the trap symbol is generated and the computation aborted, as it would happen for the grammar. Otherwise, if there are no A symbols the cycle goes towards its end doing nothing and the computation will continue in the next cycle.

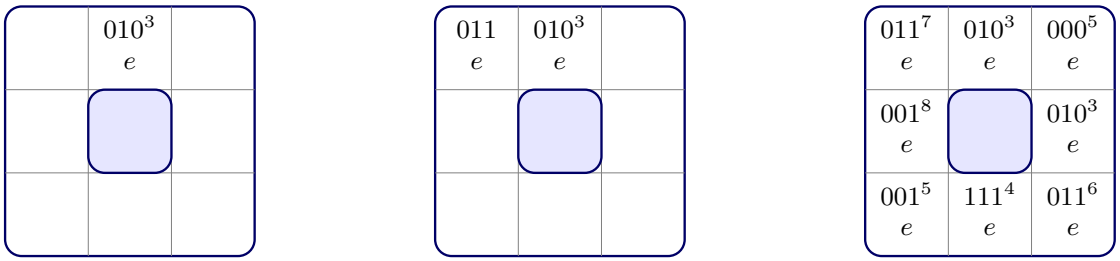
In the third step of the cycle we perform a double check, using both control objects c_1' and c_2, that all positions p_i, $1 \leq i \leq n$, are occupied. Otherwise, either one matrix was selected, but the second part was not applied due to a missing object A, or no symbol of N_1 was left (i.e. the state symbol was cancelled), but some non-terminal symbol in N_2 remains. In both cases the computation has to be aborted. This is performed by the following rules: $c_1' \to \# \, (e_0)_{p_1}$, ..., $c_1' \to \# \, (e_0)_{p_n}$ and $c_2 \to \# \, (e_0)_{p_1}$, ..., $c_2 \to \# \, (e_0)_{p_n}$.

In the fourth step of the cycle we simply cancel control objects in order to make the next cycle start in a consistent state, by means of rules $c_1' \to (e_0)_{p_{c1}}$ and $c_2 \to (e_0)_{p_{c2}}$. Note that, since positions p_{c1} and p_{c2} are occupied in step 3, these rules can not be applied, i.e. the control objects are not cancelled in step 3.

Finally, for each terminal in T there is a rule sending it out of the membrane. Since all terminals are produced in position p_{home}, which is adjacent to the edge of the membrane, such a rule can be always applied. □

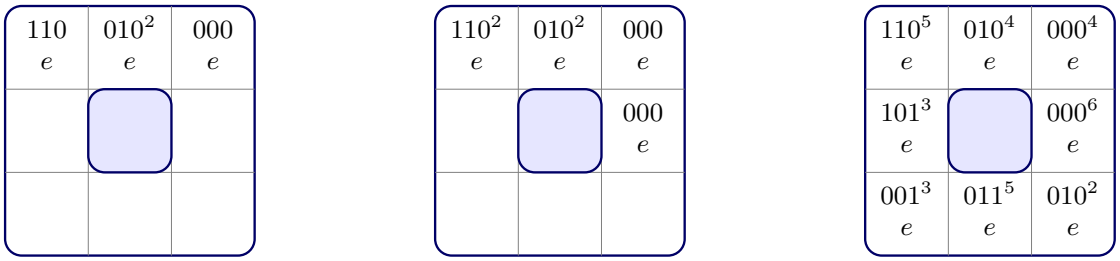
4.4 An example of application

In this section we show a simple application of our model to a classical example: the evolution of "ring species" based on small changes between geographically contiguous populations [45]. A ring species is a species which expanded along two pathways around a geographic barrier, with the forms which gradually diverge along the pathways. The intermediate contiguous forms can interbreed but, when the terminal forms meet on the other side of the barrier, they have accumulated so many changes that they behave like



(a) Initial population. (b) First step. (c) Final population.

Figure 4.4: A possible evolution of the system.



(a) First step. (b) Second step. (c) Final population.

Figure 4.5: Another possible evolution of the system, starting from the initial state depicted in Figure 4.4a.

different species. This process results in a complete ring of populations with a single species boundary. An example of a ring species is the Greenish warbler, *Phylloscopus trochiloides*. The Greenish warbler is a small insectivorous bird that breeds in forests over a range spanning much of the Palaearctic. The species consists of six subspecies, five of which form a ring around the Tibetan Plateau. Two of the subspecies coexisted without interbreeding in the Yenisey River valley of central Siberia, with gradual variation through the chain of populations to the south.

In the following we show a simple model of a species which expands around a barrier. The colonization of a new space can be associated with a small change in the genotype of the moving population. Such small changes do not prevent the possibility for two contiguous populations to interbreed.

Each population is represented by its genotype: a string of three loci each of them having, as possible values (alleles), either 0 or 1. Two populations can interbreed if their genotypes differ in one position at most.

Figure 4.4a shows the initial situation. The environment is represented by a membrane of size 3×3, and the barrier is represented by an inner membrane of size 1×1. The initial population is located in position $(1, 2)$, and it is composed of three individuals with 010 genotype (010^3). The ME object e states that a position is already colonized and it cannot be reached by a different population.

In order to present a compact model, we use the extended notation $u_1 - u_2 \to v_1 - v_2$ for specifying a group of rules. Let us introduce the following abbreviations for denoting adjacent positions: $N = (0, 1)$, $S = (0, -1)$, $E = (1, 0)$, $W = (-1, 0)$. A rule $u_1 - u_2 \to$

$v_1 - v_2$ can be applied to adjacent positions, in any direction, and formally corresponds to a set of rules defined as follows:

$$u_1 - u_2 \to v_1 - v_2 \quad \equiv \quad \begin{cases} (u_1)_{(0,0)}(u_2)_N \to (v_1)_{(0,0)}(v_2)_N \\ (u_1)_{(0,0)}(u_2)_S \to (v_1)_{(0,0)}(v_2)_S \\ (u_1)_{(0,0)}(u_2)_E \to (v_1)_{(0,0)}(v_2)_E \\ (u_1)_{(0,0)}(u_2)_W \to (v_1)_{(0,0)}(v_2)_W \end{cases}$$

The evolution rules for the model are presented in the following, where $x, y, z \in \{0, 1\}$, an overlined symbol $\overline{x}$ represents the negation of x (as if $0, 1$ represent the logical values $false$ and $true$), and d denotes a direction $d \in \{N, S, E, W\}$. For the sake of readability, we omit the indication of position $(0, 0)$.

$$xyz \to xyz \tag{1}$$
$$xyz \to \lambda \tag{2}$$
$$xyz^2 \to xyz^3 \tag{3}$$
$$xyz - xyz \to xyz^2 - xyz \tag{4}$$
$$xyz - \overline{x}yz \to xyz^2 - \overline{x}yz \tag{5}$$
$$xyz - x\overline{y}z \to xyz^2 - x\overline{y}z \tag{6}$$
$$xyz - xy\overline{z} \to xyz^2 - xy\overline{z} \tag{7}$$
$$xyz \to xyz_d\ e_d \tag{8}$$
$$xyz \to \overline{x}yz_d\ e_d \tag{9}$$
$$xyz \to x\overline{y}z_d\ e_d \tag{10}$$
$$xyz \to xy\overline{z}_d\ e_d \tag{11}$$

Rules of type 1 simply state that individuals can survive, while rules of type 2 say that individuals can die. Type 3 rules describe the reproduction of two individuals in the same position, while rules of type 4–7 describe the reproduction of individuals of two contiguous populations. The contiguous populations, in order to have offspring can have either the same genotype or two genotypes differing in only one locus. Because we assume that all individuals in a population have the same genotype, an offspring is placed in the population of the parent from which it inherits its genotype.

Rules of type 8 describe the expansion of a population (actually of an individual) in a contiguous position which is not already colonized (the expansion is possible only if the ME object e can be placed in the target position). Rules of type 9–11 describe expansions associated with small changes in the genotype.

The system can evolve in many directions. Figure 4.4 shows a possible evolution of the system, starting from the initial state depicted in Figure 4.4a. Suppose that, in the first step, rule 11 is applied to an object in position $(1, 2)$, producing object 011 in $(0, 2)$, while rule 1 is applied to the remaining two objects 010. The resulting state is depicted in Figure 4.4b. The system can evolve to a state in which all the positions are colonized, such at the one shown in Figure 4.4c. In this case, it is easy to see that the expansion followed two pathways around the barrier. Each population can interbreed with the contiguous ones apart from the populations in positions $(0, 0)$ and $(1, 0)$. The two populations have accumulated so many changes that their genotypes are incompatible.

Figure 4.5 shows another possible evolution of the system, for the same starting state as before (Figure 4.4a). Figure 4.5a shows a different possible state reached after the first step, corresponding to the application of rule 9 producing 110 in $(0, 2)$, rule 10 producing 000 in $(2, 2)$, and rule 1 deleting one object 010 from position $(1, 2)$. After another step, the system reaches the state shown in Figure 4.5b. In this case, by rule 8, species 000 colonizes position $(2, 1)$; by rule 5, another object 110 is created in $(0, 2)$; while the other 010 in $(1, 2)$ is kept unmodified by the application of rule 1. Figure 4.5c shows a reachable state in which, as the previous example, all position are colonized, and populations 110 in $(0, 2)$, and 101 in $(0, 1)$, cannot interbreed since their genotypes are too different.

Chapter 5

Spatial P systems simulation

The simulation of Spatial P systems models is a fundamental problem which we need to address in order to increase the usefulness of the calculus. Actually, simulating generic Spatial P system models can be quite complex. For this reason, we initially present an algorithm which works only for a restricted kind of models, namely models with only mutually-exclusive objects and where evolution rules have exactly one object as reactant and one as the product. We will later discuss possible extensions of the simulation procedure for more general models. Finally, we show how Spatial P systems can be used to simulate a model describing the behaviour of herring schools.

5.1 Simulation algorithm for restricted models

We present a simulation algorithm which efficiently simulates Spatial P system models constrained as follows:

- there are n mutually-exclusive objects: $E = \{a_1, \ldots, a_n\}$, and no ordinary objects ($V = \emptyset$);

- all rules have exactly one object as reactant and one as product, that is each rule is of the form $(a_x)_{(0,0)}[\overline{\pi}] \to (a_y)_p$ for some promoters $\overline{\pi}$, position p, and $a_x, a_y \in E$. Moreover, let us denote by $R^{(a_x)}$ the subset of rules in which a_x appears as the reactant of the rule. We denote by K the maximum cardinality of the set $R^{(a_x)}$, for all $a_x \in E$. We assume that the priority relation on rules in $R^{(a_x)}$ defines a total ordering;

Moreover, for simplicity in the presentation of the algorithms, we assume that the skin membrane is square, with size $N \times N$ for some $N \in \mathbb{N}$.

Let *target* be a function that gives the resulting (relative) position associated with a rule, that is: $target(a_{(0,0)}[\overline{\pi}] \to a'_{p'}) = p'$. We introduce the definition of a *filling* selection, which will be used in the definition of the simulation algorithm.

Definition 5.1.1. Given a configuration W, a selection $S \in \mathcal{S}_W^{(a,b)}$ is said to be *filling* for

W iff the following property holds:

$$\begin{aligned}&\forall (p, \mathcal{R}) \in S.\ \forall r' \in Enabled(R^{(a)}, p).\\ &\quad (\mathcal{R} = \{\} \implies w_p \setminus\setminus \eta(S, p + target(r')) \uplus T(S, p + target(r')) \neq \emptyset) \wedge\\ &\quad (\mathcal{R} = \{r\} \implies r' \leq r \vee w_p \setminus\setminus \eta(S, p + target(r')) \uplus T(S, p + target(r')) \neq \emptyset)\end{aligned}$$

We denote the set of all filling selections, for a configuration W, as $\mathcal{S}_W^{\text{filling}}$.

A filling selection is such that the resulting position of each rules r' having higher priority than the currently selected rule r, if any, is already occupied. If no rule is selected for the position p, then the resulting position of each p-enabled rule is occupied. This implies that, given a rule r applied to an object a in a position p, it is not possible to *replace* the application of r with any other p-enabled rule $r' \in R^{(a)}$ such that $r' > r$ and $target(r') \neq target(r)$, since the resulting position is already occupied. Note that the definition of filling selection considers, for possible replacements, each position singularly.

The simulation algorithm for the restricted case performs a sequence of steps, where each step corresponds to a transition of the Spatial P system and is simulated by the algorithm *SimulateStep*. Given a configuration W, *SimulateStep*(W) generates another configuration W' simulating a possible transition of the Spatial P system.

A step is composed of the following three phases:

1. The first phase determines a *filling* selection $S \in \mathcal{S}^{(a,b)}$, that is satisfying the properties of applicability (a) and mutual exclusivity (b), from Definition 4.2.4, but possibly violating the property on priority and maximality (c).

2. If the selection obtained from the first phase is not valid (that is the property on priority and maximality is violated) then the second phase finds a selection $S^\top \succ S$ which is valid ($S^\top \in \mathcal{S}^{\text{valid}}$), by computing a sequence of selections $S = S_1 \prec S_2 \prec \cdots \prec S_n = S^\top$, in which each S_i satisfies properties (a) and (b) from Definition 4.2.4.

   By the definition of filling selection, by considering any *singular* position p, it is not possible to replace the currently selected multiset $\mathcal{R} = S(p)$ (which is such that either $\mathcal{R} = \emptyset$ or $\mathcal{R} = \{r\}$, for some r) with a greater multisets $\mathcal{R}' \succ \mathcal{R}$ such that $\mathcal{R}' = \{r'\}$ for some $r' > r$, unless the resulting position of r' is the same as that of the currently occupied position. Therefore, in order to find a selection S_i greater than the preceding one S_{i-1}, it is necessary to perform various replacements all at once, in different positions. This corresponds to finding a set of positions C where it is possible to replace the multiset of rules $\mathcal{R}^{(p)}$, applied in each position $p \in C$, with another multiset of rules of higher priority $\mathcal{R}'^{(p)} \succ \mathcal{R}^{(p)}$. By performing such a replacement, we obtain a selection $S_{i+1} \succ S_i$, which is also filling. This procedure is repeated until there are no more replacements to be performed, reaching selection $S^\top$.

3. Finally, the valid selection $S^\top$ found is applied to the current configuration W giving W'.

5.1.1 Implementation

Algorithm 1 contains the implementation of *SimulateStep* algorithm. *SimulateStep* takes as input a matrix of natural numbers, of size $N \times N$, describing the content of each cell,

Algorithm 1 SimulateStep($grid : \mathbb{N}[,]$)

1: $collisions\_grid = GenerateSelection(grid)$
2: $ValidateSelection(collisions\_grid)$
3: **return** CloseStep($collisions\_grid$)

and returns the updated grid resulting from the application of a valid selection of rules to the model. The positions inside such a matrix correspond to the positions of the extents of the skin membrane, that is to $Extents(1)$ (see Section 4.2). Formally, each cell (i, j) contains a value $x = grid[i, j]$ with $0 \leq x \leq n$, where value $x = 0$ denotes an empty cell, while a value $1 \leq x \leq n$ means that the object a_x is contained in the cell.

The main data structure of the algorithm is the *collisions_grid*, which is a matrix of size $N \times N$ representing a selection of rules. The *collisions_grid* is constructed by *GenerateSelection*, which implements Phase 1 of the algorithm. Then *collisions_grid* is used by $ValidateSelection$, which implements Phase 2. Finally, *CloseStep* uses the information contained in *collisions_grid* to compute the updated configuration, which is subsequently returned as the result of the algorithm. We propose in the following two different algorithms for implementing Phase 1 of the algorithm.

We assume the algorithm FindMoves(p), which computes the list of possible moves for the object contained in position p, according to the evolution rules defined for the Spatial P system model. Let a_y be the object in position p, then FindMoves(p) gives a list of pairs $\langle(\overline{q_1}, x_1), \ldots, (\overline{q_k}, x_k), (p, a_y)\rangle$, where each pair $(\overline{q_i}, x_i)$ corresponds to the application of a rule to the object in p. Precisely, each $(\overline{q_i}, x_i)$ describes the result of the application of a rule $(a_y)_{(0,0)}[\overline{\pi}] \to (a_{x_i})_{q_i} \in Enabled(R^{(a_y)}, p)$, which puts the object a_{x_i} in position $\overline{q_i} = p + q_i$. We assume the pairs to be in descending order of priority with respect to the rules they correspond. The last element returned by the function $FindMoves$ is (p, a_y), which describes the lack of application of any rule to the object. Therefore, $FindMoves$ never returns an empty list.

Each cell of *collisions_grid* contains a list of pairs of the form $(moves, n)$, where *moves* is a list describing all the possible moves of an object as obtained from $FindMoves$, and n is the index of a move in *moves*, i.e. $1 \leq n \leq length(moves)$. A pair $(moves, n)$ describes the current move which is considered by the algorithm for the object contained in some position p, where $moves = FindMoves(p)$. During the processing, the algorithm ensures that for all positions p, all the current moves associated with $collisions\_grid[p]$ refer to the position p. That is, $\forall p.\ \forall(moves, n) \in collisions\_grid[p].\ moves[n] = (p, x)$ for some x. Therefore, the contents of *collisions_grid* describes a selection of rules, to which it is directly related.

We propose two different algorithms for implementing Phase 1 of the algorithm. Let us consider the partially ordered set $\mathcal{S}^{(a,b)}$, with respect to the priority ordering $\preceq$ defined by Equation 4.2, of selections satisfying the property (a) of applicability and (b) of mutual exclusivity from Definition 4.2.4. Algorithm *GenerateSelection*1 implements an iterative procedure which starts from the *least* selection $\perp$ of the set $\mathcal{S}^{(a,b)}$, then iteratively constructs selections greater than the previous ones, until a filling selection (Definition 5.1.1) is obtained. Differently, algorithm *GenerateSelection*2 deals with the partially ordered set $\mathcal{S}^{(a)}$, w.r.t. relation $\preceq$, of selections satisfying the property (a) of applicability. It starts from the *greatest* selection $\top$ of the set $\mathcal{S}^{(a)}$, then iteratively constructs lesser selections,

until it reaches a filling selection which also satisfies the property (b) of mutual exclusivity from Definition 4.2.4.

The *SetupStep* algorithm, presented in the following, is used by both the algorithms *GenerateSelection*1 and *GenerateSelection*2 to generate their initial selection, either $\perp \in \mathcal{S}^{(a,b)}$, for the former, or $\top \in \mathcal{S}^{(a)}$, for the latter.

SetupStep *SetupStep* uses *FindMoves* to initialize *collisions_grid*, by considering for each object either its highest or lowest priority move. The flag $which \in \{b, t\}$ of the algorithm specifies which selection should be generated. Values b, t indicate elements $\perp \in \mathcal{S}^{(a,b)}$ and $\top \in \mathcal{S}^{(a)}$, respectively. If $\perp$ is requested, then the current index of each list of moves *moves* is set to $length(moves)$ (line 11), thus referring to the lowest priority move. Otherwise, if $\top$ is requested, the current index of each list of moves *moves* is set to 1, corresponding to the highest priority move.

GenerateSelection1 Algorithm *GenerateSelection*1 starts from selection $\perp \in \mathcal{S}^{(a,b)}$, in which no rule is applied to any object, i.e. such that $\forall (p, R) \in S.\ R = \{\}$. The set Q represents, at any iteration, a superset of the positions occupied by some object. At each iteration, a position $p \in Q$ is randomly selected and removed from Q, then the algorithm looks if there is a higher priority rule which can replace the currently selected move without causing a conflict, that is without violating the constraint on mutually exclusive objects. Assuming that n denotes the currently selected move for the object (line 8), the algorithm searches for another rule to apply, corresponding to an index $index < n$. If such a rule is found ($index \neq -1$) then it is applied, by freeing position p (line 18) and putting the pair $(moves, index)$ into position q (line 19). In case there is more than one higher priority rule applicable, the algorithm selects the one with the lowest priority.

The algorithm uses a set of positions *visited*, which represents the set of positions which, since the last replacement, have been visited and in which no rule replacement could be done. During a step, if a replacement is performed, then all the previously visited positions are added to Q (line 20), since replacing a rule application can cause position p to be freed, thus enabling the application of some other rules which were previously blocked. Note that also the newly occupied position q is added to Q, in order to allow further replacements in it. Finally, *visited* is reset to the empty set. On the other hand, if no replacement is performed in a step, then position p is added to the set *visited* (line 23), as no replacement is possible in p until some other replacement is performed somewhere else.

GenerateSelection2 Algorithm *GenerateSelection*2 starts from element $\top \in \mathcal{S}^{(a)}$, and implements an iterative procedure in which, at each iteration, it resolves a conflict in some position among some mutually exclusive objects. The set Q represents a superset of all the non-empty positions violating the property (b) of mutual exclusivity from Definition 4.2.4. At each iteration, a position $p \in Q$ is randomly selected, and then the conflict in p is resolved by randomly selecting one application to be confirmed among those in $collisions\_grid[p] = \langle (moves_1, n_1), \ldots, (moves_k, n_k) \rangle$ and by cancelling the other applications. That is, exactly one pair $(moves_i, n_i)$ is kept in $collisions\_grid[p]$, while each other pair $(moves_j, n_j)$, $j \neq i$, is removed from $collisions\_grid[p]$ and added, as a pair $(moves_j, n_j{+}1)$ to the position corresponding with the lower-priority move $moves_j[n_j{+}1]$.

Intuitively, this behaviour corresponds to randomly confirming a move, and then "pushing back" all other objects towards their starting positions. At the same time, all the positions to which some object are moved to are added to set Q (line 23), as pushing back an object can cause new conflicts in those positions.

Given $collisions\_grid[p] = \langle (moves_1, n_1), \ldots, (moves_k, n_k) \rangle$, the application to confirm is randomly selected only if there is no object to which no rule can be applied anymore (either because they have been cancelled in previous iterations or, possibly, because there were no rule applicable from the beginning). In this case, that particular object which cannot be pushed back has to be confirmed for the position. This behaviour is implemented at lines 8–13.

The procedure implemented by *GenerateSelection* terminates as soon as a conflict-free selection has been reached, which satisfies the properties of applicability (a) and mutual exclusivity (b). Since it is not possible to push back the objects indefinitely, this procedure eventually terminates. In the worst case, all objects are brought back to their starting positions, meaning that no rule is applied to any object.

Note that, once a position has been occupied, it is not freed anymore. That is, let $Occ_i = \{p \mid length(collisions\_grid^{(i)}[p]) > 1\}$ be the set of positions in which, at the beginning of the i^{th} iteration, there is at least one application considered for each position. Then, for all iterations i, $Occ_i \subseteq Occ_{i+1}$. However, the object present in a position may change at different iterations, as new conflict may arise for positions which have been already confirmed in previous iterations. Finally, this implies that the additional property on the selection, required by Phase 1, is satisfied. That is, for any rule applied in a position p, the positions resulting from the application of any other higher-priority p-enabled rule are occupied.

ValidateSelection Algorithm $ValidateSelection$ implements Phase 2 of the algorithm which, given the restricted form of rules in this case, means computing a sequence of selections $S = S_1 \prec S_2 \prec \cdots \prec S_n = S^{\perp}$ where each selection S_i is determined by finding a set of occupied positions in the preceding selection S_{i-1} where it is possible to replace the rule applied in each position with another rule of higher priority. Therefore, at each iteration, the $ValidateSelection$ algorithm searches a set of applications of rules in which each application can be replaced by some higher-priority rule, while still satisfying the properties of applicability (a) and mutual exclusivity (b) from Definition 4.2.4.

The implementation uses a directed graph $G_{\text{dep}} = (V, E)$ of dependencies between applications of rules, which is such that each directed cycle in the graph describes a possible replacement of rules as required by Phase 2 of the algorithm. The nodes $V = \{p \mid length(collisions\_grid[p]) > 0\}$ of the graph correspond to non-empty positions, and the set of edges E is such that, for each position $p \in V$, there is an edge from p to each position q resulting from the application of any other higher-priority p-enabled rule. Formally, $\forall p, q \in V$ such that $collisions\_grid[p] = \langle (moves, n) \rangle$:

$$(p, q) \in E. \iff \exists i \in \{1, \ldots, n-1\}.\ (q, x) = moves(i) \text{ for some } x$$

$ValidateSelection$ algorithm initially constructs the graph G_{dep}, then iteratively searches for a cycle $C = \langle p_1, \ldots, p_k \rangle$, performs the replacement of rules described by C by updating $collisions\_grid$ (line 6), and updates the graph. The graph is updated by removing, for each selected rule of the cycle C, all the edges corresponding to the rule itself and the

Algorithm 2 SetupStep($grid : \mathbb{N}[,],\ which : \{t, b\}$)

```
1:  collisions_grid = new array[N, N]
2:  for i = 1 to N do
3:    for j = 1 to N do
4:      collisions_grid[i, j] = ⟨⟩
5:    end for
6:  end for
7:  for i = 1 to N do
8:    for j = 1 to N do
9:      if grid[i, j] ≠ 0 then
10:       moves = FindMoves(grid, i, j)
11:       if which = t then n = 1 else n = length(moves) end if
12:       ((i', j'), x) = moves[n]
13:       collisions_grid[i', j'] = collisions_grid[i', j'] @ ⟨(moves, n)⟩
14:     end if
15:   end for
16: end for
17: return collisions_grid
```

other lower priority rules, and by replacing each edge (p_i, q), corresponding to some higher priority rule, by (p_{i+1}, q) (line 7). The replacing of edges (p_i, q) with (p_{i+1}, q) means that higher priority rules can still be used for the object in p_{i+1}, which would cause it to move to some other position q. This last modification of the graph is actually the same as reconstructing the graph from the updated *collisions_grid*. In order to ensure that all possible selections of rules can be obtained from the algorithm, it is necessary that the search for a cycle randomly finds any cycle of the graph. This procedure is iterated until there are no more cycles, terminating with a *collisions_grid* which corresponds to a valid selection.

Actually, there can be different replacements which correspond to the same cycle C. This happens if there are different rule applications which would put an object in a same position, among the p-enabled rules of priority higher than the current rule. Since the algorithm does not take into account the actual object present in a position, all those rules can be considered equivalent for the purpose of the replacement. In fact, selecting any of those rules would produce a replacement which can be applied. Moreover, not selecting the rule with the highest priority would also be correct, as the highest priority rule could be found and applied in some subsequent iteration. However, the algorithm actually applies the highest priority rule among those equivalent (line 5), since the removal of edges corresponding to lower priority rules (line 7) is correct only in that case.

Finally, *CloseStep* applies the valid selection, obtained from previous steps, to the current state *grid*, returning the updated state *new_grid*.

Example 5.1.1. This example illustrates how the algorithm works, in which both implementations *GenerateSelection*1 and *GenerateSelection*2 of Phase 1 of the algorithm directly find a valid selection. Consider a Spatial P system with objects $V = \emptyset$, $E = \{a, b\}$, and the following rewrite rules with their relative priorities (where $r_1 > r_2$ indicates that

Algorithm 3 GenerateSelection1($grid : \mathbb{N}[,]$)

1: $collisions\_grid = \text{SetupStep}(grid, b)$
2: **let** $Q = \{(i,j) \mid length(collisions\_grid[i,j]) > 0\}$
3: $visited = \{\}$
4: **while** $Q \neq \{\}$ **do**
5: select randomly a position $p = (i,j) \in Q$
6: $Q = Q \setminus \{p\}$
7: $index = -1$
8: **let** $\langle (moves, n) \rangle = collisions\_grid[i,j]$
9: **for** $k = n - 1$ **to** 1 **do**
10: **let** $(q,x) = moves[k]$
11: **if** $collisions\_grid[q] = \langle\rangle$ **then**
12: $index = k$
13: **break**
14: **end if**
15: **end for**
16: **if** $index \neq -1$ **then**
17: **let** $(q,x) = moves[index]$
18: $collisions\_grid[p] = \langle\rangle$
19: $collisions\_grid[q] = \langle (moves, index) \rangle$
20: $Q = Q \cup visited \cup \{q\}$
21: $visited = \{\}$
22: **else**
23: $visited = visited \cup \{p\}$
24: **end if**
25: **end while**
26: **return** $collisions\_grid$

Algorithm 4 GenerateSelection2($grid : \mathbb{N}[,]$)

1: $collisions\_grid = \text{SetupStep}(grid, t)$
2: **let** $Q = \{(i,j) \mid length(collisions\_grid[i,j]) > 1\}$
3: **while** $Q \neq \{\}$ **do**
4: select randomly a position $p = (i,j) \in Q$
5: $Q = Q \setminus \{p\}$
6: $index = -1$
7: $collisions = collisions\_grid[i,j]$
8: **for** $i = 1$ **to** length($collisions$) **do**
9: **let** $(moves, n) = collisions[i]$
10: **if** length($moves$) $= n$ **then**
11: $index = i$
12: **end if**
13: **end for**
14: **if** $index = -1$ **then**
15: $index = \text{random}(\{1, \ldots, \text{length}(collisions)\})$
16: **end if**
17: $collisions\_grid[i,j] = \langle collisions[index] \rangle$
18: **for** $i = 1$ **to** length($collisions$) **do**
19: **if** $i \neq index$ **then**
20: **let** $(moves, n) = collisions[i]$
21: $((i', j'), x) = moves[n+1]$
22: $collisions\_grid[i', j'] = collisions\_grid[i', j'] \mathbin{@} \langle (moves, n+1) \rangle$
23: $Q = Q \cup \{(i', j')\}$
24: **end if**
25: **end for**
26: **end while**
27: **return** $collisions\_grid$

Algorithm 5 ValidateSelection($collisions\_grid : \mathbb{N}[,]$)

1: **let** $G_{\text{dep}} = (V, E)$ be the directed graph such that:

$$V = \{p \mid length(collisions\_grid[p]) > 0\}$$
$$E = \{(p,q) \mid collisions\_grid[p] = \langle (moves, n) \rangle \wedge \exists x.\ \exists i.\ 1 \leq i \leq n-1 \wedge (q,x) = moves[i]\}$$

2: **while** there exists a cycle $\langle p_1, p_2, \ldots, p_k \rangle$ in graph (V, E) **do**
3: **for** $i = 1$ **to** k **do**
4: **let** $\langle (moves, n) \rangle = collisions\_grid[p_i]$
5: **let** $m = \max\{j \mid moves[j] = p_{(i+1 \bmod k)}\}$
6: $collisions\_grid[p_i] = \langle (moves, m) \rangle$
7: $E = E \setminus \{(p_i, q) \mid \exists q\} \cup \{(moves[m], q) \mid \exists x.\ \exists j.\ 1 \leq j \leq m-1 \wedge (q,x) = moves[j]\}$
8: **end for**
9: **end while**

Algorithm 6 CloseStep($collisions\_grid : \mathbb{N}[,]$)

```
1:  new_grid = new ℕ[N, N]
2:  for i = 1 to N do
3:    for j = 1 to N do
4:      if collisions_grid[i, j] = ⟨⟩ then
5:        new_grid[i, j] = 0
6:      else
7:        let ⟨(moves, n)⟩ = collisions_grid[i, j]
8:        let ((i', j'), x) = moves[n]
9:        new_grid[i, j] = x
10:     end if
11:   end for
12: end for
13: return new_grid
```

rule r_1 has higher priority than r_2):

$$a_{(0,0)} \to \overline{a}_{(2,0)} \;>\; a_{(0,0)} \to \overline{a}_{(1,0)}$$
$$b_{(0,0)} \to \overline{b}_{(0,1)}$$

Consider the following initial state:

$$W_0 = \begin{array}{|c|c|c|}\hline a & & \\ \hline & b & b \\ \hline \end{array}$$

There are three possible transitions of the Spatial P system from configuration W_0, which result in one of the following possible configurations:

$$W_1 = \begin{array}{|c|c|c|}\hline & \overline{b} & \overline{a} \\ \hline & & b \\ \hline \end{array} \qquad W_2 = \begin{array}{|c|c|c|}\hline a & \overline{b} & \overline{b} \\ \hline & & \\ \hline \end{array} \qquad W_3 = \begin{array}{|c|c|c|}\hline & \overline{a} & \overline{b} \\ \hline & b & \\ \hline \end{array}$$

We exemplify the execution of $SimulateStep(W_0)$, using $GenerateSelection1$.

Setup $SetupStep(W_0, b)$ produces the following initial state:

$$collisions\_grid^{(0)} = \begin{array}{|c|c|c|}\hline \langle (l_1, 3)\rangle & \langle\rangle & \langle\rangle \\ \hline \langle\rangle & \langle (l_2, 2)\rangle & \langle (l_3, 2)\rangle \\ \hline \end{array}$$
$$Q^{(0)} = \{(1,2), (2,1), (3,1)\}$$
$$visited^{(0)} = \{\}$$

where

$$l_1 = FindMoves(W_0, 1, 2) = \langle (3,2) : \overline{a},\ (2,2) : \overline{a},\ (1,2) : a\rangle$$
$$l_2 = FindMoves(W_0, 2, 1) = \langle (2,2) : \overline{b},\ (2,1) : b\rangle$$
$$l_3 = FindMoves(W_0, 3, 1) = \langle (3,2) : \overline{b},\ (3,1) : b\rangle$$

Note that $collisions\_grid^{(0)}$ corresponds to the initial state W_0. The $GenerateSelection1$ algorithm performs the following steps.

Step 1 Suppose $p = (1, 2)$; then $index = 2$ and the following state is reached at the end of the step, where $U^{(1)}$ represents the configuration described by $collisions\_grid^{(1)}$:

$collisions\_grid^{(1)} =$

$\langle\rangle$	$\langle(l_1, 2)\rangle$	$\langle\rangle$
$\langle\rangle$	$\langle(l_2, 2)\rangle$	$\langle(l_3, 2)\rangle$

$U^{(1)} =$

	$\overline{a}$	
	b	b

$Q^{(1)} = \{(2, 1), (2, 2)(3, 1)\}$ $visited^{(1)} = \{\}$

Step 2 $p = (2, 2)$; $index = 1$:

$collisions\_grid^{(2)} =$

$\langle\rangle$	$\langle\rangle$	$\langle(l_1, 1)\rangle$
$\langle\rangle$	$\langle(l_2, 2)\rangle$	$\langle(l_3, 2)\rangle$

$U^{(2)} =$

		$\overline{a}$
	b	b

$Q^{(2)} = \{(2, 1), (3, 1), (3, 2)\}$ $visited^{(2)} = \{\}$

Step 3 $p = (3, 2)$; $index = -1$:

$collisions\_grid^{(3)} = collisions\_grid^{(2)}$ $U^{(3)} = U^{(2)}$

$Q^{(3)} = \{(2, 1), (3, 1)\}$ $visited^{(3)} = \{(3, 2)\}$

Step 4 $p = (3, 1)$; $index = -1$:

$collisions\_grid^{(4)} = collisions\_grid^{(3)}$ $U^{(4)} = U^{(3)}$

$Q^{(4)} = \{(2, 1)\}$ $visited^{(4)} = \{(3, 1), (3, 2)\}$

Step 5 $p = (2, 1)$; $index = 1$:

$collisions\_grid^{(5)} =$

$\langle\rangle$	$\langle(l_2, 1)\rangle$	$\langle(l_1, 1)\rangle$
$\langle\rangle$	$\langle\rangle$	$\langle(l_3, 2)\rangle$

$U^{(5)} =$

	$\overline{b}$	$\overline{a}$
		b

$Q^{(5)} = \{(3, 1), (3, 2), (2, 2)\}$ $visited^{(5)} = \{\}$

Step 6 $p = (3, 2)$; $index = -1$:

$collisions\_grid^{(6)} = collisions\_grid^{(5)}$ $U^{(6)} = U^{(5)}$

$Q^{(6)} = \{(3, 1), (2, 2)\}$ $visited^{(6)} = \{(3, 2)\}$

Step 7 $p = (3, 1)$; $index = -1$:

$collisions\_grid^{(7)} = collisions\_grid^{(6)}$ $U^{(7)} = U^{(6)}$

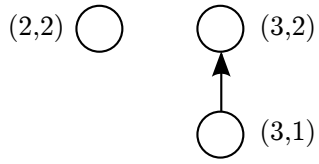
$Q^{(7)} = \{(2, 2)\}$ $visited^{(7)} = \{(3, 1), (3, 2)\}$

Step 8 $p = (2, 2)$; $index = -1$:

$collisions\_grid^{(8)} = collisions\_grid^{(7)}$ $U^{(8)} = U^{(7)}$

$Q^{(8)} = \{\}$ $visited^{(8)} = \{(3, 1), (3, 2), (2, 2)\}$

Step 6 The dependency graph G_{dep} constructed by algorithm *ValidateSelection* for Phase 2 is the following:



Since there are no cycles in the graph, the selection obtained from Phase 1 is valid. Finally *CloseStep* produces the following configuration:

$$new\_grid = \begin{array}{|c|c|c|}\hline & \overline{b} & \overline{a} \\ \hline & & b \\ \hline \end{array} = W_1$$

It is easy to see that, by randomly selecting different positions, algorithm *GenerateSelection*1 can generate all the possible resulting states W_1, W_2, W_3.

Example 5.1.2. Consider the Spatial P system model defined in the previous example. In the following we show a possible execution of $SimulateStep(W_0)$, using *GenerateSelection*2 algorithm.

Setup $SetupStep(W_0, t)$ produces the following initial state:

$$collisions\_grid^{(0)} = \begin{array}{|c|c|c|}\hline \langle\rangle & \langle(l_2,1)\rangle & \langle(l_1,1),(l_3,1)\rangle \\ \hline \langle\rangle & \langle\rangle & \langle\rangle \\ \hline \end{array} \qquad Q^{(0)} = \{(3,2)\}$$

where l_1, l_2, l_3 are as before. In this case, $collisions\_grid^{(0)}$ corresponds to the following tentative configuration:

$$U^{(0)} = \begin{array}{|c|c|c|}\hline & \overline{b} & \overline{a}\overline{b} \\ \hline & & \\ \hline \end{array}$$

The simulation algorithm performs the following steps:

Step 1.a $p = (3, 2)$

$$\begin{aligned} &collisions = \langle(l_1,1),(l_3,1)\rangle \\ &index = 1 \qquad\qquad \textit{(randomly selected)} \\ &collisions\_grid[3,2] = \langle(l_1,1)\rangle \\ &collisions\_grid[3,1] = \langle(l_3,2)\rangle \end{aligned}$$

Then, at the end of the step:

$$Q^{(1a)} = \{(3,1)\}$$

$$collisions\_grid^{(1a)} = \begin{array}{|c|c|c|}\hline \langle\rangle & \langle(l_2,1)\rangle & \langle(l_1,1)\rangle \\ \hline \langle\rangle & \langle\rangle & \langle(l_3,2)\rangle \\ \hline \end{array} \qquad U^{(1a)} = \begin{array}{|c|c|c|}\hline & \overline{b} & \overline{a} \\ \hline & & b \\ \hline \end{array}$$

Step 2.a $p = (3, 1)$

$$collisions = \langle (l_3, 2) \rangle$$
$$index = 1$$
$$collisions\_grid[3, 1] = \langle (l_3, 2) \rangle$$

$Q^{(2a)} = \{\}$

$collisions\_grid^{(2a)} =$

$\langle\rangle$	$\langle (l_2, 1) \rangle$	$\langle (l_1, 1) \rangle$
$\langle\rangle$	$\langle\rangle$	$\langle (l_3, 2) \rangle$

$U^{(2a)} =$

	$\overline{b}$	$\overline{a}$
		b

The selection obtained in this case is the same as in the execution of algorithm *GenerateSelection*1 shown previously. In fact, $collisions\_grid^{(2a)}$ is the same as $collisions\_grid^{(5)}$ previously shown. Therefore, the dependency graph G_{dep} constructed by algorithm *ValidateSelection* for Phase 2 is the same as before. Since there are no cycles in the graph, the selection obtained from Phase 1 is valid, and *CloseStep* produces configuration W_1.

Another possible execution is the following, which continues after **Setup** step:

Step 1.b $p = (3, 2)$

$$collisions = \langle (l_1, 1), (l_3, 1) \rangle$$
$$index = 2 \qquad \textit{(different value selected w.r.t. step 1.a)}$$
$$collisions\_grid[3, 2] = \langle (l_3, 1) \rangle$$
$$collisions\_grid[2, 2] = \langle (l_2, 1), (l_1, 2) \rangle$$

$Q^{(1b)} = \{(2, 2)\}$

$collisions\_grid^{(1b)} =$

$\langle\rangle$	$\langle (l_2, 1), (l_1, 2) \rangle$	$\langle (l_3, 1) \rangle$
$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

$U^{(1b)} =$

	$\overline{b}\overline{a}$	$\overline{b}$

Step 2.b $p = (2, 2)$

$$collisions = \langle (l_2, 1), (l_1, 2) \rangle$$
$$index = 1 \qquad \textit{(randomly selected)}$$
$$collisions\_grid[2, 2] = \langle (l_2, 1) \rangle$$
$$collisions\_grid[1, 2] = \langle (l_1, 3) \rangle$$

$Q^{(2b)} = \{(1, 2)\}$

$collisions\_grid^{(2b)} =$

$\langle (l_1, 3) \rangle$	$\langle (l_2, 1) \rangle$	$\langle (l_3, 1) \rangle$
$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

$U^{(2b)} =$

a	$\overline{b}$	$\overline{b}$

Step 3.b $p = (1, 2)$

$$collisions = \langle (l_1, 3) \rangle$$
$$index = 1$$
$$collisions\_grid[1, 2] = \langle (l_1, 3) \rangle$$

$Q^{(3b)} = \{\}$

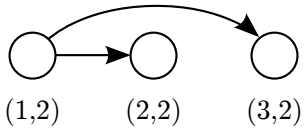
$collisions\_grid^{(3b)} =$

$\langle (l_1, 3) \rangle$	$\langle (l_2, 1) \rangle$	$\langle (l_3, 1) \rangle$
$\langle\rangle$	$\langle\rangle$	$\langle\rangle$

$U^{(3b)} =$

a	$\bar{b}$	$\bar{b}$

Step 4.b The dependency graph G_{dep} constructed by $ValidateSelection$ is:



As for the previous case, there are no cycles in the graph. The final configuration constructed by $CloseStep$ is the following:

$new\_grid =$

a	$\bar{b}$	$\bar{b}$

$= W_2$

Finally, it is easy to see that W_3 is the only other possible configuration that can be obtained, by selecting $index = 2$ at step **2.b**.

Example 5.1.3. This example shows a case in which Phase 1, by using algorithm $GenerateSelection2$, produces a selection which is not valid, as it violates the property of priority and maximality (c) from Definition 4.2.4. Consider a Spatial P system with objects $V = \emptyset$, $E = \{a, b, c\}$, and the following initial configuration:

$W_0 =$

a	b	c

Let denote by $p_1 = (1, 2)$, $p_2 = (2, 2)$, $p_3 = (3, 2)$, the positions above objects a, b, and c, respectively. The evolution rules for this model are the following, that are expressed, for the sake of simplicity, using *absolute* positions instead of relative positions:

$$r_a^{(2)} = a_{(1,1)} \to a_{p_2} \;>\; r_a^{(1)} = a_{(1,1)} \to a_{p_3}$$
$$r_b^{(2)} = b_{(2,1)} \to b_{p_2} \;>\; r_b^{(1)} = b_{(2,1)} \to b_{p_1}$$
$$r_c^{(2)} = c_{(3,1)} \to c_{p_1} \;>\; r_c^{(1)} = c_{(3,1)} \to c_{p_2}$$

$SetupStep$ tries to apply the highest priority rule $r_a^{(2)}$, $r_b^{(2)}$, and $r_c^{(2)}$, obtaining the following tentative configuration:

$U^{(0)} =$

c	ab	

Suppose that the conflict in position $(2,2)$ is resolved by confirming object a, and cancelling the application of rule $r_b^{(2)}$. Thus, the algorithm considers the application of $r_b^{(1)}$ to b, giving:

$$U^{(1)} = \begin{array}{|c|c|c|}\hline bc & a & \\ \hline & & \\ \hline \end{array}$$

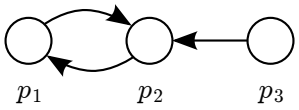
Supposing that the conflict in $(1,2)$ is now resolved by confirming the application of $r_b^{(1)}$ and cancelling $r_c^{(2)}$, the subsequent tentative configuration is:

$$U^{(2)} = \begin{array}{|c|c|c|}\hline b & ac & \\ \hline & & \\ \hline \end{array}$$

This last operation has produced a new conflict in position $(2,2)$, that we suppose is resolved by confirming the application of $r_c^{(1)}$ and cancelling $r_a^{(2)}$, giving:

$$U^{(3)} = \begin{array}{|c|c|c|}\hline b & c & a \\ \hline & & \\ \hline \end{array}$$

This last configuration obtained from Phase 1 corresponds to selecting the rules $r_a^{(1)}$, $r_b^{(1)}$, and $r_c^{(1)}$. However, this selection of rules is not valid, since we can replace the application $r_b^{(1)}$ with $r_b^{(2)}$ and $r_c^{(1)}$ with $r_c^{(2)}$, obtaining a greater selection (according to the priority relation defined by Equation 4.2) that satisfies the properties of applicability (a) and mutual exclusivity (b) from Definition 4.2.4. Such a replacement is actually found by *ValidateSelection* algorithm, which corresponds to the only cycle in the dependency graph:



Finally, the valid selection of rules $r_a^{(1)}$, $r_b^{(2)}$, $r_c^{(2)}$ produces the following configuration:

$$W_1 = \begin{array}{|c|c|c|}\hline c & b & a \\ \hline & & \\ \hline \end{array}$$

5.1.2 Correctness of the restricted algorithm

Theorem 5.1.1. *The restricted simulation algorithm, using either algorithm GenerateSelection*1 *or algorithm GenerateSelection*2 *as implementation of Phase 1, behaves correctly with respect to the semantics of Spatial P systems, that is:*

$$\forall S.\quad W \xrightarrow{S} W' \quad \Longleftrightarrow \quad W' \textit{ can be generated by } SimulateStep(W)).$$

Proof. **Case** $\Longleftarrow$) Phase 1 of the algorithm may produce a selection S which is not valid because it violates property (c) on priority and maximality. By definition of the priority relation on selections $\preceq$ (Equation 4.2), this means that there exists a valid selection $S^\top \succ S$ (i.e. $S^\top$ satisfies all properties of Definition 4.2.4). In Phase 2, such a valid selection $S^\top$ is determined.

Let S_i be a selection satisfying properties (a) and (b), but violating (c). In order to find a selection $S_{i+1} \succ S_i$ satisfying both (a) and (b), it is necessary to find a set of positions in which some rule is applied in each position, and such that either (i) the multiplicity of a rule is increased ($\mathbf{m}(\mathcal{R}, r) < \mathbf{m}(\mathcal{R}', r)$), or (ii) there is a rule with higher priority whose multiplicity is increased ($\exists r' > r.\ \mathbf{m}(\mathcal{R}', r') > \mathbf{m}(\mathcal{R}, r)$). Since in each position there is at most one object, case (i) cannot occur. For the same reason, in case (ii), increasing the multiplicity of a rule $r' > r$, with respect to the rule r currently applied in a position, actually means *replacing* the application of r with an application of r'.

Recall that Phase 1 produces a selection such that, for any position p in which a rule r is applied to an object a, all the positions which would be occupied by the application of any other p-enabled rule $r' \in R^{(a)}$, with $r' > r$, are already occupied. This implies that, for each replacement of a rule that results in a position x being freed and a position y being occupied (which can be the same as x), there are both (i) a replacement which occupies x and (ii) a replacement which frees position y. Formally, for any position p in which a rule $r^{(p)} = a_{(0,0)}[\overline{\pi}] \to a'_{p_1}$ is replaced by a rule $r'^{(p)} = a_{(0,0)}[\overline{\pi}'] \to a''_{p_2}$, there exists a position p' in which a rule $r^{(p')} = b_{(0,0)}[\overline{\pi}''] \to b'_{q_1}$ is replaced by some rule $r'^{(p')}$ such that $p + p_2 = p' + q_1$. This corresponds to finding a cycle in the dependency graph G_{dep} constructed in algorithm *ValidateSelection*. By applying such a replacement, we obtain a selection $S_{i+1} \succ S_i$. Note that there exists a selection $S_{i+1} \succ S_i$ iff there is such a replacement of rules. Since there are no infinite chains of selections $S_1 \prec S_2 \prec \cdots \prec S_i \prec \cdots$ in which all the selections satisfy properties (a) and (b) but not (c), by iterating this procedure we eventually reach a valid selection $S^\top$.

Case $\Longrightarrow$) This part of the proof is presented by considering the two proposed implementations of Phase 1, namely algorithms *GenerateSelection*1 and *GenerateSelection*2.

In the following, we denote by $\longrightarrow_1^{(1)}$ and $\longrightarrow_1^{(2)}$ the relations describing an iteration of the two algorithms implementing Phase 1, *GenerateSelection*1 and *GenerateSelection*2, respectively. That is $S_1 \longrightarrow_1^{(1)} S_2$ iff, starting from selection S_1, S_2 is reachable after one iteration of algorithm *GenerateSelection*1, and similarly for relation $\longrightarrow_1^{(2)}$ and algorithm *GenerateSelection*2. Moreover, let $\Longrightarrow_1^{(1)}$ and $\Longrightarrow_1^{(2)}$ be transitive closures of $\longrightarrow_1^{(1)}$ and $\longrightarrow_1^{(2)}$, respectively. As regards Phase 2, we denote by $\longrightarrow_2$ the relation describing an iteration of the *ValidateSelection* algorithm, and by $\Longrightarrow_2$ the transitive closure of $\longrightarrow_2$. Finally, let *target* be the function that gives the resulting (relative) position associated with a rule, that is: $target(a_{(0,0)}[\overline{\pi}] \to a'_{p'}) = p'$.

GenerateSelection1 Let $\min(\mathcal{S}^{\text{filling}})$ be the set of minimal elements of $\mathcal{S}^{\text{filling}}$. That is, $\min(\mathcal{S}^{\text{filling}}) \subseteq \mathcal{S}^{\text{filling}}$ is such that: (i) $\forall S \in \min(\mathcal{S}^{\text{filling}}).\ \nexists S' \in \mathcal{S}^{\text{filling}}.\ S' \prec S$; and (ii) $\forall S' \in \mathcal{S}^{\text{filling}}.\ \exists S \in \min(\mathcal{S}^{\text{filling}}).\ S \preceq S'$.

In order to prove the thesis, it is sufficient to prove that:

1. $\forall S \in \min(\mathcal{S}^{\text{filling}}).\ \bot \Longrightarrow_1 S$, that is any minimal filling selection can be obtained from *GenerateSelection*1;

2. $\forall S \in \mathcal{S}^{\text{valid}}.\ \exists S' \in \min(\mathcal{S}^{\text{filling}}).\ S' \Longrightarrow_2 S$, that is for any valid selection S there is a minimal filling selection S' such that algorithm *ValidateSelection* can obtain S starting from S'.

As regards part 1, we prove that, given a minimal filling selection S, for all selections $S' \prec S$, *GenerateSelection*1 algorithm can perform an iteration from S' to some other S'' which is still less than or equal to selection S. That is, given a minimal filling selection $S \in \min(\mathcal{S}^{\text{filling}})$:

$$\forall S' \in \mathcal{S}^{(a,b)}.\ S' \prec S \implies \exists S'' \in \mathcal{S}^{(a,b)}.\ S' \longrightarrow_1^{(1)} S'' \ \wedge\ S' \prec S'' \preceq S$$

This implies that *GenerateSelection*1 can generate any minimal filling selection S.

Given a configuration W, and a selection $S \in \mathcal{S}_W^{(a,b)}$, we denote by $occ(S)$ the set of occupied positions in the configuration resulting from the application of S to W, that is, given $W' = apply(W, S)$, then $occ(S) = \{p \mid w'_p \neq \emptyset\}$.

Consider a selection $S \in \min(\mathcal{S}^{\text{filling}})$, and a selection $S' \in \mathcal{S}^{(a,b)}$ such that $S' \prec S$. Let us consider the sets of occupied position for selections S and S', namely $occ(S)$ and $occ(S')$, respectively. There are two possible cases: either $occ(S) = occ(S')$, or not.

On one hand, assume that $occ(S) = occ(S')$. Then there must exist a position p in which the currently applied rule r, i.e. $S'(p) = \{r\}$, can be replaced by the algorithm with a rule $r' \succ r$ such that $target(r') \notin occ(S)$, and which is smaller than the rule r'' selected for p in S, i.e. $S(p) = \{r''\} \implies r'' \succ r'$. In fact, suppose that such a position does not exist. Since $S \in \mathcal{S}^{\text{filling}}$, also $S' \in \mathcal{S}^{\text{filling}}$ and, therefore, S would not be minimal, contradicting the hypothesis.

On the other hand, assume that $occ(S) \neq occ(S')$. Note that the number of occupied positions in S and S' is the same, that is $|occ(S)| = |occ(S')|$. Therefore, there exists a position $q \in occ(S) \setminus occ(S')$. Consider the selection S, and let p be the position in which there is a rule r'' applied in p that puts an object in q, that is $S(p) = \{r''\} \wedge target(r'') = q$. It is sufficient that the algorithm selects position p to search for a greater rule r' to replace the currently applied rule r, with $S'(p) = \{r\}$. In fact, the algorithm selects the lowest applicable rule r' such that $r'' \succeq r' \succ r$ and $target(r') \neq target(r)$. In the worst case, if no rules $\hat{r}$ such that $r \prec \hat{r} \prec r''$ can replace the currently applied rule r, then the algorithm selects rule r''. In such a case, the replacement could be forbidden only if $target(r'') = target(r')$. However, this is absurd, since it would imply that S is not minimal, contradicting the hypothesis.

As regards part 2, consider a valid selection $S \in \mathcal{S}^{\text{valid}}$. In order to obtain a minimal filling selection, it is sufficient to construct a sequence of filling selections $S = S_1 \succ S_2 \succ \cdots \succ S_k = S'$, where each S_{i+1} is obtained from S_i by replacing the rules selected for some positions with lower priority rules, ensuring that the set of positions occupied remains the same, i.e. $occ(S_{i+1}) = occ(S)$. This construction corresponds to the converse of algorithm $ValidateSelection$, and terminates as soon as a minimal filling selection $S' = S_k \in \min(\mathcal{S}^{\text{filling}})$ is obtained. Therefore $S' \Longrightarrow_2 S$.

GenerateSelection2 Consider the set $\mathcal{S}^{(a)}$ of selections that satisfy the property of applicability (a) from Definition 4.2.4, given the configuration W. The algorithm starts with a selection $\top$ (constructed by *SetupStep*) corresponding to the maximum element of $\mathcal{S}^{(a)}$, according to the priority ordering $\preceq$ defined by Equation 4.2.

In order to prove the thesis, it is sufficient to prove that any valid selection S can be obtained from Phase 1, that is for all valid selection S, $\top \Longrightarrow_1^{(2)} S$. We actually prove that, given a valid selection S, for all selections $S' \succ S$, *GenerateSelection*2 algorithm can perform an iteration from S' to some other S'' which is still greater than or equal to the

valid selection S. Formally, given a valid selection S, $\forall S' \succ S.\ \exists S''.\ S' \longrightarrow_1^{(2)} S'' \ \wedge\ S' \succ S'' \succeq S$. This implies that the algorithm can reach any valid selection S.

It is sufficient that the algorithm confirms a rule application $(p, \{r\}) \in S'$ such that no other rule application $(p', \{r'\}) \in S'$ conflicting with $(p, \{r\})$ is also in S. Formally, $\forall (p', \{r'\}) \in S \cap S'.\ (p', \{r'\}) \neq (p, \{r\}) \implies p' + target(r') \neq p + target(r)$. Since there are no conflicts in S it is always possible to select such a rule application. □

5.1.3 Algorithm complexity

In this section we discuss the time complexity of the algorithm for simulating a step. Let denote by m the number of objects of the Spatial P system, i.e. $m = |E|$. Recall that K denotes the maximum number of different rules having the same reactant object, i.e. $K = \max_{a \in E} |R^{(a)}|$. Moreover, let H denote the maximum number, among all rules, of different positions in which the promoters of a rule are contained, i.e. $H = \max\{\, \gamma \mid \exists (a_x)_{(0,0)}[(\pi_1)_{q_1} \dots (\pi_\gamma)_{q_\gamma}] \to (a_y)_p \in R\}$. Finally, recall that we assume that the algorithm works with a square grid of size $N \times N$, corresponding to a square skin membrane. The time complexity of the algorithm composing the *SimulateStep* algorithm are discussed in the following.

SetupStep The predominant computational cost is given by the iteration over all positions $p \in \{1, \dots, N\} \times \{1, \dots, N\}$ at lines 7–16 of the algorithm, where function *FindMoves* is called for each non-empty position. *FindMoves* has to retrieve the set of rules in which the reactant of each rule corresponds to the object contained in the considered position, which can be done in $\mathcal{O}(1)$. Then, it has to verify the applicability of each rule (i.e. if all the promoters are present in their relative positions), and generating the list of moves to be returned, which costs $\mathcal{O}(KH)$. Since there are m objects, the total computational cost is $\mathcal{O}(N^2 + mKH)$

GenerateSelection1 The algorithm iterates through positions in Q until it finds a position in which there is a rule of higher priority that can replace the currently selected rule for the position. The size of Q is bounded by the number of objects m, i.e. $|Q| \leq m$. In the worst case, the algorithm examines all positions in Q, and finds such a replacement only for the last position chosen. This causes all other occupied positions, which were put in the set *visited*, to be put back into Q in order to be considered again subsequently. Moreover, assume that, once a replacement is found, it involves replacing a rule with the rule of immediately higher priority.

The number of rules not yet considered is initially not grater than mK, since each object has at most K possible moves. The worst case corresponds to visiting all those rules, and finding the replacement only for the last rule considered. Since, once a rule is applied, it is never considered again, the algorithm performs at most mK steps, where each step means visiting all the rules not yet considered. Since the number of rules not yet considered decreases by one every step, the total cost of the outer loop of *GenerateSelection*1 is $\sum_{i=0}^{mK} \mathcal{O}(i) = \mathcal{O}(m^2K^2)$. Finally, since constructing the initial set of occupied positions Q at line 2 costs $\mathcal{O}(N^2)$, the total cost of the algorithm is $\mathcal{O}(N^2 + mKH + m^2K^2)$.

GenerateSelection2 In order to determine the cost of *GenerateSelection* algorithm (Algorithm 4), it is useful to introduce a lemma which allows to determine the maximum

number of iterations performed by the algorithm.

Let $Q^{(s)}$ denote the queue at the beginning of iteration s of the while loop on line 3. Let $L^{(s)}$ denote the sum of the number of moves not yet considered for every position of *collisions_grid*, at the beginning of iteration s, namely:

$$L^{(s)} = \sum_{(i,j)} \text{sum}(collisions\_grid^{(s)}[i,j])$$

$$\text{sum}(\langle (moves_1, l_1), \ldots, (moves_k, l_k) \rangle) = \sum_{\gamma=1}^{k} \text{length}(moves_\gamma) - n_\gamma + 1$$

Between an iteration and the subsequent one of the loop on line 3, the size of the set Q and the total number of moves not yet considered are related as stated in the following lemma.

Lemma 5.1.2. *After each iteration of GenerateSelection2 (Algorithm 4), the sum of the size of the set Q and the total number of all moves not yet considered decreases by at least one:*

$$|Q^{(s+1)}| + L^{(s+1)} \leq |Q^{(s)}| + L^{(s)} - 1.$$

Proof. Let $p = (i,j) \in Q^{(s)}$ be the position selected at iteration s. There are two possible cases:

1. $collisions\_grid^{(s)}[i,j] = \langle (moves_1, n_1) \rangle$
   In this case the selected application is with $index = 1$, therefore *collisions_grid* is such that $collisions\_grid^{(s+1)}[i,j] = collisions\_grid^{(s)}[i,j] = \langle (moves_1, n_1) \rangle$, and no other position in *collisions_grid* is changed, therefore $L^{(s+1)} = L^{(s)}$. Finally, since $Q^{(s+1)} = Q^{(s)} \setminus \{p\}$, we have that $|Q^{(s+1)}| = |Q^{(s)}| - 1$, proving the thesis.

2. $collisions\_grid^{(s)}[i,j] = \langle (moves_1, n_1), \ldots, (moves_k, n_k) \rangle$, with $k \geq 2$.
   Let *index* be the selected application. At the end of the iteration it holds that $collisions\_grid^{(s+1)}[i,j] = \langle (moves_{index}, n_{index}) \rangle$ and for each other $\gamma \neq index$ the pair $(moves_\gamma, n_\gamma + 1)$ is appended to $collisions\_grid[i',j']$, for a different position $(i',j') \neq p$. Therefore $L^{(s+1)} = L^{(s)} - (k-1)$. As regards the set Q, at most $k-1$ new position are added to it, hence $|Q^{(s+1)}| \leq |Q^{(s)}| - 1 + (k-1)$. □

A direct consequence of Lemma 5.1.2 is that the algorithm performs at most $|Q^{(0)}| + L^{(0)}$ iterations. At the beginning, there cannot be more than $\lfloor m/2 \rfloor$ colliding positions, therefore the size of $Q^{(0)}$ is bounded by $\lfloor m/2 \rfloor$: $|Q^{(0)}| \leq \lfloor m/2 \rfloor$. As regards the value $L^{(0)}$, since each object has at most $K+1$ possible moves, it holds that $L^{(0)} \leq m(K+1)$. Thus the algorithm performs $\mathcal{O}(mK)$ iterations.

Each iteration involves iterating through the list $collisions\_grid[p]$, for some position p. At any iteration s and any position p, the number of objects colliding in p, namely $length(collisions\_grid^{(s)}[p])$, is such that $length(collisions\_grid^{(s)}[p]) \leq m$. Each iteration therefore costs $\mathcal{O}(m)$ and, consequently, the cost of the loop, on the whole, is $\mathcal{O}(m^2K)$.

Finally, since constructing the initial set of occupied positions Q at line 2 costs $\mathcal{O}(N^2)$, the total complexity of the *GenerateSelection* algorithm is $\mathcal{O}(N^2 + mKH + m^2K)$.

ValidateSelection The dependency graph, constructed at the beginning of the algorithm, has at most $\sigma = mK$ edges, since the are m objects and at most K possible rule applications for each object. Thus, constructing the graph has complexity $\mathcal{O}(mK)$. The complexity of searching a cycle in the graph is equal to the number of edges in the graph. After each iteration, some edges are removed, thus the complexity of finding other cycles decreases. In the worst case, a cycle composed of only two nodes is found in each iteration, therefore the algorithm performs at most $\lceil \sigma/2 \rceil$ iterations. In that case, the cost of the i^{th} iteration is $\mathcal{O}(i)$, since the prevalent cost comes from updating the graph at line 7. In fact, the removal of an edge costs $\mathcal{O}(1)$, and each edge is removed at most once. Moreover, the cost of replacing edges is inversely proportional to the step, as the maximum number of edges exiting from a node decreases after each iteration. As regards finding the index of the application to be confirmed, corresponding to the rule with maximum priority (line 5), note that this information can be associated with the edges, by labelling them with the maximum index during the construction of the graph. Thus that operation does not need to be performed at each step, and its cost is already included in that of constructing the graph $\mathcal{O}(mK)$. Therefore, the total cost of $ValidateSelection$ algorithm is $\sum_{i=0}^{\lceil \sigma/2 \rceil} \mathcal{O}(i) = \mathcal{O}(\sigma^2) = \mathcal{O}(m^2K^2)$.

CloseStep The cost of $CloseStep$ is just $\mathcal{O}(N^2)$, since it iterates through all positions performing only constant time $\mathcal{O}(1)$ operations.

Given the cost of each algorithm composing the $SimulateStep$ algorithm (Algorithm 1), as detailed in the previous paragraphs, the total time complexity of the $SimulateStep$ algorithm is $\mathcal{O}(N^2 + mKH + m^2K^2)$. The main complexity of the algorithm comes from the $ValidateSelection$ algorithm, whose worst-case complexity is $\mathcal{O}(m^2K^2)$. However, real-life models rarely exhibit the worst case behaviour. Therefore, in a complete simulation, most of the time is spent in Phase 1, implemented by one of the two algorithms $GenerateSelection1$ and $GenerateSelection2$. This motivates the use of the more-efficient algorithm $GenerateSelection2$ in place of $GenerateSelection1$, since Phase 1 takes up the most of the simulation time in real models.

5.1.4 Priority relation on rules as a partial order

The simulation algorithm we have presented requires the Spatial P system model to be constrained as specified in Section 5.1. We discuss in the following how the algorithm can be extended to deal with a priority relation on rules which is a partial order, thus removing the constraint to be a *total* order. In order to make the algorithm work in this case, it is sufficient to modify the behaviour of the $FindMoves$ function. In particular, $FindMoves$ has to determine a *linear ordering* of rules, which is defined as follows.

Definition 5.1.2. Given a set of rules R, a *linear ordering* of the rules in R is an ordered sequence of rules $\langle r_1, r_2, \ldots, r_k \rangle$ such that:

a. $k = |R| \ \wedge \ \forall r \in R.\ \exists i.\ r = r_i$;

b. $\forall i, j.\ r_i > r_j \implies i < j$.

The first constraint ensures that the sequence $\langle r_1, \ldots, r_k \rangle$ is a permutation of the elements from R, while the second constraint ensures that it respects the relative priority

ordering between the rules. Note that a sequence $\langle r_1, \ldots, r_k \rangle$ represents the inverse of a *linear extension* of the partially ordered set of rules R.

Each time the algorithm $FindMoves$ is executed, it has to randomly generate, for a each non-empty position p, a linear ordering $l = \langle r_1, \ldots, r_k \rangle$ of the rules which are enabled in p, that is $Enabled(R, p) = \{r_1, \ldots, r_k\}$. Then it uses that sequence of rules l as the ordering for generating a list of pairs $\langle (\overline{q_1}, x_1), \ldots, (\overline{q_k}, x_k), (p, a_y) \rangle$, where each pair $(\overline{q_i}, x_i)$, with $1 \leq i \leq k$, corresponds to the application of rule r_i to the object in p. Recall that the last pair (p, a_y) of the list returned by $FindMoves$ describes the lack of application of any rule to the object a_y in p.

For example, let $\{r_1, r_2, r_3, r_4\}$ be the set of rules enabled in a position p, with the following relative priorities: $r_1 > r_2$, $r_2 > r_4$, $r_3 > r_4$. There are three possible linear orderings of rules: $l_1 = \langle r_1, r_2, r_3, r_4 \rangle$, $l_2 = \langle r_1, r_3, r_2, r_4 \rangle$, and $l_3 = \langle r_3, r_1, r_2, r_4 \rangle$. Algorithm $FindMoves$ then randomly generates a list of moves corresponding to one of such linear orderings of rules.

Theorem 5.1.3. *The SimulateStep algorithm, in which FindMoves has been modified to randomly generate a* linear ordering *of the rules enabled in a position as the ordering of the rules, behaves correctly with respect to the semantics of Spatial P systems.*

Proof. Any selection generated by the linear algorithm is valid with respect to the original partial order of the rules. In fact, given a selection of rules S generated by the algorithm and a position p, the property (b) of Definition 5.1.2 implies that any higher priority rule than the rule applied in p has been considered by the algorithm. In particular, the selection produced in Phase 1 of the algorithm is also filling for the original partial order of the rules. As regards Phase 2, if there are no replacements which can be performed, then this implies that there are no possible replacements also considering the original partial order of the rules. However, note that a replacement performed either in Phase 1 or Phase 2 may not correspond to obtaining a greater selection of rules with respect to the original partial order of rules. For example, consider the rules r_1, r_2, r_3, such that $r_1 > r_3$, $r_2 > r_3$, and the linear ordering $\langle r_1, r_2, r_3 \rangle$, for some position. Suppose the currently selected rule is r_2, and that the algorithm replaces r_2 with r_3. Then, in this case, r_3 is greater than r_2 w.r.t. the linear ordering, but not w.r.t. the original partial order.

On the other hand, assuming that $FindMoves$ can randomly generate *every* possible linear ordering of the rules ensures that every possible valid selection can be generated by the algorithm. In fact, in the case of using algorithm $GenerateSelection2$ as the implementation of Phase 1 of the algorithm, in order for the algorithm to be able to generate a given valid selection S, it is sufficient that for all positions p, the list returned by $FindMoves(p)$ corresponds to a linear ordering of rules $r_1, \ldots, r_{x-1}, r_x, r_{x+1}, \ldots, r_k$ such that $r_x \in S(p)$ and $\forall i < x.\ r_i > r_x$. That is, for the selected rule r_x, the rules $r_1, \ldots, r_{x-1}$ which *precede* it in the linear ordering are the all and only rules having priority higher than r_x. In this way, the selection S is also valid with respect to the linear ordering, thus it can be generated by the algorithm, according to Theorem 5.1.1. In case of using algorithm $GenerateSelection1$, instead, it is sufficient to consider a linear ordering of rules $r_1, \ldots, r_{x-1}, r_x, r_{x+1}, \ldots, r_k$, with $r_x \in S(p)$, such that the rules $r_{x+1}, \ldots, r_k$ which *follow* r_x in the linear ordering are the all and only rules having *lower* priority than r_x, i.e. $\forall i > x.\ r_i < r_x$. □

5.2 Including ordinary objects

In this section we discuss how the restricted algorithm presented in the previous section can be extended to deal with ordinary objects. We consider rules in which all the reactants are contained in position $(0,0)$, that is they are of the following form:

$$(u)_{(0,0)}[\overline{\pi}] \to (v_1)_{t_1} \dots (v_h)_{t_h} \tag{5.1}$$

where $u, v_i \in (V \cup E)^*$, and such that each rule contains either (i) *exactly* one mutually exclusive object among the reactants, and *exactly* one mutually exclusive object among the products, or (ii) no mutually exclusive objects at all among reactants and products. There are no restrictions on promoters. Given a rule r, let us denote by $n_\eta(r)$ the number of mutually exclusive objects among reactants, and by $n_T(r)$ the number of mutually exclusive objects among products, as follows:

$$n_\eta((u)_{(0,0)}[\overline{\pi}] \to (v_1)_{t_1} \dots (v_h)_{t_h}) = \sum_{x \in E} \mathbf{m}\,(u, x)$$

$$n_T((u)_{(0,0)}[\overline{\pi}] \to (v_1)_{t_1} \dots (v_h)_{t_h}) = \sum_{x \in E} \mathbf{m}\left(\biguplus_{i=1}^{h} v_i, x\right)$$

Let us denote by $\mathcal{R}^{\text{ord}}$ the set of all rules, having the form specified by Equation 5.1, in which no mutually exclusive objects appear as reactant nor as product. Moreover, let us denote by $\mathcal{R}^{\text{me}}$ the set of all rules, having the form specified by Equation 5.1, having exactly one mutually exclusive object among the reactants and one among the products. Formally:

$$\mathcal{R}^{\text{ord}} = \{r \mid n_\eta(r) = 1 \wedge n_T(r) = 1\}; \tag{5.2}$$

$$\mathcal{R}^{\text{me}} = \{r \mid n_\eta(r) = 0 \wedge n_T(r) = 0\}. \tag{5.3}$$

The set of all rules allowed is thus $\mathcal{R}^{\text{ord}} \cup \mathcal{R}^{\text{me}}$.

As regards the priority relation on rules, we require it to correspond to an *extended rule ordering*, which is represented by a sequence of sets of rules $R_1, \dots, R_k$, according to the following definition.

Definition 5.2.1. Given a set of rules R, an *extended rule ordering* of R is a sequence of sets of rules $R_1, \dots, R_k$ such that:

a. $R_1, \dots, R_k$ are a partition of the set R

b. $\forall i.\ R_i \subset \mathcal{R}^{\text{me}} \vee R_i \subset \mathcal{R}^{\text{ord}}$

c. $\forall i.\ R_i \subset \mathcal{R}^{\text{me}} \implies |R_i| = 1$

Each set R_i of the partition represents a set of rules which can be applied together. A sequence $R_1, \dots, R_k$ is in descending order of priority with respect to the relative priorities among rules. Constraint (b) requires that each set R_i contains either only mutually exclusive rules from $\mathcal{R}^{\text{me}}$, or only ordinary rules from $\mathcal{R}^{\text{ord}}$. Constraint (c) requires that each rule with mutually exclusive objects from $\mathcal{R}^{\text{me}}$ appears as a singleton set. The partial order on rules $\leq$ entailed by an extended rule ordering $ext(R) = R_1, \dots, R_k$ is defined as follows: $r > r' \iff \exists i, j.\ i < j \wedge r \in R_i \wedge r' \in R_j$.

Finally, for simplicity in the presentation, we still assume that the skin membrane membrane is square, having size $N \times N$.

5.2.1 Simulation algorithm

We present an extension of *SimulateStep* algorithm (Algorithm 1). In the following we discuss an extension based on algorithm *GenerateSelection*1 (Algorithm 3). Algorithm *GenerateSelection*2 (Algorithm 4) can also be extended to deal with ordinary objects.

Note that any selection $S \in \mathcal{S}_W^{(a,b)}$, for some configuration W, satisfying the constraints of applicability (a) and mutual exclusivity (b) from Definition 4.2.4, is such that, considering each position p, there is at most one mutually exclusive rule $r \in \mathcal{R}^{\text{me}}$ selected in p, while there can be any number of ordinary rules from $\mathcal{R}^{\text{ord}}$ selected. This implies that, in any position, there is always at most one set of rules $R_i \subset \mathcal{R}^{\text{me}}$, from the extended ordering $ext(R)$, which is selected.

The simulation of a step by the extended algorithm works analogously to the restricted algorithm defined in Algorithm 1. Before introducing the details of the algorithm, we need some auxiliary definitions.

We restrict the priority relation $\preceq$ defined as Equation 4.1 to the following priority relation, which compares two multisets of rules, such that each multiset contains at most one mutually exclusive rule from $\mathcal{R}^{\text{me}}$. The following *ordinary-rules limited ordering relation* takes into account only ordinary rules having higher priority than the mutually exclusive rule selected, if any.

Definition 5.2.2. Let $ext(R) = R_1, \ldots, R_k$ be an extended ordering. Let $\mathcal{R}, \mathcal{R}'$ be such that $\sum_{r \in \mathcal{R}^{\text{me}}} \mathbf{m}(\mathcal{R}, r) \leq 1$ and $\sum_{r \in \mathcal{R}^{\text{me}}} \mathbf{m}(\mathcal{R}', r) \leq 1$. The *ordinary-rules limited ordering relation* $\prec^{\text{up}}$ is defined as follows:

$$\begin{aligned}
\mathcal{R} \prec^{\text{up}} \mathcal{R}' \iff\ & \forall R_i \subset \mathcal{R}^{\text{ord}}. \\
& (\exists j.\ R_j = \{r'\} \subset \mathcal{R}^{\text{me}} \wedge \mathbf{m}(\mathcal{R}, r') = 1 \implies i < j) \implies \\
& \quad (\forall r \in R_i.\ \mathbf{m}(\mathcal{R}', r) < \mathbf{m}(\mathcal{R}, r) \implies \\
& \qquad \exists r' \in R_j \subset \mathcal{R}^{\text{ord}}.\ j < i \wedge \mathbf{m}(\mathcal{R}', r') > \mathbf{m}(\mathcal{R}, r'))
\end{aligned}$$

Note that $\forall \mathcal{R}, \mathcal{R}'.\ \mathcal{R} \preceq^{\text{up}} \mathcal{R}' \implies \mathcal{R} \preceq \mathcal{R}'$. Intuitively, two multisets of rules $\mathcal{R}, \mathcal{R}'$ are such that $\mathcal{R} \prec^{\text{up}} \mathcal{R}'$ only if, considering each ordinary rule $r \in \mathcal{R}^{\text{ord}}$ having higher priority than some mutually exclusive rule selected $r' \in \mathcal{R}^{\text{me}}$, its multiplicity can be decreased only if there is some other higher priority (ordinary) rule whose multiplicity is increased. This actually corresponds to restrict the priority relation $\preceq$, defined as Equation 4.1, to the only ordinary rules of higher priority than r'.

The definition of ordinary-rules limited ordering relation is used to define *ordinary-rules maximal selections*, as follows.

Definition 5.2.3. Given a configuration W, a selection $S \in \mathcal{S}_W^{(a,b)}$ is said to be *ordinary-rules maximal* (*ord-max*) for W iff the following property holds:

$$\begin{aligned}
& \forall (p, \mathcal{R}) \in S. \\
& \quad (\forall i.\ R_i = \{r\} \in \mathcal{R}^{\text{me}} \wedge \mathbf{m}(\mathcal{R}, r) = 1 \implies \forall j > i.\ \forall r' \in R_j.\ \mathbf{m}(\mathcal{R}, r') = 0) \wedge \\
& \quad (\forall \mathcal{R}'.\ \mathcal{R} \prec^{\text{up}} \mathcal{R}' \implies \mathcal{S} \setminus \{(p, \mathcal{R})\} \cup \{(p, \mathcal{R}')\} \notin \mathcal{S}_W^{(a)})
\end{aligned}$$

We denote the set of all ordinary-rules maximal selections, for a configuration W, as $\mathcal{S}_W^{\text{ordmax}}$.

Intuitively, an ordinary-rules maximal selection $S \in \mathcal{S}^{(a,b)}$ is such that, in every position p, given a mutually exclusive rule applied in p, if any, then all higher priority *ordinary* rules are applied in a maximal way, an no lower priority rule is applied. In case no mutually exclusive rule are applied, then all the ordinary rules must be applied in a maximal way.

Given a mutually exclusive rule $r \in \mathcal{R}^{\text{me}}$, let $target(r)$ denote the relative target position of the only mutually exclusive object among the products, that is $target(r) = p$ such that $\exists x \in E.\ x \in T(r, p)$. Moreover, given a multiset of rules $\mathcal{R}$, let $target(\mathcal{R})$ denote the position occupied by the object corresponding to the only mutually exclusive rule $r \in \mathcal{R} \cap \mathcal{R}^{\text{me}}$, if any. That is $target(\mathcal{R}) = target(r)$, if $\exists r \in \mathcal{R} \cap \mathcal{R}^{\text{me}}$; otherwise $target(\mathcal{R}) = (0, 0)$.

Definition 5.2.4. Given a selection $S \in \mathcal{S}^{(a)}$, and the extended ordering $ext(R)$, the set of improved multisets of rules $improved(S, p)$ for the position p is defined as follows:

$$\begin{aligned}
\mathcal{R}' \in improved(S,p) \iff\ & \mathcal{R} = S(p) \wedge \exists i.\ R_i = \{r\} \subset \mathcal{R}^{\text{me}} \wedge \\
& (\forall j.\ R_j = \{r'\} \subset \mathcal{R}^{\text{me}} \wedge \mathbf{m}(\mathcal{R}, r') = 1 \implies i < j) \wedge \\
& \mathbf{m}(\mathcal{R}, r) = 0 \wedge \mathbf{m}(\mathcal{R}', r) = 1 \wedge \\
& \forall j < i.\ \forall r' \in R_j.\ \mathbf{m}(\mathcal{R}', r') = \mathbf{m}(\mathcal{R}, r') \wedge \\
& \forall j > i.\ \forall r' \in R_j.\ \mathbf{m}(\mathcal{R}', r') = 0 \wedge \\
& \mathcal{S} \setminus \{(p, \mathcal{R})\} \cup \{(p, \mathcal{R}')\} \in \mathcal{S}^{(a)}
\end{aligned}$$

First of all, note that $S \in \mathcal{S}^{(a)}$ satisfies the constraint (a) of applicability, therefore *at most* one mutually exclusive rule is selected in each position. Let $\mathcal{R} = S(p)$ be the currently selected multiset of rules for the position p. The set of improved multisets of rules $improved(S, p)$ corresponds to all the possible multiset of rules $\mathcal{R}'$ such that:

- there exists a mutually exclusive rule $r \in \mathcal{R}^{\text{me}}$, which is selected in $\mathcal{R}'$ but not in $\mathcal{R}$, and such that the priority of r is higher than the mutually exclusive rule $r' \in \mathcal{R}^{\text{me}}$ selected in $\mathcal{R}$, if any, namely $\exists r' \in \mathcal{R}^{\text{me}} \wedge \mathbf{m}(\mathcal{R}, r') = 1 \implies r > r'$;
- the multiplicity of all higher priority rules $r' > r$ is the same in $\mathcal{R}'$ as in $\mathcal{R}$, namely $\forall r' > r.\ \mathbf{m}(\mathcal{R}', r') = \mathbf{m}(\mathcal{R}, r')$;
- the multiplicity of all lower priority rules $r' < r$ in $\mathcal{R}'$ is 0, namely $\forall r' < r.\ \mathbf{m}(\mathcal{R}', r') = 0$;
- the selection resulting from the replacement of $\mathcal{R}$ with $\mathcal{R}'$ in position p still satisfies the constraint of applicability (a), thus $\mathcal{R}'$ contains *exactly* one mutually exclusive rule.

The definition of improved multisets is the basis for the definition of a *filling* selection, which extends the definition given previously for the restricted case (see Definition 5.1.1).

Definition 5.2.5. Given a configuration W, an ordinary-rules maximal selection $S \in \mathcal{S}_W^{\text{ordmax}}$ is said to be *filling* for W iff the following property holds:

$$\begin{aligned}
& \forall p.\ \forall \mathcal{R}' \in improved(S,p). \\
& \quad target(S(p)) \neq target(\mathcal{R}') \implies \mathcal{S} \setminus \{(p, S(p))\} \cup \{(p, \mathcal{R}')\} \notin \mathcal{S}_W^{(b)}
\end{aligned}$$

We denote the set of all filling selections, for a configuration W, as $\mathcal{S}_W^{\text{filling}}$.

Algorithm 7 SimulateStepExt(W)

1: **let** S be such that $\forall p.\ S(p) = \emptyset$
2: $S_1 = MaximiseApplications(W, S)$
3: $S_2 = GenerateSelectionExt(S_1)$
4: $S_3 = ValidateSelectionExt(S_2)$
5: $S_4 = MaximiseApplications(W, S_3)$
6: **return** $apply(W, S_4)$

In this case, a filling selection is such that, given a multiset of rules $\mathcal{R}$ selected for application in a position p, it is not possible to replace $\mathcal{R}$ with any greater multiset $\mathcal{R}' \in improved(S, p)$, whose target position is different from that of $\mathcal{R}$, since that would create a conflict between mutually exclusive objects, by violating constraint (b). By the definition of *improved*, this means that it is not possible, in any position, to select a previously unselected mutually exclusive rule $r \in R_i \subset \mathcal{R}^{\text{me}}$ once the applications of all lower priority rules $r' \in R_h$, with $h > i$, are cancelled, but in which the multiplicity of all rules of higher priority is not changed. Moreover, such a replacement must cause a mutually exclusive object to be removed from a position and another one to be put in a *different* position.

The simulation algorithm for the extended case, defined as Algorithm 7, works as follows:

- *(phase 1)* generate a filling selection with respect to the extended ordering of rules $ext(R)$
- *(phase 2)* search for groups of positions in which it is possible to replace together, for all such positions, the current rule applications with application of higher priority rules;
- *(phase 3)* complete the obtained selection by maximising the application of ordinary rules from $\mathcal{R}^{\text{ord}}$, in every position.

5.2.2 Implementation

Algorithm *GenerateSelectionExt* implements phases 1 and 2; algorithm *ValidateSelectionExt* implements phase 2; and algorithm *MaximiseApplications* implements Phase 3. Both algorithm *GenerateSelection*1 and *GenerateSelection*2 can be extended to implement Phase 1 of the extended algorithm. In the following, we discuss the extension of simulation algorithm using *GenerateSelection*1.

The algorithm starts by constructing the minimal selection $\perp \in \mathcal{S}^{(a,b)}$. Such a selection $\perp$ is used by algorithm *MaximiseApplications*, defined in Algorithm 8, which constructs a ordinary-rule maximal selection $S_1 \in \mathcal{S}^{\text{ordmax}}$, in which no mutually exclusive rule is applied in any position. *MaximiseApplications* iterates through the extended rule ordering $R_1, \ldots, R_k$, from the highest priority to the lowest, by applying *only* ordinary rules in a maximal way. Recall that, differently from the restricted algorithm, more than one rule can be selected for application in a same position. Whenever a set of ordinary rules $R_i \subset \mathcal{R}^{\text{ord}}$ is encountered, the algorithm finds a *maximal* multiset of rules $\mathcal{R}'$ from R_i which is *applicable* to the objects currently available in position p, namely to

Algorithm 8 MaximiseApplications(W, S)

for all non-empty positions p **do**
  let $R_1, \ldots, R_k = ext(R)$
  for $i = 1$ **to** k **do**
    if $R_i \subset \mathcal{R}^{\text{ord}}$ **then**
      select randomly a *maximal* multiset of rules $\mathcal{R}'$ from R_i which is *applicable* to the objects $w_p \Downarrow \eta(S, p)$
      let $\mathcal{R} = S(p)$
      $S = S \setminus \{(p, \mathcal{R})\} \cup \{(p, \mathcal{R} \cup \mathcal{R}')\}$
    end if
  end for
end for
return S

Algorithm 9 GenerateSelectionExt1(S)

while $\exists p.\ \exists \mathcal{R}' \in improved(S, p).$
  $target(S(p)) \neq target(\mathcal{R}') \wedge \mathcal{S} \setminus \{(p, S(p))\} \cup \{(p, \mathcal{R}')\} \in \mathcal{S}^{(a,b)}$ **do**
  $S = S \setminus \{(p, S(p))\} \cup \{(p, \mathcal{R}')\}$
end while
return S

Algorithm 10 ValidateSelectionExt(S)

repeat
  let $G_{\text{dep}} = (V, E)$ be the dependenty graph defined by Equations 5.4 and 5.5
  if there exists a cycle $C = \langle p_1, p_2, \ldots, p_k \rangle$ in graph (V, E) **then**
    update selection S, by performing the replacements described by C
  end if
until there are no more cycles
return S

the objects $w_p \setminus\!\setminus \eta(S,p)$. This means that, in case there are different rules in R_i which compete for the same objects, then those objects has to be randomly assigned to the different rules. Moreover, maximality requires that the selected multiset of rules $\mathcal{R}'$ is such that no rule can be applied to the remaining objects $w_p \setminus\!\setminus \eta(S(p) \cup \mathcal{R}', (0,0))$. Therefore, *MaximiseApplications* initially generates an ord-max selection $S_1 \in \mathcal{S}^{\text{ordmax}}$ in which, for all positions, ordinary rules are applied in a maximal way, and no mutually exclusive rule is applied in any position.

Algorithm *GenerateSelectionExt*1, defined in Algorithm 9, behaves similarly to *GenerateSelection*1. It starts with the initial selection $S_1 \in \mathcal{S}^{\text{ordmax}}$ obtained from $MaximiseApplications(\perp)$, and then iteratively constructs greater selections $S_1 = S^{(1)} \prec S^{(2)} \prec \ldots \prec S^{(n)} = S_2$, by performing, in each iteration, a replacement of rules in a position. Precisely, in each iteration, the current multiset $S(p)$ selected in a position p is replaced with an improved multiset $\mathcal{R}' \in improved(S,p)$ which causes a change in the set of positions occupied by the mutually exclusive objects, and such that the updated selection $S' = S \setminus \{(p, S(p))\} \cup \{(p, \mathcal{R}')\} \in \mathcal{S}^{(a,b)}$. That is, selection S' has to satisfy the constraints on applicability (a) and mutual exclusivity (b) from Definition 4.2.4. The algorithm stops generating a filling selection $S_2 \in \mathcal{S}^{\text{filling}}$.

As regards Phase 2 of the algorithm, algorithm $ValidateSelectionExt$, defined in Algorithm 10, behaves similarly to $ValidateSelection$. Given the selection $S_2 \in \mathcal{S}^{\text{filling}}$ obtained from Phase 1, the algorithm constructs a graph of dependencies $G_{\text{dep}} = (V, E)$, in which nodes represent positions containing mutually exclusive objects, and each edge describes a possible replacement of rules. Then the algorithm searches a cycle in the graph, which corresponds to replacing the multisets of rules selected for a group of positions with greater multisets of rules, and then performs the replacement.

Precisely, given a selection S' obtained at a certain iteration of $ValidateSelectionExt$, each replacement corresponds to finding a set of positions Q in which it is possible to replace, in each position $q \in Q$, the multiset of rules selected $\mathcal{R} = S'(q)$ with a greater multiset $\mathcal{R}' \succ S'(q)$, and such that the replacements produce a selection $S'' \in \mathcal{S}^{(a,b)}$, i.e. S'' is applicable and does not create conflicts among mutually exclusive objects. Actually, by performing the replacement, the algorithm obtains a greater filling selection $S'' \in \mathcal{S}^{\text{filling}}$, i.e. $S'' \succ S'$. Algorithm $ValidateSelectionExt$ terminates when no more replacements can be performed, thus obtaining a *maximal* filling selection $S_3 \in \max(\mathcal{S}^{\text{filling}})$.

The dependency graph $G_{\text{dep}} = (V, E)$ used by algorithm $ValidateSelectionExt$ is formally defined in the following. Let $S_2 \in \mathcal{S}^{\text{filling}}$ be the selection obtained from Phase 1. Given a configuration W, the set of nodes V of the graph is defined as:

$$V = \{p \mid W' = apply(W, S_2) \wedge \sum_{x \in E} \mathbf{m}(w'_p, x) = 1\} \tag{5.4}$$

where function *apply* is as defined in Definition 4.2.5. As regards the edges, consider each non-empty position z in the configuration $W' = apply(W, S_2)$, and let $ext(R) = R_1, \ldots, R_k$ be the extended ordering of the rules R. Then the set of edges E of the dependency graph is the minimal set such that:

$$\begin{aligned} \mathcal{R}' \in improved(S_2, z) \;\wedge\; S \setminus \{(p, S_2(z))\} \cup \{(p, \mathcal{R}')\} \in \mathcal{S}^{(a)}_W \\ \implies \; (z + target(S_2(z)), z + target(\mathcal{R}')) \in E. \end{aligned} \tag{5.5}$$

In this way, there is an edge in the graph iff the current multiset $S_2(z)$, selected in some position z, can be replaced by a multiset $\mathcal{R}' \in improved(S_2, z)$, in which either (a) an

application of a mutually exclusive rule r is replaced by the application of another mutually exclusive rule r' with higher priority; or (b) no mutually exclusive rule is currently selected in $S_2(z)$ and a mutually exclusive rule becomes applied. In both cases, the replacement results in the position $p = z + target(S_2(z))$ being freed and $q = z + target(\mathcal{R}')$ being occupied, thus the edge (p, q) is present in the graph. Note that p and q may actually be the same position.

Once Phase 2 is completed, the algorithm has obtained a maximal filling selection $S_3 \in \max(\mathcal{S}^{\text{filling}})$, in fact mutually exclusive rules are applied in such a way that the constraints on applicability (a), mutual exclusivity (b), and priority and maximality (c) from Definition 4.2.4 are all satisfied. That is, it is not possible to apply, in any position, any other higher priority mutually exclusive rule without violating either the constraint (a) on applicability or the constraint (b) on mutual exclusivity.

Recall the definition of the priority relation on multiset of rules $\preceq$ (see Equation 4.1 of Definition 4.2.4). The maximal filling selection $S_3 \in \max(\mathcal{S}^{\text{filling}})$ obtained from Phase 2 is not necessarily a valid selection, since there can be applicable ordinary rules, in some positions, which are not selected. In particular, by the functioning of the algorithm, if a mutually exclusive rule $r \in \mathcal{R}^{\text{me}}$ is selected in a position, then no rule having lower priority than the selected rule r is applied. For this reason, Phase 3 of the algorithm is used to construct a valid selection S_4, by applying algorithm *MaximiseApplications* to S_3. Since S_3 is ordinary-rule maximal, the application of *MaximiseApplications* results in applying, in a maximal way, all the possible ordinary rules having lower priority than the selected mutually exclusive rule, if any.

5.2.3 Correctness of the extended algorithm

The following theorem proves the correctness of the extended algorithm.

Theorem 5.2.1. *The extended simulation algorithm SimulateStepExt, defined in Algorithm 7, for Spatial P system models with only ordinary rules from $\mathcal{R}^{\text{ord}}$ and mutually-exclusive rules from $\mathcal{R}^{\text{me}}$, behaves correctly with respect to the semantics of Spatial P systems, that is:*

$$\forall S.\quad W \xrightarrow{S} W' \quad \iff \quad W' \text{ can be generated by } SimulateStepExt(W)).$$

Proof. **Case** $\Longleftarrow$) As regards completeness, we have to prove that every selection generated by the algorithm is valid. We have already discussed that algorithm $MaximiseApplications(\bot)$ generates an ord-max selection $S_1 \in \mathcal{S}^{\text{ordmax}}$ in which no mutually exclusive rule is applied in any position. By the definition of ord-max selection, any greater selection $S' \succ S_1$ is such that there exists a position p and a mutually exclusive rule r, from a set $R_i = \{r\}$, which is applied in $S'(p)$ but not in $S_1(p)$, given that the multiplicity of all higher priority rules $r' > r$ is not changed. In fact, if the multiplicity of some higher priority rule $r' > r$ is changed, then we would obtain a multiset $\mathcal{R}$ which is not comparable with $S_1(p)$, that is neither $\mathcal{R} \succ S_1(p)$ nor $\mathcal{R} \prec S_1(p)$. This motivates the search, performed by algorithm *GenerateSelectionExt*1, of a position p in which the current multiset of rules $S(p)$ selected can be replaced by a greater multiset $\mathcal{R} \in improved(S, p)$, which causes a change in the set of positions containing mutually exclusive objects (constraint $target(S(p)) \neq target(\mathcal{R})$), and such that the updated selection does not violate either the property of applicability (a) or that of mutual exclusivity (b). Therefore, algorithm

*GenerateSelectionExt*1 generates a sequence $S_1 \prec S^{(1)} \prec \cdots \prec S^{(n)} = S_2$ of increasingly greater selections, until it stops with a filling selection $S_2 \in \mathcal{S}^{\text{filling}}$.

Since S_2 is filling, it is not possible to replace, in any position p, the currently selected rules $S_2(p)$ with an improved multiset $\mathcal{R}' \in \mathit{improved}(S_2, p)$ involving a different position for the mutually exclusive object, without violating either constraint (a) or (b). Therefore, the only way to find a greater selection $S' \succ S_2$ such that $S' \in \mathcal{S}^{(a,b)}$ is to perform a replacement corresponding to a cycle in the dependency graph. Moreover, since $S_2 \in \mathcal{S}^{\text{filling}}$, also $S' \in \mathcal{S}^{\text{filling}}$. The algorithm iterates this procedure until no more cycles are present, therefore it stops as soon as a maximal filling selection $S_3 \in \max(\mathcal{S}^{\text{filling}})$ is obtained.

Let us consider a maximal filling selection $S_3 \in \max(\mathcal{S}^{\text{filling}})$, and a position p. Suppose that there is a mutually exclusive rule $r \in \mathcal{R}^{\text{me}}$ selected for the position, that is $r \in S(p)$. Any lower priority mutually exclusive rule $r' < r$ does not need to be considered, since *at most* one of such kind of rules can be selected for application in a position. Therefore, it is sufficient to consider only the ordinary rules having lower priority. By the definition of the priority relation on multisets of rules $\preceq$, in order to obtain a greater multiset of rules $\mathcal{R}' \succ S(p)$ for the position p, it is sufficient to apply lower priority ordinary rules to the objects still available in the position, in a maximal way.

This is performed by algorithm $\mathit{MaximiseApplications}(S_3)$, which tries to apply in a position also the other rules having higher priority than the currently selected mutually exclusive rule. However, since those rules are already applied in a maximal way, none of them can be applied again. Thus the algorithm applies, for each position p, only lower priority rules in a maximal way with respect to the extended ordering $\leq$. By the definition of filling selection, the selection $S_4 = \mathit{MaximiseApplications}(S_3)$ is a valid selection, i.e. $S_4 \in \mathcal{S}^{\text{valid}}$.

Case $\Longrightarrow$) In the following, we denote by $\longrightarrow_{1\text{e}}$ the relation describing an iteration of algorithm *GenerateSelectionExt*1, and by $\Longrightarrow_{1\text{e}}$ its transitive closure. That is $S_1 \longrightarrow_{1\text{e}} S_2$ iff, starting from selection S_1, S_2 is reachable after one iteration of algorithm *GenerateSelectionExt*1. Moreover, we denote by $\longrightarrow_{2\text{e}}$ the relation describing an iteration of the *ValidateSelectionExt* algorithm, and by $\Longrightarrow_{2\text{e}}$ the transitive closure of $\longrightarrow_{2\text{e}}$. Finally, we denote by $\longrightarrow_{\text{m}}$ the relation describing the procedure implemented by *MaximiseApplications* algorithm, that is $S_1 \longrightarrow_{\text{m}} S_2 \iff S_2 \in \mathit{MaximiseApplications}(S_1)$.

In order to prove the thesis, it is sufficient to prove that, given the extended ordering $<$, any valid selection can be generated by the *SimulateStepExt* algorithm.

We prove the following statements:

1. $\forall S \in \min(\mathcal{S}^{\text{ordmax}}).\ \bot \longrightarrow_{\text{m}} S$, that is any minimal ord-max selection can be generated by $\mathit{MaximiseApplications}(\bot)$;

2. $\forall S \in \min(\mathcal{S}^{\text{filling}}).\ \exists \hat{S} \in \min(\mathcal{S}^{\text{ordmax}}).\ \hat{S} \Longrightarrow_{1\text{e}} S$, that is any minimal filling selection can be obtained from *GenerateSelectionExt*1;

3. $\forall S \in \mathcal{S}^{\text{valid}}.\ \exists S' \in \min(\mathcal{S}^{\text{filling}}).\ \exists S'' \in \mathcal{S}^{\text{filling}}.\ S' \Longrightarrow_{2\text{e}} S'' \longrightarrow_{\text{m}} S$, that is for any valid selection S there is a minimal filling selection S', and a filling selection S'', such that algorithm *ValidateSelection* can obtain S'' starting from S', and *MaximiseApplications* can obtain S from S''.

As regards part 1, note that every minimal selection $S \in \min(\mathcal{S}^{\text{ordmax}})$ is such that no mutually exclusive rule is selected in any position. In fact, suppose $S \in \min(\mathcal{S}^{\text{ordmax}})$ and that a mutually exclusive rule $r \in \mathcal{R}^{\text{me}}$ is selected for a position p, i.e. $r \in S(p)$. Then, a smaller multiset $\mathcal{R} \prec S(p)$ can be obtained by cancelling the application of rule r, and applying all the lower priority rules $r' < r$ in a maximal way. Since it is possible to select multiset $\mathcal{R}$ in such a way that $S' = S \setminus \{(p, S(p))\} \cup \{(p, \mathcal{R})\} \in \mathcal{S}^{(a,b)}$, this contradicts the assumption that S would be minimal. Therefore it is sufficient that the algorithm is able to generate any ord-max selection such that no mutually exclusive rule is selected in any position. Supposing that the algorithm, while iterating through $ext(R) = R_1, \ldots, R_k$, can generate *any* maximal multiset of applicable rules $\mathcal{R}'$ from the current R_i, then this implies that any $S \in \min(\mathcal{S}^{\text{ordmax}})$ can be generated.

As regards part 2, it is sufficient to prove that, given a minimal filling selection $S \in \min(\mathcal{S}^{\text{filling}})$, for all selections $S' \prec S$, *GenerateSelectionExt*1 algorithm can perform an iteration from S' to some other S'' which is still less than or equal to selection S. That is, given a minimal filling selection $S \in \min(\mathcal{S}^{\text{filling}})$:

$$\forall S' \in \mathcal{S}^{\text{ordmax}}.\ S' \prec S \implies \exists S'' \in \mathcal{S}^{\text{ordmax}}.\ S' \longrightarrow_{\text{1e}} S'' \ \wedge\ S' \prec S'' \preceq S$$

Given a configuration W, and a selection $S \in \mathcal{S}^{\text{ordmax}}_W$, we denote by $occ(S)$ the set of positions containing a mutually exclusive object, with respect to the configuration resulting from the application of S to W. That is, given $W' = apply(W, S)$, then $occ(S) = \{p \mid \sum_{x \in E} \mathbf{m}(w'_p, x) > 0\}$. Moreover, given the selections $S \in \min(\mathcal{S}^{\text{filling}})$, and $S' \in \mathcal{S}^{\text{ordmax}}$ such that $S' \preceq S$, note that, for each position p, the multiplicity of each rule with higher priority than the mutually exclusive one selected in $S(p)$, if any, is the same in $S'(p)$ and in $S(p)$, otherwise $S'(p) = S(p)$.

Consider a selection $S \in \min(\mathcal{S}^{\text{filling}})$, and a selection $S' \in \mathcal{S}^{\text{ordmax}}$ such that $S' \prec S$. On one hand, assume that $occ(S) = occ(S')$. Then there must exist a position p in which the currently selected multiset $S'(p)$, can be replaced by the algorithm with a greater multiset $\mathcal{R}' \in improved(S', p)$, $\mathcal{R}' \succ S'(p)$, such that $target(\mathcal{R}') \notin occ(S)$ and $\mathcal{R}' \prec S(p)$. In fact, if such a position would not exist, then, since $S \in \mathcal{S}^{\text{filling}}$ also $S' \in \mathcal{S}^{\text{filling}}$ and therefore S would not be minimal.

On the other hand, assume that $occ(S) \neq occ(S')$. Since $|occ(S)| = |occ(S')|$, there exists a position $q \in occ(S) \setminus occ(S')$. Let p be the position such that $target(S(p)) = q$. Then it is sufficient that the algorithm: (i) selects position p to search for a greater multiset $\mathcal{R}$ to replace the currently selected multiset $S'(p)$, and that (ii) chooses the least multiset $\mathcal{R}$ among the improved ones for which $target(S'(p)) \neq target(\mathcal{R})$, that is $\mathcal{R} \in \min(\{\mathcal{R}' \in improved(S', p) \mid target(S'(p)) \neq target(\mathcal{R}') \wedge \mathcal{S} \setminus \{(p, S'(p))\} \cup \{(p, \mathcal{R}')\} \in \mathcal{S}^{(a,b)}\})$. If no multiset $\mathcal{R}$ such that $S'(p) \prec \mathcal{R} \prec S(p)$ can replace the currently selected multiset $S'(p)$, then the algorithm selects $S(p)$. Note that, in such a case, it cannot happen that $target(S(p)) = target(S'(p))$, since that would imply that S is not minimal, contradicting the hypothesis.

As regards part 3, consider a valid selection $S \in \mathcal{S}^{\text{valid}}$. Let S'' be the selection obtained from S by cancelling, in each position, the application of any lower priority rule than the selected mutually exclusive rule, if any. Since S is valid, then $S'' \in \mathcal{S}^{\text{filling}} (\subseteq \mathcal{S}^{\text{ordmax}})$. Therefore, algorithm *MaximiseApplications* can generate S from S'', i.e. $S'' \longrightarrow_{\text{m}} S$.

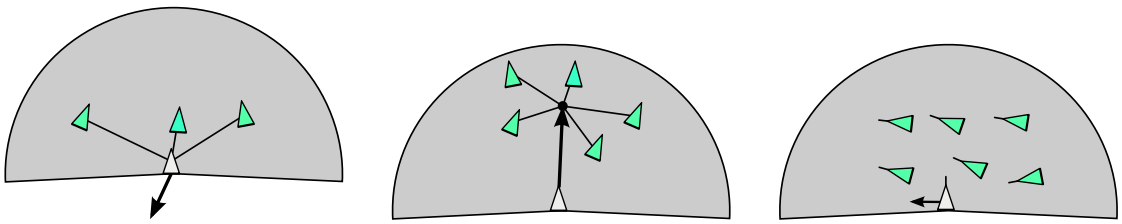
Given $S'' \in \mathcal{S}^{\text{filling}}$, we can apply a procedure corresponding to the converse of algorithm *ValidateSelectionExt*, which constructs a sequence of filling selections $S'' = S_1 \succ$

$S_2 \succ \cdots \succ S_k = S'$, where each S_{i+1} is obtained from S_i by replacing the multisets of rules selected for some positions with lesser multisets, ensuring that $occ(S_{i+1}) = occ(S'')$. Therefore, since $S'' \in \mathcal{S}^{\text{filling}}$, also $\forall i.\ S_i \in \mathcal{S}^{\text{filling}}$. This procedure terminates as soon as a minimal filling selection $S' = S_k \in \min(\mathcal{S}^{\text{filling}})$ is obtained, which is consequently such that $S' \Longrightarrow_{2e} S''$. □

5.3 A model of the schooling behaviour of fish

In this section we present a small example model, which uses Spatial P systems to describe the schooling behaviour of fish. Some kinds of fish, such as herring, swim together forming schools. This behaviour has different advantages, for example is used against predators. One interesting problem with this kind of systems, is that the single fish are able to organize themselves into schools, following local attraction rules. That is, this behaviour is not driven by some external entity which controls the behaviour of the single fish, but the common behaviour emerges as the result of local interactions between fish. Besides fish, similar behaviours also occur for other species, such as the flocking behaviour of birds, or the motion of herds of animals.

A well-known algorithm for simulating this kind of aggregate motion is the *Boids* model, proposed by Reynolds in 1987 [76]. In this model, the movement of a fish is driven by the sum of different forces, calculated with respect to the distance and direction of movement of nearby fish. Individuals can "see" only a small space around themselves, in order to determine the relative distance and behaviour of nearby individuals.



(a) Separation (b) Cohesion (c) Alignment

Figure 5.1: The different forces, in the Boids model, determining the resulting direction of an individual.

Figure 5.1 shows the forces affecting the behaviour of an individual. On one hand, there is a repulsive *separation* force that causes the fish to move away from nearby fish, whose magnitude is inversely proportional to their relative distances. On the other hand, there is an attractive *cohesion* force which instead tends to keep the school together, by driving the fish towards the direction in which there are most of them. Finally, there is an *alignment* force, which tends to align the direction of the herring to the most common direction among nearby individuals. The sum of these three forces is used to determine the resulting direction of each fish. Using these rules, a group of randomly positioned fish in a space is able to organize themselves into schools, in which individuals move in a coordinated manner.

We present in the following a Spatial P system model of the schooling behaviour of fish, which resembles the Boids model.

Spatial P systems model

In our model, a direction is associated with each herring, describing the direction where the herring is headed. There are 8 possible directions, corresponding to the directions shown in Figure 5.2. In the Spatial P system model, we use a different mutually exclusive symbol to denote each possible direction of a herring. In particular, the set of mutually exclusive objects is $E = \{1, 2, 3, 4, 5, 6, 7, 8, e\}$, where each symbol among $1, \ldots, 8$ corresponds to a direction as shown in Figure 5.2. There is also a special mutually exclusive object e used to delimit the simulation space. There are no ordinary objects: $V = \emptyset$.

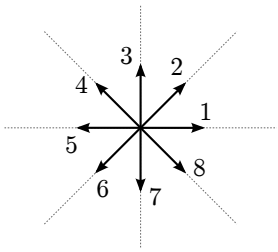


Figure 5.2: The possible directions of a herring.

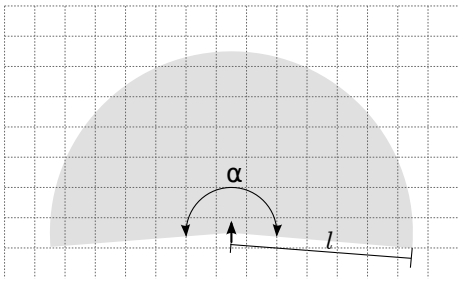


Figure 5.3: The visibility parameters: α, the visibility angle; l the visibility distance.

The model depends on a number of parameters, as detailed in the following:

- **Speed**: a real value describing the speed of the herring;
- **VisibilityDistance**: the distance at which the herring can see, as depicted in Figure 5.3;
- **VisibilityAngle**: describes the *Field of View* (FOV) of the herring, that is the angular extent that the herring can see ahead of itself (Figure 5.3);
- **SeparationCoefficient**: a real value greater than 1, used to scale down the repulsive effect of nearby individuals with respect to their relative distance; a greater value means that the repulsive effect decreases faster according to their distance;
- **CohesionCoefficient**: how much the cohesion force influences the resulting direction;
- **AlignmentCoefficient**: how much the alignment force influences the resulting direction.

In our implementation, given a herring in a position p, the separation force is computed as the sum of the vectors $h - p$ from the each nearby herring to the current herring, multiplied by a value $\|h - p\|^{-k}$, where k is the SeparationCoefficient. Therefore the effect of the separation force decreases with the increasing of the distance between the herrings. The cohesion force corresponds to the vector from p to the "mean position" among the positions of nearby herrings, multiplied by the CohesionCoefficient. The mean position is actually obtained by summing up all the vectors corresponding to the position of nearby herrings, and dividing by the number of nearby herrings. Finally, the alignment force is

obtained as the sum of each unit vector corresponding to the direction of a nearby herring, multiplied by the AlignmentCoefficient.

The model is described by using restricted rules of the form:

$$(d)_{(0,0)}[(\pi_1)_{q_1} \dots (\pi_\gamma)_{q_\gamma}] \to (d')_q$$

where $d, d' \in \{1, \dots, 8\}$, and $q \in \mathbb{Z}^2$ is the resulting position of the herring. Promoters are used to see the space nearby the herring. The resulting direction d', depends on the current direction d and on the position and direction of nearby herrings, represented by promoters. Given the promoters $\overline{\pi} = (\pi_1)_{q_1} \dots (\pi_\gamma)_{q_\gamma}$, there are different rules, with decreasing priority and the same left hand side, that describe the possible movements of the herring. Priority is used to model the movement faithfully in case there are conflicts.

Figure 5.4 shows a herring, its resulting direction, and the possible resulting positions $q_1, q_2, \dots, q_6$, in decreasing order of priority. In particular, the highest priority rule tries to put an object in the most distant position along the chosen direction, i.e. q_1 in Figure 5.4. That is, the preferred position is the one corresponding to a movement along the chosen direction, with the specified speed. If, as the result of the movement of other herrings, position q_1 happens to be occupied, then there is a rule with less priority which tries to put an object in the immediately preceding position q_2, and so on until the least priority rule which tries to put the herring position q_6, adjacent to the current position. Therefore, given some promoters $\overline{\pi}$ and an object d, there are a number of rules $r_1 > r_2 > \dots > r_6$, which try to put the object d' in one of the resulting positions $q_1, q_2, \dots, q_6$, respectively.

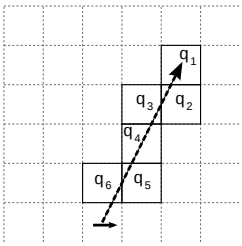


Figure 5.4: An example of possible moves of a herring.

The simulation space is delimited by mutually exclusive objects e, which are not modified by the rules. Nevertheless, objects e can appear among the promoters of the rules, in order to define the behaviour of herrings near the simulation bounds. In this way, it is possible to model a behaviour which assumes that herrings bounce back when trying to move in a position which is beyond the bounds.

Recall that promoters are checked for their presence, and not their absence. Therefore, consider the case in which there are two rules r_1, r_2 with the same symbol d as reactant, but with different products d', d'', and two collections of promoters $\overline{\pi}_1$, $\overline{\pi}_2$ respectively, such that all symbols in $\overline{\pi}_2$ also appear in $\overline{\pi}_1$, in the same relative positions. In such a case, if the former rule r_1 is applicable, then the latter rule r_2 is also applicable. In order to deal correctly with this case, it is necessary to ensure that rule r_1 has higher priority than r_2, in such a way that rule r_2 is applied only if some promoters among $\overline{\pi}_1$ are not present. That is, whenever there are two rules in which one has "at least" all the promoters of another rule, then it is necessary that the rule with the greater collection of promoters has higher priority than the other rule.

The Spatial P systems model closely resembles the Boids model, where the evolution rules are determined from the different forces driving each fish, and the actual parameters used. In case there are no nearby fish, that is the visibility space for a fish is empty, we have implemented in the model a random change of direction for the fish. In such a case, the fish changes direction, by rotating of 45° either clockwise or counter-clockwise. Since the current definition of Spatial P systems does not provide probabilities among rules, it is sufficient to provide different rules with the alternative outcome. That is, there are a set of rules $r_1 > r_2 > \cdots > r_n$, which move the object along a direction, and another set of rules $r'_1 > r'_2 > \cdots > r'_n$, which move the object along the other direction. In order to simulate a behaviour in which the fish first chooses a direction, and then tries to move along that direction, discarding the other direction, it is necessary to include a special "no-move" rule for each sequence of rules. Such a special rule describes the lack of movement for the object, and just capture the change of direction of the fish. Let denote by $\hat{r}$ and $\hat{r}'$ such special rules for the two sequences of rules, and whose priorities are such that $r_1 > r_2 > \cdots > r_n > \hat{r}$ and $r'_1 > r'_2 > \cdots > \hat{r}'$, respectively.

We have implemented a simulator for the Boids model described previously following the semantics of Spatial P systems. In particular, since the model is composed only of mutually-exclusive objects, we have implemented the algorithm for restricted models presented in Section 5.1. The model consists of a square grid of size 500×500 cells, which is initially randomly filled with 10000 fish in a square of 350×350. Figures 5.5 and 5.6 depict the simulation state at different simulation steps, where each non-empty position in represented by a black cell. The directions of fish are not shown.

The simulation has been performed using the following parameter values: Speed = 4, VisibilityDistance = 16, VisibilityAngle = 1.05π, SeparationCoefficient = 1.3, Cohesion-Coefficient = 1, AlignmentCoefficient = 1.

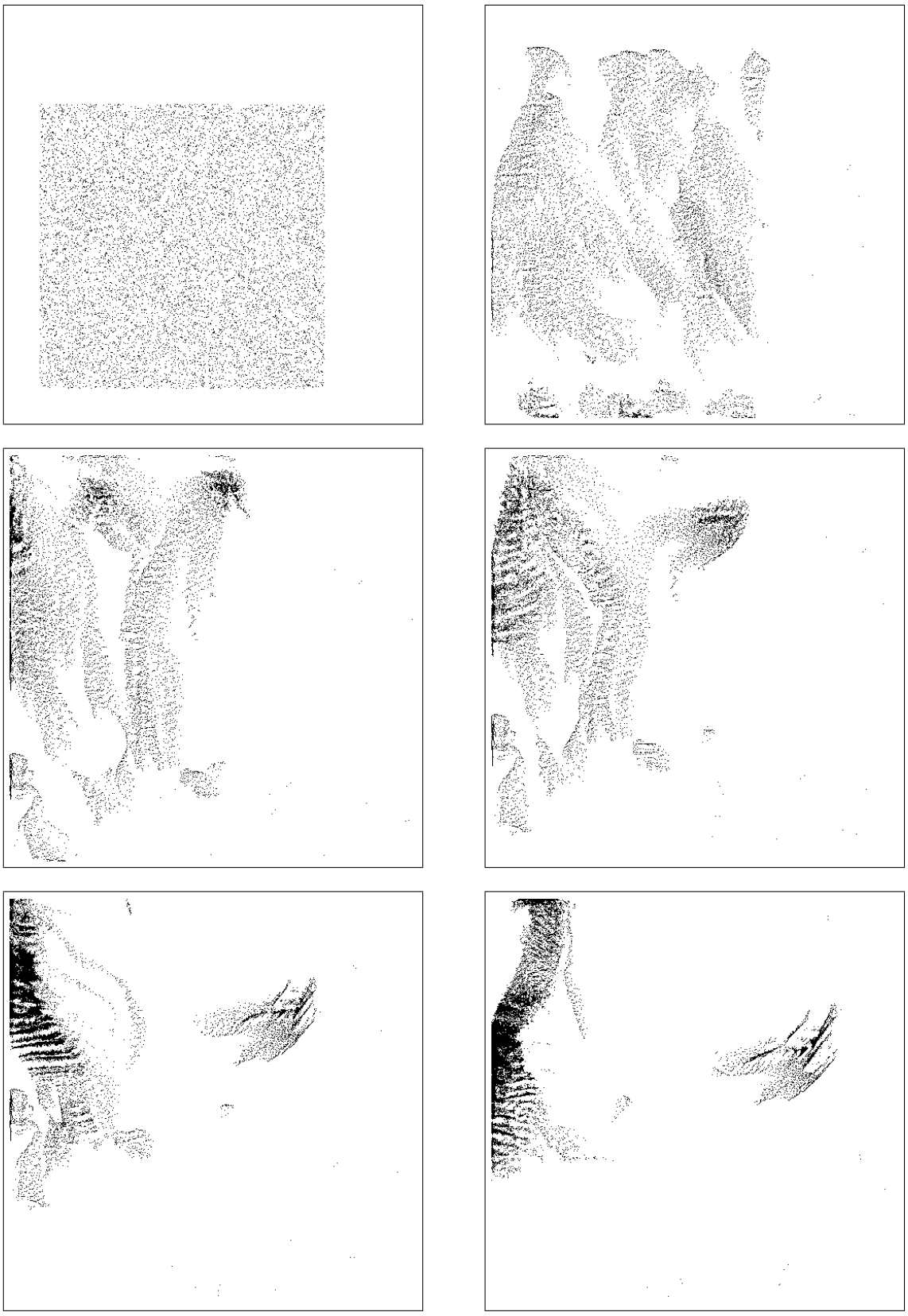


Figure 5.5: Simulation steps 0, 20, 51, 63, 84, 100.

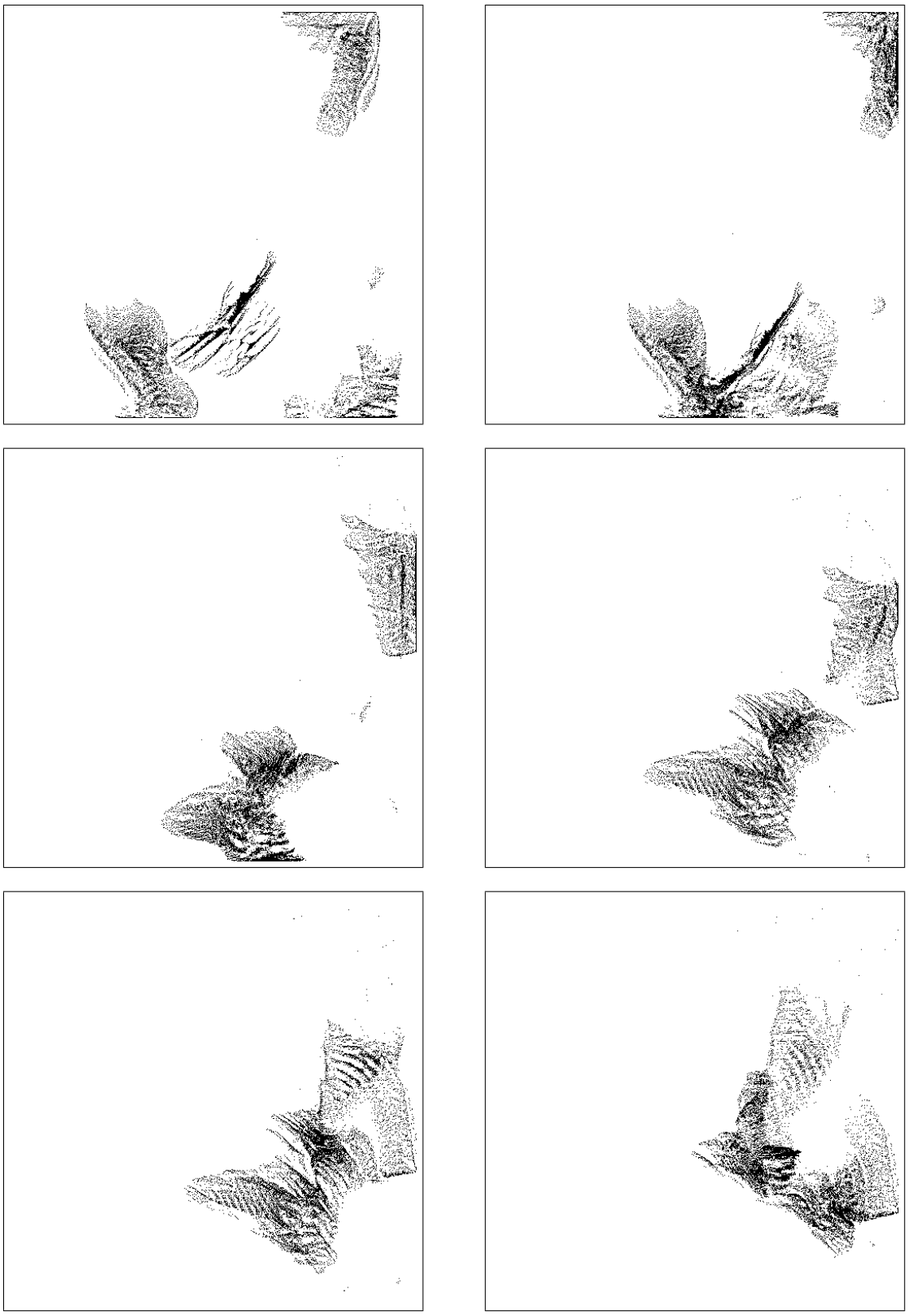


Figure 5.6: Simulation steps 190, 209, 235, 251, 261, 275.

Chapter 6

Spatial Calculus of Looping Sequences

In this chapter we present the *Spatial Calculus of Looping Sequences* (Spatial CLS), which extends CLS by allowing spatial information to be associated with CLS structures when this information is relevant for determining the system behaviour. In Spatial CLS, all structures are embedded in an Euclidean space, which may be either 2D or 3D according to the needs of the modeller. Structures are associated with a precise position in space, and their movement can be precisely described. Moreover, Spatial CLS inherits from CLS the ability to explicitly model membranes (i.e. compartments), by means of the *looping sequence* structure provided by CLS. The interaction among objects of the system, namely the applicability of rewrite rules, can be constrained to the position of the elements involved. For instance, it is possible to constrain a protein to enter cell membrane only if it is within a certain distance from the membrane itself. Both deterministic and stochastic motions of elements can be described.

An important feature of the Spatial CLS is the description of the space occupied by the elements, with the constraint that there cannot be any conflict between the space occupied by different elements. In particular, an "exclusion space" can be associated with elements, which is either circular or spherical, according to the dimension of the considered space (2D/3D). The semantics ensures that no space conflicts between elements arise during the evolution of the system. The calculus allows the use of different strategies to rearrange the elements, in case of a space conflict. The semantics prescribes that, if no valid arrangement can be found, then the event that would cause the conflict cannot occur. Finally, rewrite rules are endowed with kinetic parameters describing their stochastic application rate.

The aim of Spatial CLS is to enable a more accurate description of those biological processes whose behaviour depends on the exact position of the elements. This high level of accuracy is especially useful for cell biology, where there can be a high degree of spatial organization and molecular species may be distributed not uniformly in the space [2]. Such descriptions can then be used to simulate the systems, so as to obtain a faithful representation of their evolution. Handling spatial information in a simulator may have a high computational cost. However, Spatial CLS allows specifying spatial information only for those elements for which such information is relevant, thus enabling the modeller to mix descriptions at different levels of abstraction. As example applications of the calculus we present a model of cell proliferation, as happens during the development of a biological

tissue, and a model of the *quorum sensing* process in *Pseudomonas aeruginosa*. In the case of cell proliferation, we formalize an algorithm for the rearrangement of the objects in the system, which tries to resolve space conflicts by simulating the movement of elements as if they push each other when their exclusion spaces overlap. We show the results of simulation of the two models.

We start by recalling the definition of the variant of the Calculus of Looping Sequences used as the basis for the extension into the Spatial CLS. Then, we present the formal syntax and semantics of Spatial CLS. We provide a definition of the Arrange algorithm, used by the semantics to perform a rearrangement of elements in case of space conflicts, and prove the termination of the algorithm. Finally, we show some examples of applying the features of the calculus to biological modelling, and present the complete models of cell proliferation and the quorum sensing process, along with the results of their simulations.

6.1 The Calculus of Looping Sequences

We recall the variant of the Calculus of Looping Sequences (CLS) called CLS+ [57], which forms the basis of our spatial extension into Spatial CLS. In the definition of CLS+, we assume an infinite alphabet of symbols $\mathcal{E}$ (ranged over by $a, b, c, \ldots$) for constructing sequences. A CLS term provides a static view of the system, by describing objects and their containment relations. Formally, each object is modelled as a *sequence* of symbols, and two kinds of sequences are provided: *simple* sequences and *looping* sequences. Simple sequences are meant to model simple biological entities, such as proteins and DNA strands, while looping sequences allow modelling membranes, thus providing a containment relation.

A CLS model is composed of a term, which describes the initial state of the biological system, and a collection of *rewrite rules*, describing the possible interactions among the elements and how they evolve. The semantics is given as a transition system describing the possible evolutions of the system, where *states* are CLS terms, and each *transition* corresponds to the application of a rewrite rule to the term.

Formally, the syntax of CLS+ terms is defined as follows.

Definition 6.1.1 (Terms)**.** *Terms* T, *branes* B and *sequences* S of CLS+ are given by the following grammar:

$$\begin{array}{lcl} T & ::= & S \mid \left(B\right)^{L} \rfloor T \mid T \mid T \\ B & ::= & S \mid B \mid B \\ S & ::= & \epsilon \mid a \mid S \cdot S \end{array}$$

The sets of all terms, branes and sequences are denoted by $\mathcal{T}, \mathcal{B}$ and $\mathcal{S}$, respectively. Note that $\mathcal{B} \subset \mathcal{T}$.

The sequencing operator $\_ \cdot \_$ can be used to build sequences of symbols in $\mathcal{E}$ and ϵ denotes the empty sequence, that is a concatenation of zero symbols. For constructing terms, we have a looping operator $\left(\_\right)^{L}$, a containment operator $\_ \rfloor \_$ and a parallel composition operator $\_ \mid \_$. A term may contain simple sequences S and looping sequences $\left(B\right)^{L} \rfloor T$. The containment operator $\rfloor$ allows the representation of compartments; in fact, a looping sequence $\left(B\right)^{L} \rfloor T$ usually models a membrane with a surface modelled by B (a parallel composition of sequences) and a content modelled by T. We distinguish between branes

B and terms T in order to prevent the possibility of having looping sequences appearing on the surface of another membrane. However, according to syntax definition, note that each brane is also a valid term, thus $\mathcal{B} \subset \mathcal{T}$.

Since, in CLS+, looping $(\_)^L$ and containment $\_ \rfloor \_$ are always applied together, we consider them as a single binary operator which applies to a brane and to a term. Brackets can be used to indicate the order of application of the operators, and $(\_)^L \rfloor \_$ is assumed to have precedence over $\_ \mid \_$.

Note that a looping term abstracts a real membrane, and the elements appearing on the surface of the membrane itself are only the ones which are crucial to describe the system. For this reason, elements which are not involved in the modelled process, are not usually represented.

The structural congruence relation on terms identifies syntactically different terms that conceptually represent the same structure.

Definition 6.1.2 (Structural congruence)**.** The *structural congruence* relations on sequences $\equiv_S$ and terms $\equiv_T$ are the least congruences satisfying the following rules:

$$\begin{gathered} S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3 \\ S \cdot \epsilon \equiv_S \epsilon \cdot S \equiv_S S \\ S_1 \equiv_S S_2 \Rightarrow S_1 \equiv_T S_2 \end{gathered}$$

$$\begin{gathered} T \mid \epsilon \equiv_T T \\ T_1 \mid (T_2 \mid T_3) \equiv_T (T_1 \mid T_2) \mid T_3 \\ T_1 \mid T_2 \equiv_T T_2 \mid T_1 \end{gathered}$$

The structural congruence states the associativity of both the sequencing and the parallel operator, the commutativity of the latter, and the neutral role of ϵ.

The evolution of a system is described by a set of rewrite rules, modelling reactions among system elements. A rule is composed of a pair of *patterns* (terms with variables) with the intuitive meaning that, if the first pattern occurs in a portion of the system, then that portion can be modified according to the second pattern.

We assume the following infinite and pairwise disjoint sets of variables: $\mathcal{X}$ for element variables $x, y, \ldots$; SV for sequence variables $\widetilde{x}, \widetilde{y}, \ldots$; BV for brane variables $\overline{X}, \overline{Y}, \ldots$; and TV for term variables $X, Y, \ldots$. We denote the set of all variables by $\mathcal{V}$. We distinguish among different kinds of pattern, as in the following definitions.

Definition 6.1.3 (Brane and sequence patterns)**.** *Brane patterns* BP and *sequence patterns* SP are given by the following grammar:

$$\begin{array}{lcl} BP & ::= & SP \;\mid\; BP \mid BP \\ SP & ::= & \epsilon \;\mid\; a \;\mid\; SP \cdot SP \;\mid\; \widetilde{x} \;\mid\; x \end{array}$$

We denote the sets of all brane and sequence patterns with $\mathcal{BP}$ and $\mathcal{SP}$, respectively.

Definition 6.1.4 (Term patterns)**.** *Left patterns* P_L and *right patterns* P_R are given by

the following grammar:

$$\begin{array}{rcl} P_L & ::= & SP \mid \left(BP_{LX}\right)^L \rfloor P_{LX} \mid P_L \mid P_L \\ BP_{LX} & ::= & BP \mid BP \mid \overline{X} \\ P_{LX} & ::= & P_L \mid P_L \mid X \\ P_R & ::= & SP \mid \left(BP_{RX}\right)^L \rfloor P_R \mid P_R \mid P_R \mid X \mid \overline{X} \\ BP_{RX} & ::= & BP \mid BP_{RX} \mid \overline{X} \end{array}$$

We denote the sets of all left and right patterns with $\mathcal{P}_L$ and $\mathcal{P}_R$, respectively. We assume brane patterns to be a subset of left and right patterns, i.e. $\mathcal{BP} \subset \mathcal{P}_L (\subset \mathcal{P}_R)$.

The set of all variables appearing in a pattern P is denoted by $\mathrm{Var}(P)$. We also assume the structural congruence relation to be extended to patterns.

A CLS+ term evolves by applying rewrite rules to it. A *rewrite rule* is a pair of patterns (P_L, P_R), usually written as $P_L \mapsto P_R$, such that $P_L \not\equiv \epsilon$ and $\mathrm{Var}(P_R) \subseteq \mathrm{Var}(P_L)$. If P_L and P_R are indeed brane patterns, then the rule is called *brane (rewrite) rule*.

Given a pattern, we may obtain a term by applying an *instantiation function* $\sigma : \mathcal{V} \to \mathcal{T}$, describing the bindings between variables and their values. This application, denoted by $P\sigma$, replaces each occurrence of $v \in \mathrm{Var}(P)$ in P with $\sigma(v)$. For example, we can instantiate the pattern $P = \left(a{\cdot}\widetilde{x} \mid \overline{X}\right)^L \rfloor (c \mid Y)$ with instantiation $\sigma = \{(\widetilde{x}, b{\cdot}b), (\overline{X}, a{\cdot}b{\cdot}b \mid d \cdot b \cdot b), (Y, c \mid d)\}$ obtaining the term $P\sigma = \left(a \cdot b \cdot b \mid a \cdot b \cdot b \mid d \cdot b \cdot b\right)^L \rfloor (c \mid c \mid d)$. An instantiation function σ must respect the type of variables, namely for all $x \in \mathcal{X}$, $\widetilde{x} \in SV$, $\overline{X} \in BV$, and $X \in TV$ we have $\sigma(x) \in \mathcal{E}$, $\sigma(\widetilde{x}) \in \mathcal{S}$, $\sigma(\overline{X}) \in \mathcal{B}$, and $\sigma(X) \in \mathcal{T}$, respectively. The set of all instantiations is denoted by Σ.

A rewrite rule $P_L \mapsto P_R$ states that a term $P_L\sigma$, obtained by instantiating variables in P_L by some instantiation function σ, can be transformed into the term $P_R\sigma$. Thus, a term T may evolve to another term T' by applying a rewrite rule to a subterm of T.

The use of different kinds of pattern allows us to constrain the occurrences of variables inside a term, which allows a simpler definition of the semantics. First of all, brane and term variables may occur only on branes and inside looping sequences, respectively. Then, as regards left patterns, term variables are not allowed at top–level, and at most one brane or term variable is allowed in each compartment. We do not allow term variables on branes: this ensures that the application of a rewrite rule never yields an invalid term, i.e. a term with looping sequences on branes. For the same reason, we identify brane rewrite rules $BP_1 \mapsto BP_2$ as the only kind of rules that can be applied to branes.

As explained in [4], the constraints introduced do not restrict the expressive power of the calculus for modelling biological systems. In fact, they rule out cases which are not biologically reasonable, such as reactions involving an uncertain number of reactants, or reactions between two arbitrary portions of the content of a membrane.

The semantics of the calculus is given as a transition system, in which states correspond to terms, and each transition $\to$ represents the application of a rewrite rule. The definition uses the auxiliary transition relation $\to_{\mathcal{B}}$, that describes the evolution of branes (ensuring that only brane rewrite rules can be applied to their elements).

Definition 6.1.5 (Semantics)**.** Given a set of rewrite rules $\mathcal{R}$, let $\mathcal{R}_{\mathcal{B}}$ denote its subset of all and only brane rules ($\mathcal{R}_{\mathcal{B}} \subseteq \mathcal{R}$). The *semantics* of CLS+ is the least transition relation

$\to$ on terms closed under $\equiv_T$ and satisfying the following inference rules:

$$\frac{P_1 \mapsto P_2 \in \mathcal{R} \qquad P_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma}{P_1\sigma \to P_2\sigma} \qquad \frac{T_1 \to T_1'}{T_1 \mid T_2 \to T_1' \mid T_2}$$

$$\frac{BP_1 \mapsto BP_2 \in \mathcal{R}_{\mathcal{B}} \qquad BP_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma}{BP_1\sigma \to_{\mathcal{B}} BP_2\sigma} \qquad \frac{B_1 \to_{\mathcal{B}} B_1'}{B_1 \mid B_2 \to_{\mathcal{B}} B_1' \mid B_2}$$

$$\frac{T_1 \to T_2}{\left(B\right)^L \rfloor T_1 \to \left(B\right)^L \rfloor T_2} \qquad \frac{B \to_{\mathcal{B}} B'}{\left(B\right)^L \rfloor T \to \left(B'\right)^L \rfloor T}$$

As an example, let $T = \left(a \cdot b \cdot b \mid d \cdot b\right)^L \rfloor (c \mid d)$ and let $\left(a \cdot \widetilde{x} \mid \overline{X}\right)^L \rfloor (c \mid Y) \mapsto \left(d \cdot \widetilde{x} \mid \overline{X}\right)^L \rfloor Y$ be a rewrite rule modelling the formation of a complex on a membrane by the interaction of an element $a \cdot \widetilde{x}$ on the membrane with an element c inside the membrane. By applying the rule to T, we obtain $\left(d \cdot b \cdot b \mid d \cdot b\right)^L \rfloor d$, where $d \cdot b \cdot b$ is the resulting complex.

6.2 The Spatial CLS

The Spatial CLS extends the Calculus of Looping Sequences by enriching sequences and membranes with spatial information. In particular, elements are embedded in a two-dimensional or three-dimensional space, which is chosen by the modeller depending on the characteristics of the system to be modelled. Each simple sequence and looping sequence (collectively called *objects* in the following), which are used to model entities of the biological system, can have a precise position associated with it. Besides having a position, the movement of each object, as time passes, can be precisely described by associating a *movement function* with objects. This movement function can be used to model different kinds of motion, such as *Brownian motion* (see Section 6.5).

Rewrite rules modelling reactions are extended to allow taking into account the position of the interacting elements. Constraints on the positions of elements involved in a reaction can be precisely described. Borrowing from *Stochastic CLS* [4], rewrite rules are endowed with a stochastic reaction rate parameter, describing the propensity of application of the rule.

For maximum flexibility, it is possible to avoid keeping track of the position of some elements. Objects of the calculus are distinguished between *positional elements* and *non-positional elements*. Positional elements are as already described, while non-positional elements have neither an associated position nor a movement function. Non-positional elements are assumed to be homogeneously distributed in the compartment, and their behaviour is analogous to that of Stochastic CLS elements. In this way, the modeller can choose the most appropriate level of abstraction for describing (different parts of) a biological system.

Finally, Spatial CLS allows describing the space occupied by the elements. For the sake of simplicity, the space occupied by an element is described by a real-valued *radius* parameter, modelling an "exclusion space" around the position of the element. According to the dimensions (2D or 3D) used in the model, this exclusion space is represented either as a circle or as a sphere, centered in the position of the element. We provide a notion of

well-formedness of terms, which captures the constraint that different objects are not too close one another, ensuring that their exclusion spaces are always kept disjoint.

The semantics also allows for a rearrangement of the elements in case of a space conflict. The modeller can define the most appropriate algorithm for arranging the objects, according to his/her needs. In Section 6.4 we provide an example of definition of the arrange algorithm which tries to find a conflict-free configuration by simulating the movement of objects as if they push each other when they are too close. This algorithm is used in the model of cell proliferation.

In the following, we present the syntax of the calculus for terms and rewrite rules, in their full detail. The formal semantics of Spatial CLS is then presented in the subsequent Section 6.3.

6.2.1 Formal definition

The definition of Spatial CLS is based on CLS+, recalled in Section 6.1. The syntax of terms is an extension of that of CLS+, where objects (simple and looping sequences) are enriched with spatial information. We assume an alphabet of symbols $\mathcal{E}$ (as in CLS+), a set of names for movement functions $\mathcal{M}$, and a set Υ used to capture the internal state of a movement function.

Definition 6.2.1. *Terms* T, *branes* B and *sequences* S of Spatial CLS are defined as:

$$\begin{array}{rcl} T & ::= & \lambda \;\mid\; \big(S\big)_d \;\mid\; \big(B\big)_d^L \rfloor T \;\mid\; T \mid T \\ B & ::= & \big(S\big)_d \;\mid\; B \mid B \\ S & ::= & \epsilon \;\mid\; a \;\mid\; S \cdot S \end{array}$$

where $d \in \mathcal{D} = ((\mathbb{R}^n \times (\mathcal{M} \times \Upsilon) \cup \{\cdot\}) \times \mathbb{R}^+$, with $n \in \{2, 3\}$. The set of all sequences, branes and terms are denoted by $\mathcal{S}$, $\mathcal{B}$ and $\mathcal{T}$, respectively. Similarly to CLS+ (Definition 6.1.1), note that $\mathcal{B} \subset \mathcal{T}$.

The term λ denotes the *empty term*, while ϵ denotes the empty sequence. The parameter d associated with the elements describes their spatial information. In Spatial CLS each term encoding a simple sequence $\big(S\big)_d$ or looping sequence $\big(B\big)_d^L$ has an associated spatial representation, modelling the space occupied by the element.

The distinction between *positional* and *non-positional* elements depends on the form of the parameter d, associated with elements. In the case of positional elements, the parameter d is of the form $d = \langle [p, m], r \rangle$, where: p represents the position of the elements, m denotes the movement function, and r is the radius of the exclusion space of the element. Because we assume that models are always given in either two-dimensional or three-dimensional space, the position p is expressed as a vector of either $\mathbb{R}^2$ or $\mathbb{R}^3$. For non-positional elements, their positional information is of the form $d = \langle \cdot, r \rangle$, where the position and movement function are replaced by a special symbol '$\cdot$' (*dot*). Note that the information on the radius is kept, as it expresses the space occupied by the element.

In the case of looping sequences $\big(\_\big)_d^L \rfloor \_$ the radius r also describes the space available inside the membrane for the contained elements. The position of positional elements contained inside a looping sequence, or appearing on its brane, is relative to the center of the looping sequence itself. In this way, if the position of a looping sequence is updated, all its internal elements are moved together with it, without having to update their positions.

Parameter m denotes a pair composed of the name of a movement function $n \in \mathcal{M}$, and a $\upsilon \in \Upsilon$ capturing the internal state of the movement function. Each $n \in \mathcal{M}$ denotes a function n_{fun} describing the movement of an element over time. In particular, n_{fun} computes a tuple (P, Π, υ), where $P \subseteq \mathbb{R}^n$ are the possible resulting positions for the element, $\Pi : P \to [0, 1]$ is a probability distribution function describing the probability of reaching each one of the possible positions $p' \in P$, and υ is the new updated state of the movement function. We assume the set P of resulting positions to be finite.

A movement function $(P, \Pi, \upsilon') = n_{\text{fun}}(p, r, x, l, t, \delta t, \upsilon)$ describes the possible resulting positions of the element after a time interval δt from the current time t, by also taking into account the following parameters: (i) its current position $p \in \mathbb{R}^n$; (ii) its radius $r \in \mathbb{R}^+$; (iii) a parameter $x \in \{in, on\}$ specifying if the element appears on the surface or inside a membrane; (iv) a parameter $l \in \mathbb{R}^+ \cup \{\infty\}$, specifying the radius of the parent membrane; (v) the previous internal state υ of the movement function. In particular, value *on* of parameter x denotes that the element appears on the surface of a membrane, whereas value *in* denotes that the element is either inside a membrane with radius l, or at top–level (in such case $x = in$, $l = \infty$). The parameter υ and return value υ' allows the function to maintain an internal state between each evaluation.

This formalization of Spatial CLS allows direct description of stochastic motions. This feature is particularly important for modelling biological systems, as it allows representing common non-deterministic movements, such as *Brownian motion*. See Section 6.5 for examples of definitions of movement function.

The calculus also provides the following structural congruence relations.

Definition 6.2.2. The *structural congruence* relations on sequences $\equiv_S$ and on terms $\equiv_T$ are the least congruences satisfying the following rules:

$$S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3$$
$$S \cdot \epsilon \equiv_S \epsilon \cdot S \equiv_S S$$
$$S_1 \equiv_S S_2 \Rightarrow \left(S_1\right)_d \equiv_T \left(S_2\right)_d$$

$$T \mid \lambda \equiv_T T$$
$$T_1 \mid (T_2 \mid T_3) \equiv_T (T_1 \mid T_2) \mid T_3$$
$$T_1 \mid T_2 \equiv_T T_2 \mid T_1$$

6.2.2 Well-formed terms

Spatial CLS allows representing the space occupied by an element by defining an exclusion space around it. As we have anticipated, the exclusion space is modelled as a circle (in 2D) or a sphere (in 3D), described by the radius parameter. Formally, we define the set of well-formed terms, which captures the constraint that different elements cannot have overlapping exclusion spaces. Actually, we define a rather strict notion of well-formed terms, which also takes into account the space available inside looping sequences, by requiring that there is enough room to accommodate all the internal elements (both positional and non-positional). Note, however, that different definitions of well-formedness of terms can be used if necessary.

As regards positional elements, we introduce the following constraints:

- elements inside a membrane (or at top–level) must not occupy the same space;

- elements of a brane must not occupy the same space;
- elements of a brane must not occupy the space of any other element either inside or outside the membrane;
- elements inside a membrane must not exceed the limits of the exclusion space representing the membrane;
- the center of the elements of a brane must be located exactly at a distance r from brane center, where r is the radius of the exclusion space.

Moreover, we want to ensure that the space occupied by all the elements in a membrane does not exceed the volume of the membrane. To take into account the space occupied by non–positional elements, we assume a function SpaceCheck that determines whether there is enough space in a membrane for all the elements inside it and for all those on its surface.

The described constraints are captured by the following definition of *well-formedness*, where we assume the function $\mathrm{dist} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^+$ that gives the Euclidean distance between two points.

Definition 6.2.3 (Well-formed terms)**.** The set of *well-formed terms* is defined as[1]

$$\mathcal{T}_{wf} = \{T \in \mathcal{T} \mid \exists I \in \mathcal{I}.\ I \models T\}$$

where $\mathcal{I} = \mathscr{P}(\mathcal{J} \times \mathscr{P}(\mathcal{J}))$, $\mathcal{J} = (\mathbb{R}^n \cup \{\cdot\}) \times \mathbb{R}^+$, and relation $\_ \models \_ \subseteq \mathcal{I} \times \mathcal{T}$ is defined by the following inference rules (where $\mathbf{0}$ denotes the null vector):

$$\emptyset \models \lambda \qquad \{(\cdot, r, \emptyset)\} \models (S)_{\cdot,r} \qquad \{(p, r, \emptyset)\} \models (S)_{[p,m],r}$$

$$\frac{\begin{array}{c} I_1 \models B \qquad I_2 \models T \qquad \mathrm{SpaceCheck}(r, I_1, I_2) = \mathit{true} \\ \forall\, (p_1, r_1) \in \mathrm{All}(I_1).\ \mathrm{dist}(\mathbf{0}, p_1) = r \qquad \forall\, (p_2, r_2) \in \mathrm{All}(I_2).\ \mathrm{dist}(\mathbf{0}, p_2) + r_2 \leq r \\ \forall\, (p_1, r_1) \in \mathrm{All}(I_1),\, (p_2, r_2) \in \mathrm{All}(I_2).\ \mathrm{dist}(p_1, p_2) \geq r_1 + r_2 \end{array}}{\{(\cdot, r, \mathrm{All}(I_1))\} \models (B)^L_{\cdot,r} \rfloor T}$$

$$\frac{\{(\cdot, r, J)\} \models (B)^L_{\cdot,r} \rfloor T}{\{(p, r, J)\} \models (B)^L_{[p,m],r} \rfloor T}$$

$$\frac{\begin{array}{c} I_1 \models T_1 \quad I_2 \models T_2 \\ \forall\, (p_1, r_1) \in \mathrm{All}(I_1),\, (p_2, r_2) \in \mathrm{All}(I_2).\ \mathrm{dist}(p_1, p_2) \geq r_1 + r_2 \end{array}}{I_1 \cup I_2 \models T_1 \mid T_2}$$

where $\mathrm{SpaceCheck} : \mathbb{R}^+ \times \mathcal{I} \times \mathcal{I} \to \{\mathit{true}, \mathit{false}\}$ is assumed, and $\mathrm{All} : \mathcal{I} \to (\mathbb{R}^n \times \mathbb{R}^+)$ is defined as $\mathrm{All}(I) = \bigcup_{(p,r,J) \in I \mid \cdot \neq p} \{(p, r)\} \cup \{(p + p', r') \mid (p', r') \in J \wedge p' \neq \cdot\}$.

[1] Symbol $\mathscr{P}$ denotes the powerset operator.

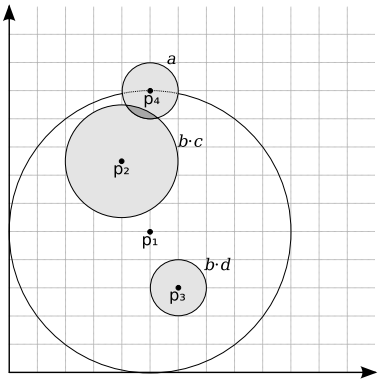


Figure 6.1: An example of non well-formed Spatial CLS term.

The above inference rules allow deriving pairs of the form $I \models T$, where I describes all the elements appearing at top-level in T and, for each top-level looping sequence, all the elements appearing on its brane. In particular, each tuple (p, r, J) in a set $I \in \mathcal{I}$ contains the spatial information of an element with position p and radius r. If the element is a simple sequence, J is empty. Otherwise, in the case of looping sequences, $J = \{(p_1, r_1), \ldots, (p_n, r_n)\}$ describes the spatial information of the elements appearing on its brane. A term T is well-formed iff there exists an I such that $I \models T$.

Since the position of elements in J is relative to p, the function All is defined such that it translates the position p' of all elements in J to $p + p'$. In particular, the function All gives, when applied to a tuple (p, r, J), a set containing: (1) the position and radius of the element itself (p, r); (2) the translated position and radius of each element on the brane.

The function SpaceCheck takes as parameters the radius r of the looping sequence, and the spatial descriptions of the elements inside it (I_2) and of those on the brane (I_1). It uses that information to determine if there is enough space for all the elements, i.e. if there exists an arrangement of the (non-positional) elements for which all the elements included in a membrane fit within the bounds. We expect this function to compute approximate solutions, in order to be efficiently computable.

Note that, given a term T, if a set I such that $I \models T$ exists, then it is unique. Therefore, the set I really describes the spatial information of (some) elements of term T, and such a set can be derived from the inference rules iff the term is well-formed. Moreover, note that for each possible term there is at most one applicable inference rule. Hence, if a set I exists for a given term T then there is a unique proof tree that allows $I \models T$ to be inferred. This means also that in order to prove whether well-formedness holds or not for a given term T, it is sufficient to try to construct the corresponding proof tree by applying the rules in the only possible way. Finally, it is also easy to see that well-formedness is preserved by structural congruence namely, given a set I and terms T_1, T_2 such that $T_1 \equiv T_2$, then $I \models T_1$ iff $I \models T_2$.

Function SpaceCheck is used to determine if there is sufficient space inside a looping sequence to accommodate all the elements, both positional and non-positional, contained in it. A simple definition of SpaceCheck function is the one which computes the available space by subtracting the volume of the positional elements to the volume of the considered looping sequence and give *true* if the result is greater than the sum of the volumes of the non-positional elements. More precise versions could be used if necessary.

Example As an example of application of the well-formedness rules (in which we assume SpaceCheck to be defined as in the example given above) let us consider the following Spatial CLS term:

$$T \ ::= \ \big(\big(a\big)_{\langle [p_4,m],1\rangle}\big)^{L}_{\langle [p_1,m],5\rangle} \rfloor \big(\big(b\cdot c\big)_{\langle [p_2,m],2\rangle} \mid \big(b\cdot d\big)_{\langle [p_3,m],1\rangle}\big)$$

where $p_1 = (5,5), p_2 = (-1,2.5), p_3 = (1,-2)$ and $p_4 = (0,5)$. A visual representation of term T is given in Figure 6.1. By the following proof tree we have that the content of the looping sequence of T is well-formed:

$$\frac{\{(p_2,2,\emptyset)\} \models \big(b\cdot c\big)_{\langle [p_2,m],2\rangle} \qquad \{(p_3,1,\emptyset)\} \models \big(b\cdot d\big)_{\langle [p_3,m],1\rangle} \quad \mathrm{dist}(p_2,p_3) = \sqrt{2^2+4.5^2} = 4.92 \geq 2+1 = 3}{\{(p_2,2,\emptyset),(p_3,1,\emptyset)\} \models \big(b\cdot c\big)_{\langle [p_2,m],2\rangle} \mid \big(b\cdot d\big)_{\langle [p_3,m],1\rangle}}$$

However, the whole term T is not well-formed because sequence a overlaps with sequence $b\cdot c$. In fact, in the derivation of a proof tree for the whole term we have

$$\begin{array}{l} \{(p_4,1,\emptyset)\} \models \big(a\big)_{\langle [p_4,m],1\rangle} \\ \{(p_2,2,\emptyset),(p_3,1,\emptyset)\} \models \big(b\cdot c\big)_{\langle [p_2,m],2\rangle} \mid \big(b\cdot d\big)_{\langle [p_3,m],1\rangle} \\ \mathrm{SpaceCheck}(5,\{(p_4,1,\emptyset)\},\{(p_2,2,\emptyset),(p_3,1,\emptyset)\}) = \mathit{true} \\ \mathrm{dist}(\mathbf{0},p_4) = 5 \\ \mathrm{dist}(\mathbf{0},p_2)+2 = \sqrt{1^2+2.5^2}+2 = 4.69 \leq 5 \\ \mathrm{dist}(\mathbf{0},p_3)+1 = \sqrt{1^2+2^2}+1 = 2.24 \leq 5 \end{array}$$

but also

$$\mathrm{dist}(p_4,p_2) = \sqrt{1^2+2.5^2} = 2.69 < 1+2 = 3$$

that is a violation of a requirement in the premise of the inference rule for looping sequences.

6.2.3 Patterns and rewrite rules

As for CLS+, in Spatial CLS we have different kinds of pattern. In Spatial CLS, however, we also distinguish between brane patterns appearing on the left and on the right part of a rewrite rule. The sets of variables $\mathcal{X}$, SV, BV and TV are assumed as in CLS+, with $\mathcal{V} = \mathcal{X} \cup SV \cup BV \cup TV$. Moreover, we assume a set of position variables PV ranged over by $u, v, \ldots$. We distinguish between the instantiation of variables $\mathcal{V}$ and that of position variables PV. An *instantiation function* for variables in $\mathcal{V}$ is a partial function $\sigma : \mathcal{V} \to \mathcal{T}_{wf} \cup \mathcal{B} \cup \mathcal{S} \cup \mathcal{E}$ that respects the type of variables, while the one for position variables is a partial function $\tau : PV \to \mathcal{D}$. We denote by Σ and $\mathbf{T}$ the sets of all instantiation functions of the two kinds, respectively.

The distinction between left and right patterns is required to handle the spatial information associated with elements. Position variables $u \in PV$ are associated with elements of the left pattern to capture the spatial information of the elements to which the pattern is instantiated, while elements on right patterns can use the information associated with

position variables to compute their new spatial information. Technically, each element on a right pattern is associated with function $g : \mathbf{T} \to \mathcal{D}$, which uses the current instantiation $\tau \in \mathbf{T}$ of position variables from the left pattern to compute the updated spatial information $d \in \mathcal{D}$ of the element.

Definition 6.2.4 (Sequence and brane patterns)**.** *Left brane patterns* BP_L, *right brane patterns* BP_R, and *sequence patterns* SP are defined by the following grammar:

$$\begin{aligned} BP_L &::= \big(SP\big)_u \;\mid\; BP_L \,|\, BP_L \\ BP_R &::= \big(SP\big)_g \;\mid\; BP_R \,|\, BP_R \\ SP &::= \epsilon \;\mid\; a \;\mid\; SP \cdot SP \;\mid\; \widetilde{x} \;\mid\; x \end{aligned}$$

where $u \in PV$, $g : \mathbf{T} \to \mathcal{D}$. We denote the sets of all left and right brane patterns, and sequence patterns, by $\mathcal{BP}_L$, $\mathcal{BP}_R$ and $\mathcal{S}$, respectively.

Definition 6.2.5 (Term patterns)**.** *Left patterns* P_L and *right patterns* P_R are given by the following grammar:

$$\begin{aligned} P_L &::= \big(SP\big)_u \;\mid\; \big(BP_{LX}\big)^L_u \rfloor P_{LX} \;\mid\; P_L \,|\, P_L \\ BP_{LX} &::= BP_L \;\mid\; BP_L \,|\, \overline{X} \;\mid\; \overline{X} \\ P_{LX} &::= P_L \;\mid\; P_L \,|\, X \\ \\ P_R &::= \lambda \;\mid\; \big(SP\big)_g \;\mid\; \big(BP_{RX}\big)^L_g \rfloor P_R \;\mid\; P_R \,|\, P_R \;\mid\; X \;\mid\; \overline{X} \\ BP_{RX} &::= BP_R \;\mid\; BP_{RX} \,|\, \overline{X} \;\mid\; \overline{X} \end{aligned}$$

where $u \in PV$, $g : \mathbf{T} \to \mathcal{D}$. We denote the sets of all left patterns by $\mathcal{P}_L$, the set of all right patterns by $\mathcal{P}_R$, and we assume them to be supersets of $\mathcal{BP}_L$ and $\mathcal{BP}_R$, respectively. We denote by $\mathrm{Var}(P)$ the set of all variables appearing in a pattern P, including position variables from PV.

Patterns form rewrite rules, which are used to model the reactions that can occur in the system. Conceptually, a reaction occurs among the elements of the system that match the sequences (simple and looping) appearing in the left pattern. Term and brane variables appearing in the left pattern are used as placeholders for the other elements of a compartment which are not involved in the reaction. Formally, rewrite rules are defined as follows.

Definition 6.2.6. A *rewrite rule* is a 4-tuple (f_c, P_L, P_R, k), usually written as

$$[\, f_c \,] \quad P_L \overset{k}{\mapsto} P_R$$

where $f_c : \mathbf{T} \to \{true, false\}$, $k \in \mathbb{R}^+$, $\mathrm{Var}(P_R) \subseteq \mathrm{Var}(P_L)$, and each function g appearing in P_R refers only to position variables in $\mathrm{Var}(P_L)$. A rewrite rule where P_L and P_R are brane patterns BP_L and BP_R, respectively, is called *brane (rewrite) rule.*

A term can be obtained from a pattern by applying a pair of instantiation functions τ and σ to it. This entails the instantiation of the variables of the pattern and, for right patterns P_R, the replacement of each function $g : \mathbf{T} \to \mathcal{D}$ with the value obtained by

applying each of them to τ. A rewrite rule states that, if there exists a pair of instantiation functions τ and σ such that $P_L\tau\sigma$ matches a subterm of the current system, then that subterm may be rewritten to $P_R\tau\sigma$. The function τ conceptually carries the bindings between position variables of the left pattern of the rule and the actual spatial information (radius and, possibly, position and movement function) of the matched elements. In this way, the g functions are used to compute the spatial information for the elements on the right patterns, by using the spatial information of the elements that match with the left pattern.

The rewrite rule, besides left and right patterns, is formed by a function f_c that specifies its *application constraints*, that is whether or not the rule can be applied to specific matching elements. The applicability of the rule is determined by evaluating function f_c over the τ used for the matching. For instance, this function may be used to check the positions of the involved elements, and to allow the reaction only if they are close enough.

Similarly to the variant of CLS called Stochastic CLS [4], rewrite rules are endowed with a *kinetic* constant $k \in \mathbb{R}^+$, describing their propensity of application. Formally, k represents the parameter of a negative exponential distribution modelling the expected duration of a reaction involving a specific combination of reactants.

Example An example of Spatial CLS rewrite rule is the following, where dist denotes the function that gives the Euclidean distance between two points:

$$[\,\mathrm{dist}(p_1, p_2) \leq 10\,] \quad \big(b \cdot c\big)_{[p_1,m],r_1} \,|\, \big(b \cdot d\big)_{[p_2,m],r_2} \overset{5}{\mapsto} \big(b \cdot c \cdot d\big)_{[\frac{p_1+p_2}{2},m],r_1+r_2}\,.$$

This rule can be applied to a sequence $b \cdot c$ and a sequence $b \cdot d$ if the distance between them is less than 10. The result of the application of the rule is a single sequence $b \cdot c \cdot d$ with a position that is in the middle of the positions of $b \cdot c$ and $b \cdot d$ and with a radius that is the sum of the radii of the two sequences. The kinetic constant associated with the rule is 5, meaning that each occurrence of the reaction modelled by the rule lasts 0.2 time units on the average.

6.3 Spatial CLS Semantics

A biological system, described by a term and a set of rewrite rules, evolves by performing a sequence of steps. A step represents the evolution of the system in a finite time-span, and is conceptually composed of two phases:

1. *at most one* reaction occurs;
2. the objects are moved according to their movement functions.

During the evolution, the time length of the step varies to accommodate for the different number of possible reactions that can occur in each state. In particular, since the processes we describe are continuous-time stochastic processes, the assumption that at most one reaction occurs at each step is justified by choosing a step which is short enough.

In the first phase, the application of a rewrite rule could yield to a non well-formed term (according to Definition 6.2.3), i.e. space conflicts arise. For this reason, if the application of a rewrite rule would yield to a space conflict, the semantics tries to perform a

rearrangement of the objects in order to find a conflict-free spatial arrangement. Formally, this rearrangement is performed by the Arrange algorithm, which has to be provided by the modeller. However, if a conflict-free arrangement cannot be found (as denoted by a special value returned by Arrange), we forbid the particular application which causes the problem. In a state, we define the *enabled* applications of rewrite rules, as all those possible applications for which a conflict-free arrangement can be found. Note that the application of Arrange algorithm is orthogonal to the evolution of the system, as it consumes *no time* during the evolution of the system.

In order to derive the semantics of a Spatial CLS term, the modeller has to define a parameter $N \in \mathbb{N}$, which determines the maximum allowed step length as $1/N$. However, during the evolution, the actual step length can be much smaller, as at most one reaction per step may occur. Formally, if there is at least one *enabled* rule application, the actual length of the step is a fraction of the maximum step length, namely $1/(N\, m_T)$, where m_T denotes the total number of enabled rules in the current state T. As discussed in Section 6.3.2, the choice of N is not completely free, but it has to satisfy some constraints in order to assume that at most one reaction occurs in a step.

In the second phase, the positions of all the (positional) elements of the system are updated according to their movement functions. The positions of elements are *atomically* updated to the positions reached after a time-span corresponding to the length of the step. As for the first phase, the movement can yield to a non-well-formed term, therefore the semantics performs a rearrangement also after moving the objects. However, differently from the first phase, if a conflict-free arrangement is not found the objects are kept in their current positions (which actually means skipping the movement phase). As before, the application of Arrange algorithm consumes no time during the evolution of the system.

The maximum length of the step affects the precision in describing the movement, as the semantics takes into account only the position of objects at the beginning and at the end of a step, excluding all intermediate positions. For example, in the case of a fast-moving object and a big step length, it may happen that some interactions, which would occur by considering intermediate positions, go unnoticed. A similar problem may occur if the rearrangement after a movement fails and the objects are not moved. It is up to the modeller to choose a value of parameter N which determines a smaller length of the step, thus mitigating these problems by allowing the semantics to see intermediate states.

Non-positional elements correspond to Stochastic CLS terms, thus their behaviour is in accordance with the law of Mass Action: they are assumed to be homogeneously distributed in the space available inside the compartment, and the reaction rate of the rules involving those elements is proportional to the product of the concentrations of the reactants.

We remark that distinguishing between positional and non-positional elements allows using two different levels of abstraction in the same biological model. Details about spatial information can be included only for those elements for which spatiality has a significant role. Such details can be omitted for elements that can be safely assumed to be homogeneously distributed in the space. This can improve the efficiency of simulators and analysis techniques based on Spatial CLS.

We now introduce some auxiliary definitions that will be used in the semantics.

$$\frac{\left(R : [\, f_c \,] \quad P_L \overset{k}{\mapsto} P_R\right) \in \mathcal{R} \qquad f_c(\tau) = true \qquad \tau \in \mathbf{T} \qquad \sigma \in \Sigma}{P_L\tau\sigma \xrightarrow{R,P_L\tau\sigma,comb(P_L,\tau,\sigma)}_{\text{appl}} P_R\tau\sigma}$$

$$\frac{B \xrightarrow{R,T,c}_{\text{appl}} B' \qquad R \in \mathcal{R}_{\mathcal{B}}}{\left(B\right)^L_d \rfloor T_1 \xrightarrow{R,(B)^L_d \rfloor T_1,c}_{\text{appl}} \left(B'\right)^L_d \rfloor T_1} \qquad \frac{T_1 \xrightarrow{R,T,c}_{\text{appl}} T_1'}{\left(B\right)^L_d \rfloor T_1 \xrightarrow{R,(B)^L_d \rfloor T_1,c}_{\text{appl}} \left(B\right)^L_d \rfloor T_1'}$$

$$\frac{T_1 \xrightarrow{R,T,c}_{\text{appl}} T_1'}{T_1 \mid T_2 \xrightarrow{R,T,c\cdot binom(T,T_1,T_2)}_{\text{appl}} T_1' \mid T_2}$$

Figure 6.2: The inference rules defining transition relation $\rightarrow_{\text{appl}}$, used for computing the rate of a rule application.

6.3.1 Auxiliary definitions

The first definition is used to find all possible rule applications in a term, and it will be used by the semantics to compute the rate of a rule application. In particular, the transition relation defined in Figure 6.2 allows determining each group of elements to which a rewrite rule can be applied. These definitions follow closely those of Stochastic CLS given in [4], to which we refer the reader for more details.

Each transition $T_1 \xrightarrow{R,T_r,c}_{\text{appl}} T_2$ describes the application of rewrite rule R inside term T_1 yielding to T_2. Label T_r is a term denoting the position inside T_1 where the rewrite rule have been applied. We consider two reactions to be different if they involve different reactants T_r or different resulting terms T_2.

The value $c \in \mathbb{N}$ corresponds to the number of different reactant combinations among which the reaction described by R may, conceptually, occur. For example, if we consider a rewrite rule involving non-positional elements, such as $\left(a\right)_{\cdot,0} \mid \left(b\right)_{\cdot,0} \overset{k}{\mapsto} \left(a \cdot b\right)_{\cdot,0}$, then the reaction can conceptually occur between each pair of elements $\left(a\right)_{\cdot,0}$ and $\left(b\right)_{\cdot,0}$ contained in a compartment. Nevertheless, all of them yield to the same term, obtained by replacing one $\left(a\right)_{\cdot,0}$ and one $\left(b\right)_{\cdot,0}$ with $\left(a \cdot b\right)_{\cdot,0}$. Thus, in this case, the value c is the number of pairs of elements that can react, which is equal to $\binom{\#a}{1}\binom{\#b}{1}$, where $\#a$, $\#b$ denote the number of elements $\left(a\right)_{\cdot,0}$ and $\left(b\right)_{\cdot,0}$ in the compartment, respectively.

Formally, given a finite set of rewrite rules $\mathcal{R}$, with $\mathcal{R}_{\mathcal{B}} \subseteq \mathcal{R}$ denoting the set of all brane rules in $\mathcal{R}$, the transition relation $\xrightarrow{R,T_r,c}_{\text{appl}}$, with $R \in \mathcal{R}$, $T_r \in \mathcal{T}$ and $c \in \mathbb{N}$, is defined as the least labeled transition relation on terms defined by the inference rules shown in Figure 6.2, and closed with respect to $\equiv_T$. The definition in Figure 6.2 makes use of the functions *comb*, *comb*′ and *binom*, which allow to compute, in a compositional way, the number of possible reactant combinations associated with a precise application of a rewrite rule.

Let $\overline{P}$ denote the multiset of top-level elements appearing in a pattern (or term) P, and assume the function $\mathbf{n} : \mathcal{T} \times \mathcal{T} \rightarrow \mathbb{N}$ that, applied to a term T_1, representing a (simple or looping) sequence, and another term T_2, gives the number of times T_1 appears at top-level in T_2. Functions $comb, comb' : \mathcal{P}_L \times \mathbf{T} \times \Sigma \rightarrow \mathbb{N}$ and $binom : \mathcal{T} \times \mathcal{T} \times \mathcal{T} \rightarrow \mathbb{Q}$ are

recursively defined as follows:

$$\begin{aligned} comb(P_{L1} \mid P_{L2}, \tau, \sigma) &= comb(P_{L1}, \tau, \sigma) \cdot comb(P_{L2}, \tau, \sigma) \\ comb(\big(BP_{LX}\big)^{L}_{u} \rfloor P_{LX}, \tau, \sigma) &= comb'(BP_{LX}, \tau, \sigma) \cdot comb'(P_{LX}, \tau, \sigma) \\ comb(\big(SP\big)_{u}, \tau, \sigma) &= 1 \end{aligned}$$

$$comb'(P_L \mid U, \tau, \sigma) = \prod_{T \in \overline{P_L \tau \sigma}} \binom{\mathbf{n}((P_L|U)\tau\sigma, T)}{\mathbf{n}(P_L\tau\sigma, T)} \cdot comb(P_L, \tau, \sigma) \quad U \in BV \cup TV$$

$$comb'(P_L, \tau, \sigma) = comb(P_L, \tau, \sigma)$$

$$binom(T_1, T_2, T_3) = \prod_{T \in \overline{T_1}} \prod_{i=1}^{\mathbf{n}(T_3,T)} \frac{\mathbf{n}(T_2, T) + i}{\mathbf{n}(T_2, T) - \mathbf{n}(T_1, T) + i}$$

Using the definition of transition relation $\rightarrow_{\text{appl}}$, we define the ApplTargets(T) function that gives the set of terms reachable after applying a rewrite rule to the term T, by also considering the subsequent rearrangement. Formally, it is defined as follows:

$$\text{ApplTargets}(T) = \left\{ T'' \;\middle|\; T \xrightarrow{R,T_r,c}_{\text{appl}} T' \wedge T'' = \text{Arrange}(T') \neq \bot \right\}$$

The second auxiliary definition that we introduce is used to perform the movement phase inside a step. We define the transition relation $\langle T, t, \delta t\rangle \xrightarrow{x,l,\pi,ps}_{\text{mov}} T'$, where $x \in \{in, on\}$, $l \in \mathbb{R}^+ \cup \{\infty\}$, $\pi \in [0, 1]$, and ps is a list of positions, as the least transition relation defined by the rules shown in Figure 6.3. Given the initial term T, describing the current state at time t, the transition relation allows deriving all reachable terms T', describing the state after a time interval δt from the current time t, where the positions of all positional elements appearing in T have been updated using the movement functions of the elements. Label π is the probability of performing the transition, and is obtained by combining the probability of reaching the different positions of each movement function. Label x denotes where term T appears. The value $x = in$ means that T is either inside a looping sequence or at top-level (i.e. it is neither inside nor in the brane of any looping sequence), while the value $x = on$ means that T is on the brane of a looping sequence. In both cases, l represents the radius of the looping sequence where T appears, or, if T is at top-level, $l = \infty$. Hence, if T is at top-level we have $x = in$ and $l = \infty$.

Finally, the transition keeps track of the positions chosen by each movement function in the label ps, which contains the list of chosen positions. This is just a technical trick that allows us to keep distinct those transitions that would produce the same resulting term T' with the same probability π, even if the positions chosen by movement function are different. This works because structural congruence is not considered in the definition.

Using the transition relation $\rightarrow_{\text{mov}}$, we define the function MovTargets($T, t, \delta t$) that gives the set of terms reachable after the movement phase, by also taking into account the rearrangement:

$$\text{MovTargets}(T, t, \delta t) = \left\{ T'' \;\middle|\; \langle T, t, \delta t\rangle \xrightarrow{in,\infty,\pi,ps}_{\text{mov}} T' \wedge T'' = \text{Arrange}(T') \neq \bot \right\}$$

$$\frac{x \in \{in, on\} \qquad l \in \mathbb{R}^+}{\langle \lambda, t, \delta t\rangle \xrightarrow{x,l,1,[]}_{\text{mov}} \lambda}$$

$$\frac{x \in \{in, on\} \qquad l \in \mathbb{R}^+}{\langle (S)_{\cdot,r}, t, \delta t\rangle \xrightarrow{x,l,1,[]}_{\text{mov}} (S)_{\cdot,r}}$$

$$\frac{\begin{array}{c}(P, \Pi, \sigma') = n_{\text{fun}}(p, r, x\ l, t, \delta t, \sigma)\\ p' \in P \qquad m = (n, \sigma) \qquad x \in \{in, on\} \qquad l \in \mathbb{R}^+\end{array}}{\langle (S)_{[p,m],r}, t, \delta t\rangle \xrightarrow{x,l,\Pi(p'),[p']}_{\text{mov}} (S)_{[p',(n,\sigma')],r}}$$

$$\frac{\begin{array}{c}\langle B_1, t, \delta t\rangle \xrightarrow{on,r,\alpha,ps_B}_{\text{mov}} B_2 \qquad \langle T_1, t, \delta t\rangle \xrightarrow{in,r,\beta,ps_T}_{\text{mov}} T_2\\ (P, \Pi, \sigma') = n_{\text{fun}}(p, r, x\ l, t, \delta t, \sigma)\\ p' \in P \qquad m = (n, \sigma) \quad \pi = \alpha\beta\Pi(p') \quad x \in \{in, on\} \quad l \in \mathbb{R}^+\end{array}}{\langle (B_1)^L_{[p,m],r} \rfloor T_1, t, \delta t\rangle \xrightarrow{x,l,\pi,[p']@ps_B@ps_T}_{\text{mov}} (B_2)^L_{[p',(n,\sigma')],r} \rfloor T_2}$$

$$\frac{\begin{array}{c}\langle B_1, t, \delta t\rangle \xrightarrow{on,r,\alpha,ps_B}_{\text{mov}} B_2 \qquad \langle T_1, t, \delta t\rangle \xrightarrow{in,r,\beta,ps_T}_{\text{mov}} T_2\\ x \in \{in, on\} \quad l \in \mathbb{R}^+\end{array}}{\langle (B_1)^L_{\cdot,r} \rfloor T_1, t, \delta t\rangle \xrightarrow{x,l,\alpha\beta,ps_B@ps_T}_{\text{mov}} (B_2)^L_{\cdot,r} \rfloor T_2}$$

$$\frac{\langle T_1, t, \delta t\rangle \xrightarrow{x,l,\alpha,ps_B}_{\text{mov}} T_1' \qquad \langle T_2, t, \delta t\rangle \xrightarrow{x,l,\beta,ps_T}_{\text{mov}} T_2'}{\langle T_1 \mid T_2, t, \delta t\rangle \xrightarrow{x,l,\alpha\beta,ps_B@ps_T}_{\text{mov}} T_1' \mid T_2'}$$

Figure 6.3: Rules of the transition relation $\rightarrow_{\text{mov}}$.

6.3.2 Definition of the semantics

We assume a function Arrange : $\mathcal{T} \rightarrow (\mathcal{T} \cup \{\perp\})$ which tries to rearrange the elements of a system in case of a space conflict. Thus, given a term T, Arrange(T) either produces a well-formed term T', or returns the special value $\perp$ denoting that no conflict-free arrangement could be found. We assume that, if the given T is well-formed, then Arrange(T) = T.

Given a term T, let us denote by $m_T^{(R)}$ the number of different reactant combinations enabled in state T for a reaction R, and by m_T the total number of reactions considering a set of rules $\mathcal{R}$. Formally:

$$m_T^{(R)} = \sum \left\{ c \;\middle|\; T \xrightarrow{R,T_r,c}_{\text{appl}} T' \wedge \text{Arrange}(T') \neq \perp \right\}$$

$$m_T = \sum_{R \in \mathcal{R}} m_T^{(R)}.$$

Note that we explicitly exclude all rule applications that would not yield to a well-formed term after the rearrangement.

Let T describe the state of the system at a certain step, and k_R denote the rate associated with a rewrite rule R. At each step of the evolution of the system, in order

to assume that at most one reaction can occur, we have to choose a time interval δt such that $\left(\sum_{R\in\mathcal{R}} k_R\, m_T^{(R)}\right)\ \delta t \leq 1$. Given a set of rewrite rules $\mathcal{R}$, we let the modeller choose an arbitrary value N such that for each rule $R \in \mathcal{R}$, $0 < k_R/N \leq 1$. In this way, by using a time interval $\delta t = 1/(N\ m_T)$ for a step, we satisfy the above condition. If there are no enabled reactions, then the length of the step is $1/N$ time units, which corresponds to the maximum step length.

The following definitions are used to compute the probability associated with transitions in the semantics.

$$P_{appl}(T \to T'') = \sum \left\{ \frac{k_R\ c}{N m_T} \;\middle|\; T \xrightarrow{R,T_r,c}_{\text{appl}} T' \wedge T'' = \text{Arrange}(T') \right\} \tag{6.1}$$

$$P_{noappl}(T) = 1 - \sum \left\{ P_{appl}(T \to T') \mid T' \in \text{ApplTargets}(T) \right\} \tag{6.2}$$

$$P_{mov}(\langle T, t, \delta t\rangle \to T'') = \sum \left\{ \pi \;\middle|\; \langle T, t, \delta t\rangle \xrightarrow{in,\infty,\pi,ps}_{\text{mov}} T' \wedge T'' = \text{Arrange}(T') \right\} \tag{6.3}$$

$$P_{nomov}(T, t, \delta t) = 1 - \sum \left\{ P_{mov}(\langle T, t, \delta t\rangle \to T') \mid T' \in \text{MovTargets}(T) \right\} \tag{6.4}$$

Function $P_{appl}(T \to T'')$ gives the probability of performing a transition from T to T'' in the time interval $1/(N\, m_T)$, by summing up the probabilities of all the different rule applications which, after the rearrangement, end up in the term T''. The probability of a single rule application is $\frac{k_R\, c}{N m_T}$, where $k_R\, c$ is the kinetic rate (expected number of reactions per time unit). Function $P_{noappl}(T)$ gives the probability that no reaction occurs in term T. Similarly, function $P_{mov}(\langle T, t, \delta t\rangle \to T'')$ gives the probability of passing from state T at time t, to state T'' after a time interval δt, by performing the movement phase and the rearrangement. Finally, function $P_{nomov}(T, t, \delta t)$ gives the probability that no movement is performed, by considering the cases in which no conflict-free arrangement is found.

The following definition presents the semantics of Spatial CLS, given as a *probabilistic transition system*. Two kinds of states are present. States of the form $\langle T, t\rangle$ describe the system at time t. These states represent the beginning of a step, in which the first phase has to take place. All transitions from a state of this kind are of the form $\langle T, t\rangle \xrightarrow{p} \langle T', t, \delta t\rangle$, where each transition represents the application of at most one rewrite rule, with probability p. In particular, a state of the form $\langle T', t, \delta t\rangle$ is reached, denoting that the second phase has to take place, and where δt is the length of the time interval for the step. Transitions $\langle T', t, \delta t\rangle \xrightarrow{p} \langle T'', t'\rangle$, with $t' = t + \delta t$, describe the possible effects of the movement of the objects. Each possible resulting term T'' of the objects has a probability p of being reached.

Definition 6.3.1 (Semantics)**.** Given a finite set of rewrite rules $\mathcal{R}$, the semantics of

Spatial CLS is the least relation satisfying the following inference rules:

$$\frac{p = P_{appl}(T_1 \to T_2) \qquad T_1 \not\equiv T_2 \qquad T_2 \in \text{ApplTargets}(T_1) \qquad \delta t = 1/(N\, m_{T_1})}{\langle T_1, t\rangle \xrightarrow{p} \langle T_2, t, \delta t\rangle} \tag{6.5}$$

$$\frac{p = P_{appl}(T \to T) + P_{noappl}(T) \qquad \delta t = 1/(N\ \max\{1, m_T\})}{\langle T, t\rangle \xrightarrow{p} \langle T, t, \delta t\rangle} \tag{6.6}$$

$$\frac{p = P_{mov}(\langle T_1, t, \delta t\rangle \to T_2) \qquad T_1 \not\equiv T_2 \qquad T_2 \in \text{MovTargets}(T_1)}{\langle T_1, t, \delta t\rangle \xrightarrow{p} \langle T_2, t + \delta t\rangle} \tag{6.7}$$

$$\frac{p = P_{mov}(\langle T, t, \delta t\rangle \to T) + P_{nomov}(T, t, \delta t)}{\langle T, t, \delta t\rangle \xrightarrow{p} \langle T, t + \delta t\rangle} \tag{6.8}$$

Inference rules 6.5 and 6.6 describe the first phase of a step, while rules 6.7 and 6.8 describe the second phase of a step. In particular, rule 6.5 is used to derive transitions in which a reaction occurs (i.e. a rewrite rule is applied) and ending up in a different term T_2 from T_1. Rule 6.6 allows deriving the transition in which the resulting state is not modified. The probability of this transition includes both (i) the probability $P_{noappl}(T)$ that no reaction occurs in the step, and (ii) the probability $P_{appl}(T \to T)$ of all the reactions which happen to produce the same state T.

Rules 6.7 and 6.8 describe the second phase of a step, and are analogous to rules 6.5 and 6.6. In fact, rule 6.7 allows deriving transitions in which the resulting state T_2 is different from the initial state T_1, while rule 6.8 is for transitions in which the term T describing system state is not modified. Rule 6.8 takes into account both the cases in which a movement is performed, and the cases in which such a movement is not performed because the rearrangement fails.

6.4 Handling space conflicts

In Spatial CLS, during the evolution of the system, space conflicts may arise which are resolved by the algorithm Arrange, which has to be provided by the modeller. The reason space conflict may occur is that, in Spatial CLS, both rule application and movement of elements are discrete events. This is different from the reality in which, for example, the size of a growing cell increases incrementally. Hence, the growth of a cell is a continuous event which causes the other adjacent cells to be pushed away. Movement is also a continuous event in the reality. The idea of the Arrange algorithm is to approximate the reality by simulating the movement which would occur in the case of a conflict. The main difference from real systems is the time in which the arrangement is performed.

In this section we show a possible definition of the Arrange algorithm in which spatial conflicts are resolved by pushing objects in opposite directions when they overlap, and precisely tracking the movement that would arise. Note that the precision required depends on the kind of system one wants to model. Therefore the modeller could use simpler or more approximated (and, hopefully, faster) algorithms if such a high precision is not required for a model. Finally, recall that the arrangement is orthogonal with respect to the evolution of the system, since the arrangement is assumed to be instantaneous. Therefore

the movement simulated by this implementation of the Arrange algorithm is not exposed to the semantics, apart from the final arrangement possibly found.

The following definition of the Arrange algorithm is able to find a well-formed term according to Definition 6.2.3, by also taking into account the space available inside looping sequences for the inner elements. The behaviour of the Arrange algorithm is defined by assuming an instant velocity associated with each element, whose direction and speed depend on the instant position of every element. Given an element, the velocity it is subjected to is calculated as the sum of other velocities:

- for each other element it overlaps with, we assume a velocity, in the opposite direction to the other element (the elements are trying to increase their distance), and whose magnitude is proportional to the length of the overlap;
- if the element is not completely within the bounds of the containing membrane, then we assume a velocity directed towards the center of the membrane, whose magnitude is inversely proportional to the distance from the center;
- if the element is on the surface of a membrane, but its center is not located on the surface of the sphere modelling the exclusion space, then we assume a velocity directed towards the nearest point of the sphere with a magnitude proportional to the distance from the sphere.

If any of those velocities is not well-defined (for instance, if the centers of two elements coincide), then we assume an arbitrary fixed direction along which the elements move. We also assume that the radius of each element is not greater than the radius of the membrane in which it is contained; otherwise the conflict could not be resolved.

This behaviour is modelled by a system of differential equations. Given a term T, let $\vec{x_1}, \ldots, \vec{x_m}$ be the variables for the centers of the positional elements appearing in T, and I_L and I_S the set of indices denoting looping and simple sequences, respectively ($I_L \cup I_S = \{1, \ldots, m\}$). Moreover, let $\mathrm{In}(i)$, with $i \in I_L$, be the set of indices denoting elements inside the looping sequence i, or, if $i = 0$, the indices of the top–level elements; let $\mathrm{On}(i)$, with $i \in I_L$, denote the elements appearing on the surface of i (we assume $On(0) = \emptyset$) and $\mathrm{Inner}(i) = \mathrm{In}(i) \cup \bigcup_{j \in \mathrm{In}(i)} \mathrm{On}(j)$.

The system of differential equations on which the Arrange algorithm is based is the following, where h_i is such that $i \in \mathrm{Inner}(h_i)$:

$$\begin{cases} \dfrac{d\vec{x_i}}{dt} = \left(\displaystyle\sum_{j \in \mathrm{Inner}(h_i) \setminus \{i,k\} \cup \mathrm{In}(k)} \vec{v_{ij}} \right) - \vec{u_i} + \vec{w_i} & \forall i.\ \exists k.\ i \in \mathrm{On}(k) \\ \dfrac{d\vec{x_i}}{dt} = \left(\displaystyle\sum_{j \in \mathrm{Inner}(h_i) \setminus (\{i\} \cup \mathrm{On}(i))} \vec{v_{ij}} \right) - \vec{u_i} & \forall i \in \mathrm{In}(h_i) \end{cases}$$

Vectors $\vec{v_{ij}}$ denote velocities due to the overlap between two elements. Each vector $\vec{u_i}$ denote the velocity that models, for the elements that are not completely contained in a membrane, the movement towards the center of the membrane. Vectors $\vec{w_i}$ are relative to elements appearing on the surface of membranes, and that are not correctly positioned on the membrane surface itself. The formal definitions of these vectors are the following,

where $\vec{a}$, $\vec{b}$ denote non-null vectors specifying the directions used when velocities are not well defined:

$$\vec{y_i} = \begin{cases} \vec{x_i} + \vec{y_j} & \text{if } \exists j \neq 0.\ i \in \mathrm{On}(j) \vee i \in \mathrm{In}(j) \\ \vec{x_i} & \text{otherwise} \end{cases}$$

$$v_{ij} = \max\{0, r_i + r_j - \mathrm{dist}(\vec{y_i}, \vec{y_j})\}$$

$$\vec{v_{ij}} = \begin{cases} v_{ij}\,\widehat{(\vec{y_i} - \vec{y_j})} & \text{if } \vec{y_i} \neq \vec{y_j} \\ +v_{ij}\,\vec{a} & \text{if } \vec{y_i} = \vec{y_j} \text{ and } i < j \\ -v_{ij}\,\vec{a} & \text{if } \vec{y_i} = \vec{y_j} \text{ and } i \geq j \end{cases}$$

$$u_i = \begin{cases} \max\{0, \mathrm{dist}(\mathbf{0}, \vec{x_i}) + r_i - r_h\} & \text{if } \exists h \neq 0.\ i \in \mathrm{Inner}(h) \\ 0 & \text{otherwise} \end{cases}$$

$$\vec{u_i} = \begin{cases} \mathbf{0} & \text{if } i \in \mathrm{Inner}(0) \\ u_i\,\widehat{(\vec{y_h} - \vec{y_i})} & \text{if } \exists h \neq 0.\ i \in \mathrm{Inner}(h) \end{cases}$$

$$\vec{w_i} = \begin{cases} w_i\,\widehat{\vec{x_i}} & \text{if } \vec{x_i} \neq \mathbf{0} \\ w_i\,\vec{b} & \text{if } \vec{x_i} = \mathbf{0} \end{cases} \qquad w_i = r_k - \mathrm{dist}(\mathbf{0}, \vec{x_i}) \quad \exists k.\ i \in \mathrm{On}(k)$$

where $\widehat{\vec{x}}$ denotes the normalized vector $\vec{x}$, i.e. $\widehat{\vec{x}} = \vec{x}/\mathrm{dist}(\mathbf{0}, \vec{x})$ if $\vec{x} \neq \mathbf{0}$.

The Arrange algorithm stops when a stable setting is reached. If the term representing this setting is not well-formed, the algorithm returns the special value $\perp$, otherwise it returns a term where the positions of all the elements are updated. Note that, even if all space conflicts are resolved, still the term may not be well-formed according to Definition 6.2.3 because there is not enough space for the non-positional elements, as determined by the function SpaceCheck.

The use of this mechanism is not the only solution for dealing with space conflicts among the elements of a system. The modeller can provide its own definition of Arrange algorithm, tailored to specific needs. For example, a more physically sound modelling of the behaviour of the system in case of space conflicts, could be obtained by associating a "weight" and consequently a "pushing strength" with elements. The Arrange algorithm could take such weights into account, so that an element could push other elements according to its strength.

Implementation of Arrange algorithm

Algorithm 11 contains a specification of the Arrange algorithm, written in pseudo-code. The algorithm takes a term as input, and returns a well-formed term obtained from the input by simulating the system of differential equations shown above, if it is able to determine it. Otherwise, it returns a special value $\perp$. The behaviour of the algorithm is determined by the parameter Δt, representing the time step length. The algorithm performs a sequence of steps, using the algorithm *DoStep*() to compute the movement of each element in the current step, until the amount of movement for each element decreases below the threshold $\epsilon\,\Delta t$.

Algorithm 12 describes the *DoStep*() algorithm, which computes the vectors describing the movement according to the system of differential equations. Formally, given a vector

Algorithm 11 *Arrange*(T), with $T \in \mathcal{T}$

1: let $[\vec{x_i}]_{i=1,\ldots,m}$ and $[\vec{r_i}]_{i=1,\ldots,m}$ denote the position and radii of positional elements in T
2: **repeat**
3: $\quad \Delta x = DoStep(x, r)$
4: $\quad x = x + \Delta x$
5: **until** $\forall i \in \{1, \ldots, m\}$. $\text{dist}(\mathbf{0}, \Delta x_i) \leq \epsilon\, \Delta t$
6: $T' = T$ updated with positions x
7: **if** $T' \in \mathcal{T}_{wf}$ **then return** T' **else return** $\perp$ **end if**

x of positions of all positional elements, with radii described by vector r, the *DoStep*() algorithm computes a vector Δx representing the movement of each positional element in the current step. A parameter Δt denotes the time interval used for approximating the movement described by the system of differential equations. We assume $\Delta t < 1$. Note that a smaller value of Δt increases the precision, as the time interval is smaller.

The algorithm uses a set Q of indices of looping sequences, also including the special value 0, each of them representing a looping sequence whose content has still to be considered. Each iteration of the outermost "while" loop deals with the content of a different looping sequence, except at the first iteration, in which it deals with top-level elements. Therefore, the algorithm starts by computing the movement vectors for all top level elements, and elements appearing on the brane of top-level looping sequences. Then, the algorithm descends recursively into looping sequences, and so on, until all compartments have been considered. The "for" loop at lines 6–13 checks for conflicts between each pair of elements inside the current element i. The second "for" loop at lines 14–20 checks the elements appearing on the brane, by moving each of them towards a position at distance r_i from the center. Finally, lines 21–28 deal with inner elements, by moving towards the center the elements that exceed bounds.

Since the Arrange algorithm is used by the semantics of Spatial CLS, it is important to prove its termination. For the sake of simplicity we consider a restricted version of the algorithm dealing only with parallel compositions of sequences (Algorithm 13). We shall briefly discuss at the end of this section how the proof of termination could be extended to the complete algorithm (Algorithm 11).

Algorithm 13 (as Algorithm 11) performs a sequence of steps, each with a duration given by the parameter Δt. At each step, each element is subjected to a velocity which depends on the overlap with other elements, corresponding to vectors $\vec{v}$ of Algorithm 12. The algorithm terminates when the length of each movement vector goes below the threshold $\epsilon\, \Delta t$.

In order to prove the termination of the algorithm, we need to introduce some geometry concepts. A convex polyhedron P can be described as an intersection of a finite number of half-spaces. In the case of the two-dimensional space $\mathbb{R}^2$, this means that there must exist $A \in \mathbb{R}^{m \times 2}$ and $b \in \mathbb{R}^m$ such that

$$P = \{\, x \mid Ax \leq b \,\}$$

In the following, we deal only with *convex polygons*, which are finite convex polyhedrons in $\mathbb{R}^2$. Let I be a subset of row indices $\{1, \ldots, n\}$ of a matrix M. We denote by M_I

Algorithm 12 *DoStep*(x, r), with $x \in (\mathbb{R}^n)^m$ and $r \in \mathbb{R}^m$

$Q = \{0\}$
$\Delta x = \mathbf{0} \in (\mathbb{R}^n)^m$
while $Q \neq \emptyset$ **do**
    let $i \in Q$
    $Q = Q \setminus \{i\}$
    for all $j_1, j_2 \in \text{Inner}(i)$ such that $j_1 < j_2$ **do**
        $c = r_{j_1} + r_{j_2} - \text{dist}(\vec{y_{j_1}}, \vec{y_{j_2}})$
        if $c > \epsilon$ **then**
            $\vec{v} = \begin{cases} \widehat{(\vec{y_{j_1}} - \vec{y_{j_2}})} & \text{if } \vec{y_{j_1}} \neq \vec{y_{j_2}} \\ \vec{a} & \text{otherwise} \end{cases}$
            $\Delta x_{j_1} = \Delta x_{j_1} + c\,\vec{v}\,\Delta t$
            $\Delta x_{j_2} = \Delta x_{j_2} - c\,\vec{v}\,\Delta t$
        end if
    end for
    for all $j \in \text{On}(i)$ **do**
        $c = r_i - \text{dist}(\mathbf{0}, x_j)$
        if $|c| > \epsilon$ **then**
            $\vec{v} = \begin{cases} \widehat{x_j} & \text{if } x_j \neq \mathbf{0} \\ \vec{b} & \text{otherwise} \end{cases}$
            $\Delta x_j = \Delta x_j + c\,\vec{v}\,\Delta t$
        end if
    end for
    if $i \neq 0$ **then**
        for all $j \in \text{Inner}(i)$ **do**
            $c = \text{dist}(\mathbf{0}, x_j) + r_j - r_i$
            if $c > \epsilon$ **then**
                $\Delta x_j = \Delta x_j + c\,\widehat{(\vec{y_i} - \vec{y_j})}\,\Delta t$
            end if
        end for
    end if
    $Q = Q \cup (\text{In}(i) \cap I_L)$
end while
return Δx

Algorithm 13 *Arrange*$(\left(S_1\right)_{d_1} \mid \cdots \mid \left(S_n\right)_{d_n})$, with $\left(S_i\right)_{d_i} \in \mathcal{T}$

let $[\vec{x_i}]_{i=1,\ldots,n}$ and $[\vec{r_i}]_{i=1,\ldots,n}$ denote the position and radii of the positional elements in input
repeat
  $\Delta x_i = \mathbf{0} \in \mathbb{R}^2 \quad i \in \{1, \ldots, n\}$
  for all $(j_1, j_2) \in \{1, \ldots, n\} \times \{1, \ldots, n\}$ such that $j_1 \neq j_2$ **do**
    $c = r_{j_1} + r_{j_2} - \text{dist}(\vec{x_{j_1}}, \vec{x_{j_2}})$
    if $c > \epsilon$ **then**
      $\vec{v} = normalize(\vec{x_{j_1}} - \vec{x_{j_2}})$
      $\Delta x_{j_1} = \Delta x_{j_1} + c\,\vec{v}\,\Delta t$
    end if
  end for
  $x = x + \Delta x$
until $\forall i \in \{1, \ldots, n\}.\ \text{dist}(\mathbf{0}, \Delta x_i) \leq \epsilon\,\Delta t$
$T' = T$ updated with positions x
if $T' \in \mathcal{T}_{wf}$ **then return** T' **else return** $\perp$ **end if**

the sub-matrix of M containing the only rows of M with indices in I. Moreover, let $\overline{I} = \{1, \ldots, n\} \setminus I$ denote the complement of I. A *vertex* of a convex polygon $P = \{\ x \mid Ax \leq b\ \}$ is a point $v \in \mathbb{R}^2$ such that

$$\exists I \subseteq \{1, \ldots, n\}.\ |I| = 2 \wedge rank(A_I) = 2 \wedge A_I v = b_I \wedge A_{\overline{I}} v \leq b_{\overline{I}}$$

This means that v is a point of P which corresponds to the intersection of the two lines $A_{\{i_1\}}x = b_{\{i_1\}}$ and $A_{\{i_2\}}x = b_{\{i_2\}}$ indicated by the indices in $I = \{i_1, i_2\}$. The fact that $rank(A_I) = 2$ ensures that the intersection point v exists and is unique.

A convex polygon (which is finite) can also be described as the *convex hull* of its set of vertices, where the convex hull of a finite set of points $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^2$ is the minimum convex set containing all the points specified. Formally, it can be defined as:

$$CH(X) = \left\{ y = \sum_{i=1}^{n} \lambda_i x_i \;\middle|\; \sum_{i=1}^{n} \lambda_i = 1,\ \lambda_i \geq 0\ i = 1, \ldots, n \right\}$$

In Figure 6.4a, the convex hull of the points $x_1, \ldots, x_6$ is represented by the entire area contained inside the dashed curve.

Definition 6.4.1. Given a convex polygon $P = \{\ x \mid Ax \leq b\ \}$, the *cone of movement* of a vertex v corresponding to the row indices I of A is defined as follows:

$$CM(v) = \{\ x \mid A_I x \geq b_I\ \}$$

Figure 6.4b shows the vertices $v_1, \ldots, v_4$ of the convex hull of figure 6.4a, and their corresponding cones of movement, depicted by the shaded areas.

Definition 6.4.2. Given a convex polygon P with vertices $v_1, \ldots, v_k$, its *perimeter* is given by the sum of the distances between each pair of adjacent vertices. We denote the perimeter by $\ell(P)$.

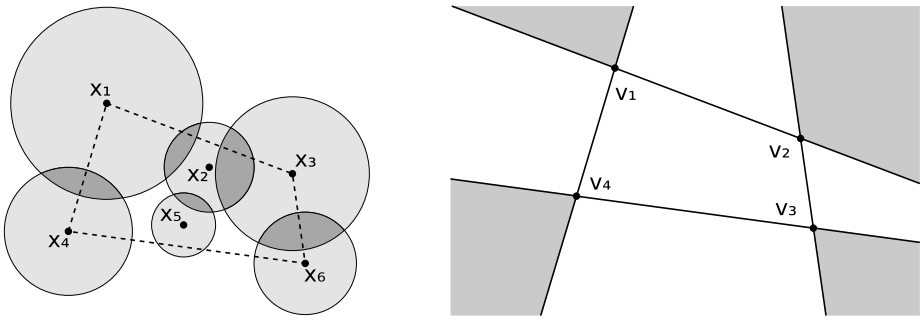


Figure 6.4: A polygon corresponding to a connected set of components, and the cones of movement of the vertices of the polygon.

Algorithm 13 is applied to the initial set of centers of the elements $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^2$, whose radii are $\{r_1, \ldots, r_n\} \subset \mathbb{R}$. Let k denote the step number of the outer loop of the algorithm (lines 2–12). We denote by $X^{(k)} = x_1^{(k)}, \ldots, x_n^{(k)}$ the centers of the elements at the beginning of step k.

Definition 6.4.3. Let $X^{(k)} = x_1^{(k)}, \ldots, x_n^{(k)}$ be the centers of the elements at step k, whose radii are $r_1, \ldots, r_n$. The overlap between any pair of elements i, j at iteration k is defined as:

$$c_{ij}^{(k)} = \begin{cases} r_i + r_j - \text{dist}(x_i^{(k)}, x_j^{(k)}) & \text{if } r_i + r_j - \text{dist}(x_i^{(k)}, x_j^{(k)}) > \epsilon; \\ 0 & \text{otherwise.} \end{cases}$$

The following definition formally defines *connected sets*, which are the smallest sets of elements in which every element of P overlaps with at least another element in the same set and without any element external to the set.

Definition 6.4.4. Let $G = (N, A)$ be an undirected graph, whose nodes $N = \{1, \ldots, n\}$ correspond to element indices, and the set of arcs is such that $(i, j) \in A$ iff $c_{ij} > 0$. A set of elements $P \subseteq N$ is a called *connected* iff it is a connected component of the graph and $|P| \geq 2$.

We denote by $CH(P, X)$ the convex hull of the elements specified by $P \subseteq \{1, \ldots, n\}$ whose positions are specified by $X = \{x_1, \ldots, x_n\}$, i.e. $CH(P, X) = CH(\{\, x_i \mid i \in P \,\})$.

For any step k of the algorithm, we let $P_1^{(k)}, \ldots, P_m^{(k)}$ denote the sets of indices of connected elements. Note that the sets $P_1^{(k)}, \ldots, P_m^{(k)}$ represent a partition of a subset of indices of elements $\{1, \ldots, n\}$.

Lemma 6.4.1. *Given a connected set of elements $P^{(k)}$ with positions $X^{(k)}$, let $C = CH(P^{(k)}, X^{(k)})$. Each vertex of C, which is also contained in $X^{(k)}$, is moved inside its cone of movement. Formally, for each vertex $x_i^{(k)} \in X^{(k)}$ of C:*

$$x_i^{(k+1)} \in CM(x_i^{(k)}) \text{ and } x_i^{(k+1)} \neq x_i^k$$

Proof. In order to prove that each vertex $x_i^{(k)}$ of C is moved inside its cone of movement $CM(x_i^{(k)})$ we have to show that Δx_i is directed towards such a cone. Δx_i corresponds to

the sum of the $c\,\vec{v}\Delta t$ vectors computed at each iteration of the inner loop of the algorithm. Values c and Δt are positive, hence they do not affect the direction of the vector. Vector $\vec{v}$ lies on the line that connects x_i with another point $x_j \in C$ and is directed backward with respect to x_j. Since x_j is contained in C it follows that $\vec{v}$ is directed towards the cone of movement $CM(x_i)$. Since this holds for all vectors $\vec{v}$ computed in the inner loop of the algorithm, we have that Δx_i is directed towards $CM(x_i)$. The property $x_i^{(k+1)} \neq x_i^{(k)}$ holds because: (i) $\Delta t > 0$, (ii) in all iterations of the inner loop $\vec{v}$ is a unit vector, and (iii) there is at least an iteration in which $c > \epsilon$ (because the vertex is part of a connected component). □

The following lemma shows that the perimeter of each polygon corresponding to each connected set of elements increases at each iteration of the outer loop of the algorithm.

Lemma 6.4.2. *Let $P_1^{(k)}, \ldots, P_m^{(k)}$ represent the connected sets of elements at a generic iteration k of the outer loop of the algorithm, and $X^{(k)} = x_1^{(k)}, \ldots, x_n^{(k)}$ their positions.*

$$\forall i = 1, \ldots, m. \quad \ell(CH(P_i^{(k)}, X^{(k)})) < \ell(CH(P_i^{(k)}, X^{(k+1)}))$$

Proof. Lemma 6.4.1 implies that $CH(P_i^{(k)}, X^{(k)}) \subset CH(P_i^{(k)}, X^{(k+1)})$ for all $i = 1, \ldots, m$, therefore their perimeters are increased. □

Finally, the following theorem proves that the restricted version of the Arrange algorithm, defined as Algorithm 13, always terminates.

Theorem 6.4.3. *Algorithm 13 always terminates.*

Proof. Suppose that the algorithm does not terminate. There exists a subset Z of elements, containing at least two elements, where each element belongs infinitely many times to some connected component. Moreover, there exists a step $\overline{k}$ from which, for all subsequent steps, the only elements moving are those of Z. Let denote by $CH(Z)$ the convex hull of the elements in Z at some step $k' \geq \overline{k}$. Each vertex of $CH(Z)$ is either a vertex of a connected component inside Z, or it will become such in some subsequent step. Therefore, each vertex will eventually move inside the cone of movement of a connected component, which is contained in the cone of movement of $CH(Z)$, hence the perimeter of $CH(Z)$ increases.

In particular, the perimeter of $CH(Z)$ tends towards infinity, since the amount of movement of a vertex is at least $\epsilon\,\Delta t$. Therefore, the distance among vertices continuously increases, and this suggests that the elements may reach a configuration in which they are partitioned into subsets that are independent from each other, namely an element of a subset never forms a connected component with any other element from the other subsets.

Suppose that the set of elements are partitioned in two independent subsets Z', Z'', such that the elements from a subset Z', from a certain step, do never form a connected component with any element from the other subset Z''. Therefore, we can apply the same argument using $CH(Z')$ and $CH(Z'')$. Since the set Z is finite, this procedure will eventually terminate with all the elements disconnected, that is a situation in which the algorithm terminates.

On the other hand, suppose that there exists a set of objects $Z''' \subset Z$ which cannot be partitioned in independent subsets, and that the perimeter of $CH(Z''')$ increases towards infinity. From a certain step the distance among vertices of Z''' will become too big for any

element to be able to cross it. In order to prove this, let us consider a complete weighted graph $G(Z''')$ whose nodes are connected components in Z''' and arcs have the distance between the corresponding connected components as weights. Since the perimeter of $CH(Z''')$ increases towards infinity, for every $w \in \mathbb{R}^+$ there exists a step of the algorithm in which there is a cut in $G(Z''')$ such that all the arcs crossing the cut have a weight greater than w. This means that Z''' can be partitioned into two subsets Z_1''' and Z_2''' arbitrarily far from each other.

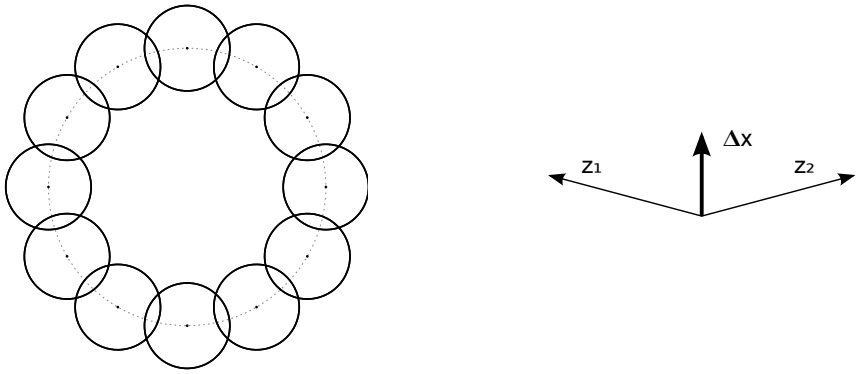
Both $CH(Z_1''')$ and $CH(Z_2''')$ continuously increase in size. Let us consider one of the two subsets, say Z_1'''. We can repeat the procedure and find a value $w' \in \mathbb{R}^+$ such that there exists a step with a cut in $G(Z_1''')$ such that all the arcs crossing it have a weight greater than w'. Since Z''' is finite, we can continue with this procedure until we reach a subset Z_i''' consisting of a single connected component, and such that its distance from every other component is greater than some arbitrary value $w_i \in \mathbb{R}^+$. Since we are assuming that Z''' cannot be partitioned in two independent subsets, eventually a collision has to occur between an element of Z_i''' and one of its complement. In other words, either some elements of Z_i''' or some elements of its complement (or both) have to cover at least the distance w_i. For sure w_i can be chosen big enough such that the elements of Z_i''' cannot cover it. In fact, Z_i''' consists of a single connected component and, since Lemma 6.4.1 implies that $CH(Z_i'''^{(k)}) \subseteq CH(Z_i'''^{(k')})$ with $k' > k$, we have that its elements cannot cover an arbitrary long distance without disconnecting.

Once disconnected, the elements of Z_i''' are still rather close to each other since the length of the moves that disconnects them has as an upper bound the overlaps at the previous step. As a consequence, after disconnection we are in a situation in which the elements of Z_i''' are still at an arbitrary distance from the elements in its complement and can only try to cover such a distance by means of further disconnection. However, since Z_i''' is finite, we have that the number of disconnections is finite as well.

This proves that the elements Z_i''' cannot cover the distance w_i. In order to prove that also those in the complement of Z_i''' cannot, we can iterate the approach by constructing $G(\overline{Z_i'''})$, where $\overline{Z_i'''}$ is the complement of Z_i''', by finding a cut in such a graph similar to the previous one and by continuing until we reach a situation in which all the connected components are far enough from each other. This contradicts the assumption that Z''' cannot be partitioned in independent subsets. □

Note that there are cases in which the Arrange algorithm terminates without resolving all the conflicts. This happens, for instance, in a system like the one represented in Figure 6.5a, in which a number of elements of the same size are conceptually positioned along a circumference, and in which the distance between each pair of adjacent elements is constant. Figure 6.5b shows the movement vector Δx to which an element is subject to, obtained from the sum of the two vectors z_1 and z_2 caused by the collisions with adjacent elements. For every value of the threshold ϵ, it is possible to construct a system of that kind, in which the algorithm terminates without resolving all the conflicts. It can be obtained by choosing the right number of elements and the radii of elements and circumference, in such a way that the amount of each collision is greater than $\epsilon\,\Delta t$, while the total movement vector Δx_i for every element is such that $\Delta x_i < \epsilon\,\Delta t$. Hence the algorithm terminates even if there are still collisions among the elements.

The proof of termination of the complete algorithm (Algorithm 11) could be given as an extension of the proof of Theorem 6.4.3. In particular, in the case of a term with looping



(a) (b)

Figure 6.5: (a) A situation in which the Arrange algorithm might terminate in a non-well formed arrangement; (b) the movement vector of the element on the top (not in scale).

sequences Theorem 6.4.3 could be exploited to prove that the movement of the top-level elements of the term eventually terminates. As regards the elements that are contained in some looping sequence we have that Lemma 6.4.2 may not hold. In fact, it might happen that some elements contained in a looping sequence are moved by the algorithm in such a way that they exceed the bounds of the looping sequence. These elements are then moved back by the algorithm possibly causing the perimeters of the connected components containing them to be reduced. However, the fact that $\Delta t < 1$ implies that these backward movements are smaller than the movements causing the bounds to be exceeded. This should allow to prove that the termination condition of the algorithm $\forall i.\ \mathrm{dist}(\mathbf{0}, \Delta x_i) \leq \epsilon\, \Delta t$ is eventually reached, even though it may not correspond to a well-formed arrangement of the elements.

6.5 Examples of modelling

In this section we show examples of using the Spatial CLS to model biological systems. First of all, we show the definition of a movement function which realizes Brownian motion. This movement function is then used in the subsequent Spatial CLS models, one describing cell proliferation and another one describing the quorum sensing process.

6.5.1 Describing movement

The movement function associated with elements allows the precise description of their motion as time passes. The first example of movement function, which is often needed in models, is the one associated with the elements that are not moving. We denote this function by the name $n_0 \in \mathcal{M}$, and define it as a movement function always giving the same position p passed as argument with probability 1, and ignoring the parameter υ representing the internal state. Formally, this corresponds to the function:

$$n_{0\mathrm{fun}}(p, r, x, l, t, \delta t, \upsilon) = (p, \Pi, \upsilon)$$

where Π is such that $\Pi(p) = 1$. This function can be used by assigning spatial information $d = ([p, m_0], r)$ to an object, where $m_0 = (n_0, \perp)$.

Algorithm 14 $BrownianMotion(p, r, x, l, t, \delta t, \upsilon) = (P, \Pi, \upsilon')$

if $\upsilon = \perp$ **then**
  // initialization
  $\upsilon_0 = (t, p)$
  $(P, \Pi, \upsilon') = BrownianMotion(p, r, x, l, t, \delta t, \upsilon_0)$
else
  let $(t_0, p_{prev}) = \upsilon$
  $k = \left\lceil \frac{t - t_0}{\rho} \right\rceil$
  $\vec{v} = s \, \frac{p - p_{prev}}{\mathrm{dist}(p, p_{prev})}$
  $\Delta t_1 = t_0 + k\rho - t$
  $q = p + \vec{v}\,\Delta t_1$
  if $t + \delta t > t_0 + k\rho$ **then**
    // a change of direction occurs between time t and $t + \delta t$
    $\Delta t_2 = t_0 + k\rho - (t + \delta t)$
    $P = \{q + x \mid x \in Positions(z, s\,\Delta t_2)\}$
    Π is such that $\forall x \in P.\ \Pi(x) = 1/z$
    $\upsilon' = (t_0, q)$
  else
    $P = \{q\}$
    Π is such that $\Pi(q) = 1$
    $\upsilon' = \upsilon$
  end if
end if

A more interesting function, which is still deterministic, is the one modelling a linear motion. Given a vector $\vec{v}$ describing the velocity, this movement function can be defined as follows:

$$m_{\mathrm{fun}}(p, r, x, l, t, \delta t, \upsilon) = (p + \vec{v}\,\delta t, \Pi, \upsilon)$$

where, as before, Π gives probability 1 to the only possible resulting position.

Another useful function is the one describing Brownian motion. A simple implementation can be obtained by performing a linear motion, and by repeatedly changing direction after a fixed time interval. The change of direction is randomly chosen, with uniform probability, among a finite number of equally-spaced directions. Let $\rho \in \mathbb{R}^+$ be the length of the time interval after which there is a change of direction, and z be the number of directions allowed at each change. By denoting the creation time of the object as t_0, we perform a change of direction at each time instant $t_0 + k\,\rho$, for each $k \in \mathbb{N}$. The change of direction corresponds to assigning probability $1/z$ to each angle in the set $\{2\pi x/z \mid 0 \leq x < z\}$. The distance covered between each change of direction depends on the speed of the object, described by parameter s.

Algorithm 14 shows the definition of movement function modelling Brownian motion using pseudo code. In the definition, we assume that the time interval ρ (of changing direction) is always greater than or equal to the actual step length δt. This is ensured by taking $\rho \geq 1/N$. Moreover, by $Positions(z, d)$ we denote the set of z equally-spaced positions, at distance d from the origin; formally: $Positions(z, d) =$

$\{(d\cos\alpha, d\sin\alpha) \mid \alpha = 2\pi x/z,\ 0 \leq x < z\}$.

A special value $\perp$ for the state argument υ of the movement function is used to initialize the state argument. In particular, state argument $\upsilon = (t_0, p)$ is a pair composed of the time instant t_0 when the movement begins, and the position $p \in \mathbb{R}^n$ in which the last change of direction has occurred. The value $\perp$ should be used as the initial value of state parameter for movement function. In the following, an object with Brownian motion is described by a term having spatial information $d = ([p, m_B], r)$, where $m_B = (BrownianMotion, \perp)$.

Finally, note that the presented implementation of Brownian motion is quite simple, since its main aim is to show how a probabilistic movement function can be formally defined. More faithful implementations could be used if needed.

6.5.2 A model of cell proliferation

We show a very simple model of the development of a biological tissue. We represent the way in which a cell performs the mitosis cycle, and the development of the tissue as a consequence of it. The structure of a cell is made up of a permeable cell membrane, which separates it from external environment but still allows messages to pass through, and contains several organelles scattered in the cytoplasm. We model a simple eukaryotic cell as a membrane containing the nucleus, which, in turn, contains two DNA molecules (the chromosomes). Each cell performs the *cell cycle* [3], that is the sequence of phases that lead to its division into two daughter cells, structurally alike to the mother cell. Customarily, cell cycle repeats for every generated cell, but, in particular cases, the cell may decide to stop the process in a permanent or temporary way (for instance, in case of unfavorable ambient conditions).

In this example we are interested only in observing the way in which the cells, described by our model, fill the environment during the mitosis process. Thus the model is not realistic, for example cell apoptosis (cell death) is not taken into account.

The initial state of the biological system is described by the following term[2]:

$$T = \big(b\big)^L_{\cdot,50} \rfloor \big(m\big)^L_{[(0,0),m_B],10} \rfloor \big(n\big)^L \rfloor (cr \cdot g_1 \cdot g_2 \cdot g_3 \mid cr \cdot g_4 \cdot g_5)$$

The term contains the looping sequence $\big(b\big)^L_{\cdot,50}$, representing the space available for the proliferation as a circle with a $50\mu m$ radius. It contains a single cell $\big(m\big)^L$, positioned in $(0, 0)$ and with a radius of $10\mu m$. The cell is subjected to a small Brownian motion, modelled by $m_B = (BrownianMotion, \perp)$, where $BrownianMotion$ denotes an instantiation of Algorithm 14 defined in Section 6.5.1. The nucleus, represented as $\big(n\big)^L$, and the contained chromosomes, are represented as non-positional elements. The nucleus may be in two states, depending on the symbol appearing on its looping sequence. Initially, the nucleus is identified by the symbol n appearing on the surface of its membrane. During the evolution of the system, the symbol n is replaced by n_{dup}, indicating a state in which the nucleus has started the duplication process and is about to divide. Chromosomes are modelled as sequences starting with cr symbol, followed by the genes, represented by the symbols g_i's. A duplicated chromosome is identified by having $2cr$ as its first symbol in the sequence.

[2]For the sake of clarity, we omit the spatial information for non-positional elements, i.e. those elements $\big(\_\big)_{\langle q,r\rangle}$ such that $\langle q, r\rangle = \langle \cdot, 0\rangle$.

The Brownian motion of cells represents the small movements the cells are subjected to in a biological tissue. The use of Spatial CLS to represent the cell cycle allows showing the spatial arrangement of cells during the tissue development. This is an important point in many biological tissues, see for example [38] for a spatial mathematical modelling of a neural tissue.

The evolution of the system is modelled by the following rewrite rules[3]:

$$R_1 : [\, r = 7 \,] \quad \big(m\big)^L_{[p,f],r} \rfloor X \overset{0.33}{\mapsto} \big(m\big)^L_{[p,f],10} \rfloor X$$

$$R_2 : [\, r = 10 \,] \quad \big(m\big)^L_{[p,f],r} \rfloor X \overset{0.25}{\mapsto} \big(m\big)^L_{[p,f],14} \rfloor X$$

$$R_3 : [\, r = 14 \,] \quad \big(m\big)^L_{[p,f],r} \rfloor \Big(\big(n\big)^L \rfloor X\Big) \overset{0.5}{\mapsto} \big(m\big)^L_{[p,f],r} \rfloor \Big(\big(n_{\mathrm{dup}}\big)^L \rfloor X\Big)$$

$$R_4 : \big(n_{\mathrm{dup}}\big)^L \rfloor (cr \cdot \widetilde{x} \mid X) \overset{0.125}{\mapsto} \big(n_{\mathrm{dup}}\big)^L \rfloor (2cr \cdot \widetilde{x} \mid X)$$

$$R_5 : \big(n_{\mathrm{dup}}\big)^L \rfloor (2cr \cdot \widetilde{x} \mid 2cr \cdot \widetilde{y}) \overset{0.17}{\mapsto} \big(n\big)^L \rfloor (cr \cdot \widetilde{x} \mid cr \cdot \widetilde{y}) \mid \big(n\big)^L \rfloor (cr \cdot \widetilde{x} \mid cr \cdot \widetilde{y})$$

$$R_6 : \big(m\big)^L_{[(x,y),f],r} \rfloor \Big(\big(n\big)^L \rfloor X \mid \big(n\big)^L \rfloor Y\Big) \overset{1}{\mapsto}$$
$$\big(m\big)^L_{[(x-5,y),f],7} \rfloor \big(n\big)^L \rfloor X \mid \big(m\big)^L_{[(x+5,y),f],7} \rfloor \big(n\big)^L \rfloor Y$$

The first three rules describe the growth of the cell. Rule R_1 increases the radius from 7 to 10, leading a just-split cell to its normal size. Rule R_2 represents the starting of the division process, where the cell grows to 14 and will eventually divide. A cell may block its cell cycle if there is not enough space: this happens when neither R_1 nor R_2 are applicable to it. The application of rule R_3 signals the start of the division process for the nucleus. Rule R_4 models the duplication of a single chromosome. Finally, rule R_5 and R_6 describe the division of the nucleus and the subsequent cellular division. In rule R_6, the split cells are arbitrarily positioned one next to the other, near the position of the parent cell. The rates have been estimated according to the common relative lengths of the phases forming the cell cycle, and so as to obtain, on the average, a duration of 24 hours for the complete cycle [3].

Figure 6.6 shows the state of the system at certain times during the simulation, obtained by an ad hoc simulator. At time $t = 0$ the system contains only one cell, positioned inside the limiting membrane. The proliferation stops at time $t = 141h$, when the space left is not enough for any cell to grow. We can also see that, from time $102h$ to $108h$, a cell near the center has split into two small cells, and another cell on the bottom right has grown, thus initiating the division process. By growing, the cell pushed the surrounding cells and caused the rearrangement.

6.5.3 A model of the quorum sensing process

Many bacteria have the ability of monitoring their population density and modulating their gene expressions according to this density. This process is called *quorum sensing*, and the main entities involved in it are the *autoinducers*, small molecules that can cross the cellular membrane and can diffuse freely either out or in bacteria, and the *R-proteins*, transcriptional activator proteins located within the cell.

[3]For the sake of readability, we use a simpler syntax for writing rewrite rules: in the left part of the rules, we use placeholders for positions, movement functions and radii, and reuse them in the right part in a shorthand notation for defining instantiation functions for position variables.

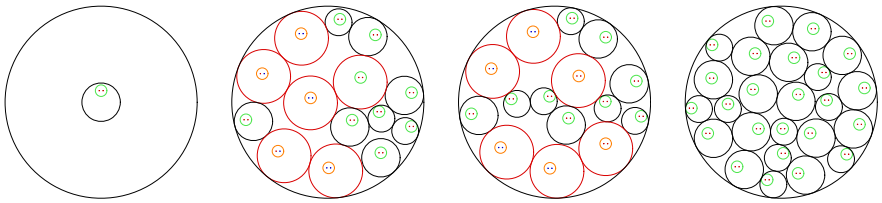


Figure 6.6: The graphical representation of the system at times 0, $102h$, $108h$ and $141h$ during simulation.

The production of the autoinducer is regulated by the R-protein. The R-protein by itself is not active without the corresponding autoinducer. The autoinducer molecule can bind to the R-protein to form an autoinducer/R-protein complex, which binds to a target of the DNA sequence enhancing the transcription of specific genes. Usually, these genes regulate both the production of specific behavioural traits and the production of the autoinducer and of the R-protein.

At low cell density, the autoinducer is synthesized at basal levels and diffuse in the environment where it is diluted. With high cell density both the extracellular and intracellular concentrations of the autoinducer increase until they reach thresholds beyond which the autoinducer is produced autocatalytically. This autocatalytic production results in a dramatic increase of its concentration. Note that there is not a fixed concentration of the autoinducer that, when it is reached, causes the autocatalytic production to start. Instead, having a stochastic model means that there is a low probability of starting the autocatalytic production at low concentrations, while this probability increases as the concentration gets higher. Moreover, the autocatalytic production continues as long as autoinducer concentration remains high enough, which happens only when bacteria density is sufficient. With low bacteria density, when an autocatalytic production starts, it usually ends quite soon, as the autoinducers diffuse in the environment and, consequently, their concentration lowers.

We show a simple model of the quorum sensing process in *Pseudomonas aeruginosa* (see [35] for a more detailed description of the phenomenon). Such a bacterium uses quorum sensing to keep low the expression of virulence factors until the colony has reached a certain density, when an autoinduced production of virulence factors is started. The initial state of each bacterium is:

$$Bact_i = \big(m\big)^L_{[p_i,m_0],r_{\mathrm{BACT}}} \rfloor (lasO \cdot lasR \cdot lasI)$$

where the bacterium membrane, denoted $\big(m\big)^L$, contains only a DNA strand. Bacteria do not move, hence their movement function is $m_0 = (n_0, \perp)$ as defined in Section 6.5.1, and their radii are represented by r_{BACT}. The DNA is modelled as a sequence of genes $lasO \cdot lasR \cdot lasI$, where $lasO$ represents the target to which a complex autoinducer/R-protein binds to promote transcription. The following rewrite rules model the system behaviour.

$$R_1 : lasO \cdot lasR \cdot lasI \overset{k_1}{\mapsto} lasO \cdot lasR \cdot lasI \mid LasR$$

$$R_2 : lasO \cdot lasR \cdot lasI \overset{k_2}{\mapsto} lasO \cdot lasR \cdot lasI \mid LasI$$

$R_3 : LasI \overset{k_3}{\mapsto} LasI \mid 3oxo$

$R_4 : 3oxo \mid LasR \overset{k_4}{\mapsto} 3R$

$R_5 : 3R \overset{k_5}{\mapsto} 3oxo \mid LasR$

$R_6 : 3R \mid lasO \cdot lasR \cdot lasI \overset{k_6}{\mapsto} 3RO \cdot lasR \cdot lasI$

$R_7 : 3RO \cdot lasR \cdot lasI \overset{k_7}{\mapsto} 3R \mid lasO \cdot lasR \cdot lasI$

$R_8 : 3RO \cdot lasR \cdot lasI \overset{k_8}{\mapsto} 3RO \cdot lasR \cdot lasI \mid LasR$

$R_9 : 3RO \cdot lasR \cdot lasI \overset{k_9}{\mapsto} 3RO \cdot lasR \cdot lasI \mid LasI$

$R_{10} : \left(m\right)^L_{[p,f],r} \rfloor (3oxo \mid X) \overset{k_{10}/z}{\mapsto} \left(3oxo\right)_{[p+(r+d_{OUT})q,m_B],r_{3oxo}} \mid \left(m\right)^L_{[p,f],r} \rfloor X$
with $q \in Positions(z, 1)$

$R_{11} : [\, \mathrm{dist}(p_1, p_2) \leq r_2 + d_{\mathrm{INT}} \,]$
$\left(3oxo\right)_{[p_1,m_B],r_{3oxo}} \mid \left(m\right)^L_{[p_2,f],r_2} \rfloor X \overset{k_{11}}{\mapsto} \left(m\right)^L_{[p_2,f],r_2} \rfloor (3oxo \mid X)$

$R_{12} : LasI \overset{k_{12}}{\mapsto} \lambda$

$R_{13} : LasR \overset{k_{13}}{\mapsto} \lambda$

$R_{14} : \left(3oxo\right)_u \overset{k_{14}}{\mapsto} \lambda$

Rules R_1 and R_2 describe the production from the DNA of proteins $LasR$ and $LasI$, respectively. Note that $lasR/lasI$ (with small case initial letter) denote a gene in the DNA strand, while $LasR/LasI$ (with capital initial letter) denote the corresponding proteins originated by the gene. Rule R_3 describes the production of the autoinducer, modelled as $3oxo$, performed by the LasI enzyme. Rules R_4 and R_5 describe the complexation and decomplexation of the autoinducer and the $LasR$ protein, where the complex is denoted $3R$. Rules R_7, R_8, R_9 describe the binding of the activated autoinducer (the $3R$ complex) to the DNA and its influence in the production of $LasR$ and $LasI$. Rules R_{10} and R_{11} describe the ability of the autoinducer to cross the membrane, in both directions (exiting and entering the bacterium). Actually, R_{10} is a rule schemata, in which each concrete rule puts the autoinducer in a different position outside the bacterium, as a positional element. The possible outside positions are $p+(r+d_{OUT})q$, with respect to the bacterium position p and radius r, and where $q \in Positions(z, 1)$. (Function $Positions$ is defined in Section 6.5.1.) Therefore, the possible resulting positions are the z equally-spaced position at distance d_{OUT} from the bacterium. Value z is chosen big enough to provide sufficient variability. Note that the rate of each concrete rule is $1/z$ of the expected rate k_{10}.

An autoinducer inside the bacterium is modelled as a non-positional element, while autoinducers outside have an associated position and movement function m_B. Parameter r_{3oxo} denotes the radius of each external autoinducer. The parameter $m_B = (n_B, \perp)$ represents a realization of Brownian motion (defined as in Section 6.5.1), which describes the diffusion of the autoinducer in the environment. In this case, we use a value $\rho = 1/N$ time units describing the time interval between each random change of direction. Moreover, we denote by s_{oxo} the speed of the autoinducer.

Finally, rules R_{12}, R_{13}, R_{14} describe the degradation of proteins. In particular, rule R_{14} models the degradation of the autoinducer, which can happen both inside and outside

the bacterium.

This Spatial CLS model of the quorum sensing process is more accurate than other stochastic models (such as the Stochastic CLS model given in [4]). In fact, other stochastic models are usually based on the assumption that biological entities are homogeneously distributed in the environment (*well-stirred* assumption), and in a quorum sensing process this is not true for the autoinducer proteins outside the bacteria. Note that taking into account the spatial diffusion has a particular significance when reactions are comparatively faster than diffusion rates [84].

To our knowledge, there are no wet or in vitro experiments, presented in the literature, from which it is possible to derive precise quantities (such as speed of autoinducer molecules, kinetic constants of the reactions, distance of bacteria) for our model. Interestingly, we can use the results of wet experiments for inferring approximations of such quantities. Given an experiment with an initial number of bacteria, we adapt the constant values in our model, in order to have the same final results of the real experiments. The obtained constants can be used as an approximation either for constructing new model or to have hints for predicting the behaviour of the real phenomena.

For instantiating our general model we use the experiment, presented in [80], in which the cellular toxicity of three *Pseudomonas aeruginosa* strains, PA103, PA01 and 6294, is investigated. The experiment consists in culturing a human bronchial epithelial cell line, then mixing 2×10^4 cells with inocula of *Pseudomonas aeruginosa* at different concentrations. After six hours the cytotoxicity of the inocula was stated by measuring the production of lactate dehydrogenase (LDH). A high level of LDH corresponds to a high cytotoxicity which reveals that the quorum sensing process occurred.

On the basis of the results in [80], we tried to infer values for our model able to reproduce the behaviour of *Pseudomonas aeruginosa* 6294 strain. In the real experiment the human bronchial epithelial cells are mixed with inocula of the 6294 strain at three different concentrations: 10^5, 10^7 and 10^9 colony forming units, CFU, per milliliter. A CFU is a bacterium able to divide and to form a colony, thus CFU is a measure of "good bacteria". The results for the 6294 strain are summarized in Figure 1 in [80]: with concentrations of 10^5 and 10^7 CFU/ml no cytotoxicity is expressed, while with a 10^9 CFU/ml concentration a high cytotoxicity is measured after six hours, revealing that the quorum sensing process occurred.

In our model we consider three different concentrations in which the number of bacteria is, respectively, 1, 10 and 100 for space unit. Recall that Spatial CLS deals with plane surfaces, thus these numbers roughly respect the orders of magnitude in the real volume concentrations. We assume a space unit of $300\mu m^2$ (the dimension of a single bacterium is nearly $1\mu m^2$). Actually, for computational reasons, instead of running simulations with 100 bacteria in a space of $300\mu m^2$, we ran simulations with 40 bacteria in a space of $120\mu m^2$. We performed various simulations and we found that the real experiment can be approximated by using the values shown in Figure 6.7.

As regards the implementation of the simulator, we have followed a more abstract approach than the one used in the semantics of the calculus. In particular, since bacteria contain only non positional elements, it is possible to simulate the internal evolution of each bacterium by using the Gillespie algorithm [40]. This allows us to easily compute the time and kind of the subsequent internal event of a bacterium. The simulator performs a sequence of steps of length at most $1/N$ for dealing with the spatial movement of external autoinducers. This sequence of steps is interleaved with the execution of the internal

Parameter	Value
Bacterium radius (r_{BACT})	5
Bact. interaction distance (d_{INT})	0.5
Autoinducer exiting distance (d_{OUT})	0.5
N	630
Autoinducer speed (s_{oxo})	1575
Autoinducer radius (r_{3oxo})	0.01

Param.	Value
k_1	126
k_2	31.5
k_3	50.4
k_4	1.575
k_5	2520
k_6	1.575
k_7	63

Param.	Value
k_8	7560
k_9	1890
k_{10}	189
k_{11}	126
k_{12}	6.3
k_{13}	1.89
k_{14}	1.89

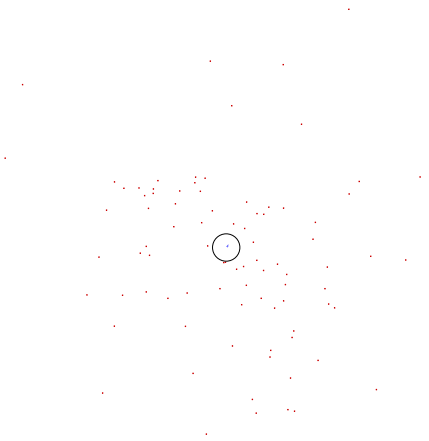
Figure 6.7: Values of parameters used for the simulation of the Quorum Sensing process.

bacteria events, which are ensured to occur at the right time by performing, if needed, a step shorter than $1/N$. If at any time an autoinducer happens to enter a bacterium (as per rule R_{11}), then the next scheduled internal event for the bacterium is removed and a new one is computed, according to Gillespie algorithm. This is needed to account for the change in the internal state of the bacterium.

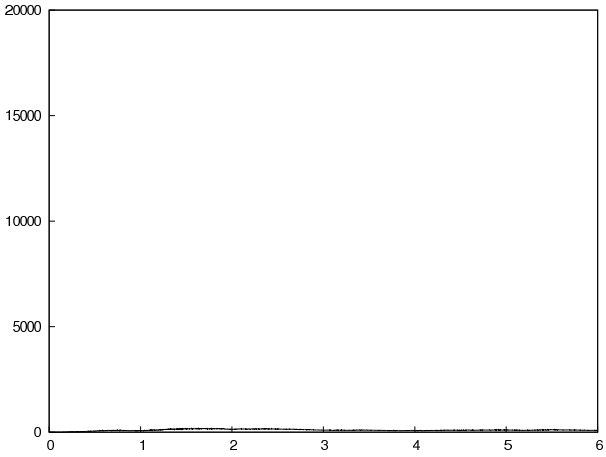
In Figure 6.8a a single bacterium in a space unit is shown. In this case only few molecules of the autoinducer can reach other bacteria. Thus the bacterium is not able to sense the presence of other ones and the quorum sensing does not occur. In this case the production of the autoinducer remains at the basal level, as shown in Figure 6.8b.

In Figure 6.8c the production of the autoinducer by 10 bacteria is shown. With respect to the previous case it is possible to observe a bigger number of autoinducer molecules close to (and consequently inside) each bacterium. In some instant, this concentration is high enough to start the autocatalytic production of the autoinducer itself. However, such a autocatalytic production is not supported by the production of the autoinducer by other bacteria, thus the quorum sensing process is not triggered. Figure 6.8d shows the autoinducer production inside a single bacterium.

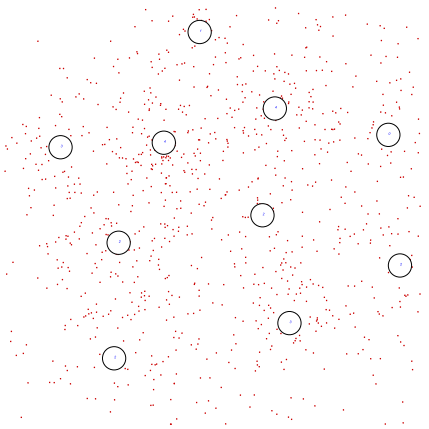
Finally, Figure 6.8e shows the behaviour of a colony of 40 bacteria in a space of $120\mu m^2$, corresponding to 40% of the space unit. In this case the autocatalytic production is supported by the high density of the autoinducer itself, and the quorum sensing behaviour occurs. Figure 6.8f shows the autoinducer production inside a single bacterium.



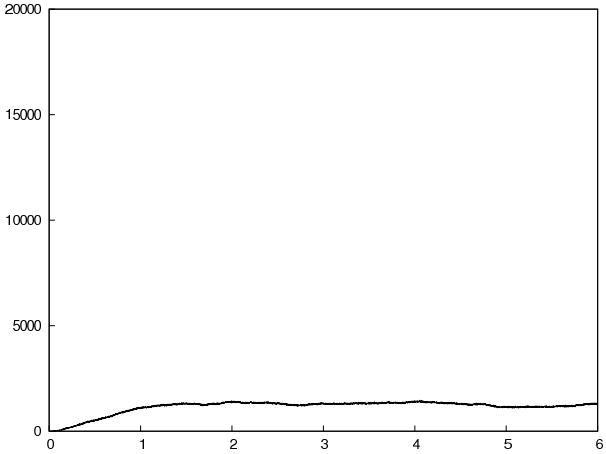
(a) 1 bacterium at time $5h$, in $300\mu m^2$ space



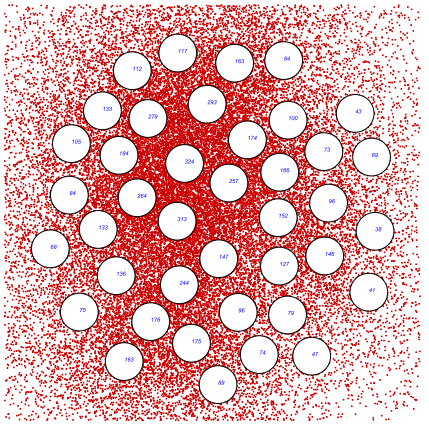
(b) Number of autoinducers with 1 bacterium



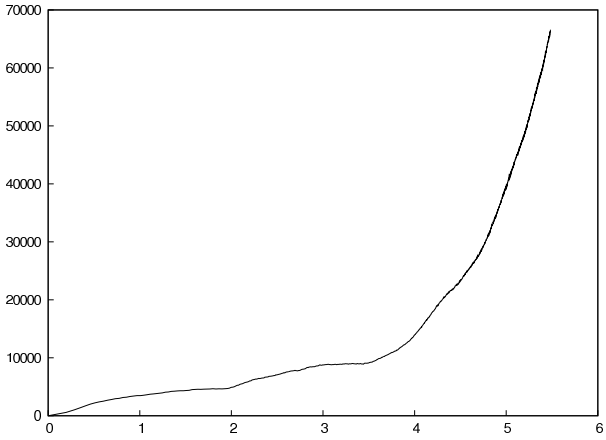
(c) 10 bacteria at time $5h$, in $300\mu m^2$ space



(d) Number of autoinducers with 10 bacteria



(e) 40 bacteria at time $5h$, in $120\mu m^2$ space



(f) Number of autoinducers with 40 bacteria

Figure 6.8: Graphical representation of bacteria and autoinducers, and the results of the different simulations.

Chapter 7

Conclusions

The use of formal methods for studying biological systems is an interesting approach that allows unambiguous descriptions of the processes, which can be used to perform simulations and other analyses. In this thesis we have extended with spatial features three modelling formalisms for the description of biological systems, in order to allow more precise descriptions of the behaviour of systems. Since each formalism is aimed at the modelling of different kinds of systems, the suitable abstraction of the real space is different in each case.

We have defined the MIM calculus as a process calculus providing high-level operators directly inspired by Molecular Interaction Maps (MIM), a graphical notation used in biology. The version for spatial modelling that we have developed, namely the *MIM calculus with compartments*, provides the most abstract form of space modelling among the extensions that we have developed. Since the formalism has been developed as a process calculus, it is possible to exploit common features of process calculi, such as the incremental definition of models, techniques for analysis and verification of properties, and the development of simulators. Moreover, the correspondence of the operators of the calculus with biological reactions allows an easy translation of Molecular Interaction Maps into terms of the MIM calculus.

We have studied conditions under which a term of the MIM calculus is a formal representation of a MIM diagram, by providing different consistency definitions for the terms of the MIM calculus. By means of an example, we have shown how the MIM calculus with compartments is particularly suited to the description of microbiological processes, such as the interactions that happen at the level of the cell. In fact, the ability to describe compartments in an abstract way allows a fine description of the positions of elements, sufficiently precise for this kind of systems. For example, is it possible to distinguish if a protein is attached to the internal or external surface of a membrane, or if is embedded in it. As future work, different properties of calculus, such as the expressiveness, could be investigated. Moreover, quantitative extensions of the calculus can be developed to allow for a better description of biological systems, and for the development of simulators.

We have also developed an extension of the computing formalism P systems, the *Spatial P systems*, which provides an explicit representation of space inside membranes, in the form of a two-dimensional discrete space. As in standard P systems, evolution rules are associated with membranes, while objects are associated with positions inside membranes. Evolution rules are extended to allow objects to be moved to different positions. Objects

belonging to the special kind of mutually exclusive objects are subject to the constraint that a position can contain at most one of such a kind.

On the one hand, we have studied the computational power of mutually-exclusive objects, and shown that mutually-exclusive objects, even when using only non-cooperating rules, are sufficient to obtain universality. As an example application of the formalism in the field of Ecology, we have developed a Spatial P systems model of the evolution of *ring species*, by which a species spreads in two different directions around a geographical barrier, which evolve independently in the two sides causing the generation of separate species which cannot interbreed. On the other hand, we have started investigating the problem of simulation of Spatial P system models. We have proposed algorithms for the efficient simulation of some restricted kinds of models, and shown how they can be used to model the schooling behaviour of herrings. The complete theoretical characterization of the problem of simulation of Spatial P systems is planned as a future work. It would also be interesting to study other simulation algorithms, such as parallel and approximate versions.

As regards the formalism, in order to improve the usefulness of models, various extensions of the Spatial P systems could be developed. A stochastic version would allow more faithfully descriptions of the behaviour of systems than the non-deterministic semantics, and also to pave the way for quantitative simulations (see for example [64, 28]). Another possible extension of Spatial P systems would be to three-dimensional space.

Finally, we have presented the Spatial Calculus of Looping Sequences (Spatial CLS), which extends the Calculus of Looping Sequences (CLS) by allowing spatial information to be associated with the structures of the calculus. The Spatial CLS formalism enables the accurate description of those biological processes whose behaviour depends on the exact position of the elements in a continuous 2D or 3D space. Spatial CLS allows defining models with different levels of abstraction, since it is possible to associate the spatial information with the only elements whose spatial position affects the behaviour of the system.

As example applications of the calculus, we have presented a model of cell proliferation and a model of the quorum sensing process in *Pseudomonas aeruginosa*. In both the examples the use of spatial information allows modelling aspects which cannot be appreciated otherwise. In the example of cell proliferation we observe the way in which cells arrange themselves during the mitosis and how they fill the space until no more growth or division can occur. In the example of quorum sensing, which is a kind of reaction-diffusion system, we can appreciate how the autoinducer molecules diffuse from bacteria, and are used to sense the local concentration of bacteria.

As for the other formalisms, having a formal semantics for describing the models enables the development of simulators whose behaviour is precisely defined, which is also particularly important for formal reasoning. In the case of Spatial CLS, however, the development of simulators can be quite challenging, as it needs to deal efficiently with the intrinsic complexity of spatial models of biological processes. As a future work, it would be interesting to develop efficient algorithms for the simulation of Spatial CLS models, for example by means of approximation techniques. Moreover the formal semantics allows the use of verification techniques, such as bisimulation [16]. The application of formal techniques to Spatial CLS could be subject of future works.

Bibliography

[1] O. Andrei, G. Ciobanu, and D. Lucanu. A rewriting logic framework for operational semantics of membrane systems. *Theoretical Computer Science*, 373(3):163–181, 2007.

[2] S. S. Andrews and D. Bray. Stochastic simulation of chemical reactions with spatial resolution and single molecule detail. *Physical Biology*, 1(3):137–151, 2004.

[3] C. J. Avers. *Molecular Cell Biology*. Addison-Wesley, 1986.

[4] R. Barbuti, G. Caravagna, A. Maggiolo-Schettini, P. Milazzo, and G. Pardini. The Calculus of Looping Sequences. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016 of *Lecture Notes in Computer Science*, pages 387–423. Springer Berlin / Heidelberg, 2008.

[5] R. Barbuti, D. Lepri, A. Maggiolo-Schettini, P. Milazzo, G. Pardini, and A. Rama. Simulation of Kohn's Molecular Interaction Maps through Translation into Stochastic CLS+. In A. Pnueli, I. Virbitskaite, and A. Voronkov, editors, *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 58–69. Springer Berlin / Heidelberg, 2010.

[6] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and G. Pardini. Spatial Calculus of Looping Sequences. *Electronic Notes in Theoretical Computer Science*, 229(1):21–39, 2009. Proceedings of the Second Workshop From Biology to Concurrency and Back (FBTC 2008).

[7] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and G. Pardini. Spatial Calculus of Looping Sequences. *Theoretical Computer Science*, 2011. In press (doi:10.1016/j.tcs.2011.01.020).

[8] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, G. Pardini, and A. Rama. A Process Calculus for Molecular Interaction Maps. In G. Ciobanu, editor, *Membrane Computing and Biologically Inspired Process Calculi (MeCBIC) 2009*, volume 11 of *Electronic Proceedings in Theoretical Computer Science*, pages 35–49, 2009.

[9] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, G. Pardini, and L. Tesei. Spatial P systems. *Natural Computing*, 10(1):3–16, 2011.

[10] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, P. Tiberi, and A. Troina. Stochastic Calculus of Looping Sequences for the Modelling and Simulation of Cellular Pathways. In C. Priami, editor, *Transactions on Computational Systems Biology IX*, volume 9 of *Lecture Notes in Computer Science*, pages 86–113. Springer Berlin / Heidelberg, 2008.

[11] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and S. Tini. A P Systems Flat Form Preserving Step-by-step Behaviour. *Fundamenta Informaticae*, 87(1):1–34, 2008.

[12] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and S. Tini. Compositional semantics and behavioral equivalences for P Systems. *Theoretical Computer Science*, 395(1):77–100, 2008.

[13] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and S. Tini. Compositional semantics of spiking neural P systems. *Journal of Logic and Algebraic Programming*, 79(6):304–316, 2010.

[14] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and S. Tini. An Overview on Operational Semantics in Membrane Computing. *International Journal of Foundations of Computer Science*, 22(1):119–131, 2011.

[15] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. A Calculus of Looping Sequences for Modelling Microbiological Systems. *Fundamenta Informaticae*, 72(1-3):21–35, 2006.

[16] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. Bisimulations in calculi modelling membranes. *Formal Aspects of Computing*, 20(4-5):351–377, 2008.

[17] F. Bernardini and M. Gheorghe. Population P Systems. *Journal of Universal Computer Science*, 10(5):509–539, 2004.

[18] D. Besozzi, P. Cazzaniga, D. Pescini, and G. Mauri. Modelling Metapopulations with Stochastic Membrane Systems. *Biosystems*, 91(3):499–514, 2008.

[19] L. Bortolussi, S. Fonda, and A. Policriti. Constraint-based simulation of biological systems described by Molecular Interaction Maps. In *IEEE International Conference on Bioinformatics and Biomedicine (BIBM) 2007*, pages 288–293, 2007.

[20] P. Bottoni, C. Martín-Vide, G. Păun, and G. Rozenberg. Membrane systems with promoters/inhibitors. *Acta Informatica*, 38:695–720, 2002.

[21] L. Brodo, P. Degano, and C. Priami. A stochastic semantics for bioambients. In V. Malyshkin, editor, *Parallel Computing Technologies*, volume 4671 of *Lecture Notes in Computer Science*, pages 22–34. Springer Berlin / Heidelberg, 2007.

[22] N. Busi. Using well-structured transition systems to decide divergence for catalytic P systems. *Theoretical Computer Science*, 372(2-3):125–135, 2007.

[23] L. Cardelli. Brane Calculi – Interactions of Biological Membranes. In V. Danos and V. Schachter, editors, *Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer Berlin / Heidelberg, 2005.

[24] L. Cardelli and P. Gardner. Processes in Space. In F. Ferreira, B. Löwe, E. Mayordomo, and L. Mendes Gomes, editors, *Programs, Proofs, Processes*, volume 6158 of *Lecture Notes in Computer Science*, pages 78–87. Springer Berlin / Heidelberg, 2010.

[25] L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[26] M. Cardona, M. À. Colomer, A. Margalida, A. Palau, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and D. Sanuy. A computational modeling for real ecosystems based on P systems. *Natural Computing*, 10(1):39–53, 2011.

[27] M. Cardona, M. À. Colomer, A. Margalida, I. Pérez-Hurtado, M. J. Pérez-Jiménez, and D. Sanuy. A P System Based Model of an Ecosystem of Some Scavenger Birds. In G. Păun, M. J. Pérez-Jiménez, A. Riscos-Núñez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 5957 of *Lecture Notes in Computer Science*, pages 182–195. Springer Berlin / Heidelberg, 2010.

[28] M. Cardona, M. À. Colomer, M. J. Pérez-Jiménez, D. Sanuy, and A. Margalida. Modeling Ecosystems Using P Systems: The Bearded Vulture, a Case Study. In D. W. Corne, P. Frisco, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing: 9th International Workshop*, volume 5391 of *Lecture Notes in Computer Science*, pages 137–156, 2009.

[29] F. Ciocchetta and J. Hillston. A framework for the modelling and analysis of biological systems. Technical report, School of Informatics, University of Edimburgh, 2008.

[30] F. Ciocchetta and J. Hillston. Process Algebras in Systems Biology. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016 of *Lecture Notes in Computer Science*, pages 265–312. Springer Berlin / Heidelberg, 2008.

[31] F. Ciocchetta and J. Hillston. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, 410(33-34):3065–3084, 2009. Concurrent Systems Biology: To Nadia Busi (1968-2007).

[32] F. Ciocchetta, C. Priami, and P. Quaglia. Modeling Kohn Interaction Maps with Beta-Binders: An Example. In C. Priami, E. Merelli, P. Gonzalez, and A. Omicini, editors, *Transactions on Computational Systems Biology III*, volume 3737 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin / Heidelberg, 2005.

[33] M. À. Colomer, A. Margalida, D. Sanuy, and M. J. Pérez-Jiménez. A bio-inspired computing model as a new tool for modeling ecosystems: The avian scavengers as a case study. *Ecological Modelling*, 222(1):33–47, 2011.

[34] P. Degano, D. Prandi, C. Priami, and P. Quaglia. Beta-binders for Biological Quantitative Experiments. *Electronic Notes in Theoretical Computer Science*, 164(3):101–117, 2006. Proceedings of the 4th International Workshop on Quantitative Aspects of Programming Languages (QAPL 2006), Quantitative Aspects of Programming Languages 2006.

[35] J. Dockery and J. Keener. A Mathematical Model for Quorum Sensing in Pseudomonas aeruginosa. *Bulletin of Mathematical Biology*, 63(1):95–116, 2001.

[36] X. Efthymiou and A. Philippou. A Process Calculus for Spatially-explicit Ecological Models (Extended Abstract). In *Proceedings of the First International Workshop*

on Application of Membrane Computing, Concurrency and Agent-based Modelling in Population Biology (AMCA-POP 2010), 2010.

[37] R. Freund, G. Păun, and M. J. Pérez-Jiménez. Tissue P systems with channel states. *Theoretical Computer Science*, 330(1):101–116, 2005.

[38] L. Galli-Resta, E. Novelli, Z. Kryger, G. H. Jacobs, and B. E. Reese. Modelling the mosaic organization of rod and cone photoreceptors with a minimal-spacing rule. *European Journal of Neuroscience*, 11:1461–1469, 1999.

[39] M. Gheorghe, V. Manca, and F. Romero-Campero. Deterministic and stochastic P systems for modelling cellular processes. *Natural Computing*, 9:457–473, 2010.

[40] D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[41] D. T. Gillespie. Simulation Methods in Systems Biology. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016 of *Lecture Notes in Computer Science*, pages 125–167. Springer Berlin / Heidelberg, 2008.

[42] M. Guerriero, C. Priami, and A. Romanel. Modeling Static Biological Compartments with Beta-binders. In H. Anai, K. Horimoto, and T. Kutsia, editors, *Algebraic Biology*, volume 4545 of *Lecture Notes in Computer Science*, pages 247–261. Springer Berlin / Heidelberg, 2007.

[43] J. Hillston. *A compositional approach to performance modelling*. Cambridge University Press, New York, NY, USA, 1996.

[44] P. Hogeweg. Evolving Mechanisms of Morphogenesis: on the Interplay between Differential Adhesion and Cell Differentiation. *Journal of Theoretical Biology*, 203(4):317–333, 2000.

[45] D. E. Irwin, J. H. Irwin, and T. D. Price. Ring Species as Bridges between Microevolution and Speciation. *Genetica*, 112–113(1):223–243, 2001.

[46] M. John, R. Ewald, and A. M. Uhrmacher. A Spatial Extension to the π-Calculus. *Electronic Notes in Theoretical Computer Science*, 194(3):133–148, 2008. Proceedings of the First Workshop From Biology To Concurrency and back (FBTC 2007).

[47] M. John, C. Lhoussaine, and J. Niehren. Dynamic Compartments in the Imperative π-Calculus. In P. Degano and R. Gorrieri, editors, *Computational Methods in Systems Biology*, volume 5688 of *Lecture Notes in Computer Science*, pages 235–250. Springer Berlin / Heidelberg, 2009.

[48] M. John, C. Lhoussaine, J. Niehren, and A. M. Uhrmacher. The Attributed Pi Calculus. In M. Heiner and A. M. Uhrmacher, editors, *Computational Methods in Systems Biology*, volume 5307 of *Lecture Notes in Computer Science*, pages 83–102. Springer Berlin / Heidelberg, 2008.

[49] M. John, C. Lhoussaine, J. Niehren, and A. M. Uhrmacher. The Attributed Pi-Calculus with Priorities. In C. Priami, R. Breitling, D. Gilbert, M. Heiner, and A. M. Uhrmacher, editors, *Transactions on Computational Systems Biology XII*, volume 5945 of *Lecture Notes in Computer Science*, pages 13–76. Springer Berlin / Heidelberg, 2010.

[50] K. W. Kohn. Molecular Interaction Map of the Mammalian Cell Cycle Control and DNA Repair Systems. *Molecular Biology of the Cell*, 10:2703–2734, 1999.

[51] K. W. Kohn, M. I. Aladjem, S. Kim, J. N. Weinstein, and Y. Pommier. Depicting combinatorial complexity with the molecular interaction map notation. *Molecular Systems Biology*, 2:1–12, 2006.

[52] K. W. Kohn, M. I. Aladjem, J. N. Weinstein, and Y. Pommier. Molecular Interaction Maps of Bioregulatory Networks: A General Rubric for Systems Biology. *Molecular Biology of the Cell*, 17:1–13, 2005.

[53] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6:128–142, 2004.

[54] V. Manca. The metabolic algorithm for P systems: Principles and applications. *Theoretical Computer Science*, 404(1-2):142–155, 2008.

[55] V. Manca and L. Bianco. Biological networks in metabolic P systems. *Biosystems*, 91(3):489–498, 2008.

[56] C. Martín-Vide, G. Păun, J. Pazos, and A. Rodríguez-Patón. Tissue P systems. *Theoretical Computer Science*, 296(2):295–326, 2003.

[57] P. Milazzo. *Qualitative and Quantitative Formal Modeling of Biological Systems*. PhD thesis, Università di Pisa, 2007.

[58] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[59] J. D. Murray. *Mathematical Biology: I. An Introduction*. Springer-Verlag, 3rd edition, 2002.

[60] J. D. Murray. *Mathematical Biology: II. Spatial Models and Biomedical Applications*. Springer-Verlag, 3rd edition, 2003.

[61] J. V. Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[62] A. A. Patel, E. T. Gawlinski, S. K. Lemieux, and R. A. Gatenby. A Cellular Automaton Model of Early Tumor Growth and Invasion: The Effects of Native Tissue Vascularity and Increased Anaerobic Tumor Metabolism. *Journal of Theoretical Biology*, 213(3):315–331, 2001.

[63] M. Patel and S. Nagl. Mathematical Models of Cancer. In S. Nagl, editor, *Cancer Bioinformatics: From Therapy Design to Treatment*, chapter 4, pages 59–93. John Wiley & Sons, 2006.

[64] D. Pescini, D. Besozzi, G. Mauri, and C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17(1):183–204, February 2006.

[65] C. Priami. Stochastic π-Calculus. *The Computer Journal*, 38(7):578–589, 1995.

[66] C. Priami and P. Quaglia. Beta Binders for Biological Interactions. In V. Danos and V. Schächter, editors, *Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 20–33. Springer Berlin / Heidelberg, 2005.

[67] C. Priami, A. Regev, E. Shapiro, and W. Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80(1):25–31, 2001.

[68] P Systems web page. `http://ppage.psystems.eu`.

[69] G. Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[70] G. Păun. *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.

[71] G. Păun and F. J. Romero-Campero. Membrane Computing as a Modeling Framework. Cellular Systems Case Studies. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016 of *Lecture Notes in Computer Science*, pages 168–214. Springer Berlin / Heidelberg, 2008.

[72] A. Regev. *Computational Systems Biology: a Calculus for Biomolecular Knowledge.* PhD thesis, Tel Aviv University, 2002.

[73] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Y. Shapiro. BioAmbients: an abstraction for biological compartments. *Theoretical Computer Science*, 325(1):141–167, 2004.

[74] A. Regev, W. Silverman, and E. Y. Shapiro. Representing biomolecular processes with computer process algebra: π-calculus programs of signal transduction pathways. In *Proceedings of the Pacific Symposium on Biocomputing*, 2000.

[75] A. Regev, W. Silverman, and E. Y. Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 459–470, 2001.

[76] C. W. Reynolds. Flocks, Herds and Schools: A Distributed Behavioral Model. *SIGGRAPH Computer Graphics*, 21:25–34, 1987.

[77] F. J. Romero-Campero and N. Krasnogor. An Approach to the Engineering of Cellular Models Based on P Systems. In *Mathematical Theory and Computational Practice*, volume 5635 of *Lecture Notes in Computer Science*, pages 430–436. Springer Berlin / Heidelberg, 2009.

[78] F. J. Romero-Campero and M. J. Pérez-Jiménez. A Model of the Quorum Sensing System in Vibrio fischeri Using P Systems. *Artificial Life*, 14(1):95–109, 2008.

[79] F. J. Romero-Campero, J. Twycross, M. Cámara, M. Bennett, M. Gheorghe, and N. Krasnogor. Modular Assembly of Cell Systems Biology Models Using P Systems. *International Journal of Foundations of Computer Science*, 20(3):427–442, 2009.

[80] T. Sawa, M. Ohara, K. Kurahashi, S. S. Twining, D. W. Frank, D. B. Doroques, T. Long, M. A. Gropper, and J. P. Wiener-Kronish. In Vitro Cellular Toxicity Predicts Pseudomonas aeruginosa Virulence in Lung Infections. *Infection and Immunity*, 66(7):3242–3249, 1998.

[81] D. Sburlan. Non-cooperative P Systems with Priorities Characterize PsET0L. In R. Freund, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing*, volume 3850 of *Lecture Notes in Computer Science*, pages 363–370. Springer Berlin / Heidelberg, 2006.

[82] H. S. Silva and M. L. Martins. A cellular automata model for cell differentiation. *Physica A: Statistical Mechanics and its Applications*, 322:555–566, 2003.

[83] P. Sosík and R. Freund. P Systems without Priorities Are Computationally Universal. In G. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing*, volume 2597 of *Lecture Notes in Computer Science*, pages 400–409. Springer Berlin / Heidelberg, 2003.

[84] K. Takahashi, S. N. Arjunan, and M. Tomita. Space in systems biology of signaling pathways – towards intracellular molecular crowding in silico. *FEBS Letters*, 579(8):1783–1788, 2005.

[85] C. Versari. A Core Calculus for a Comparative Analysis of Bio-inspired Calculi. In R. De Nicola, editor, *Programming Languages and Systems*, volume 4421 of *Lecture Notes in Computer Science*, pages 411–425. Springer Berlin / Heidelberg, 2007.

[86] C. Versari and N. Busi. Stochastic biological modelling in the presence of multiple compartments. *Theoretical Computer Science*, 410(33-34):3039–3064, 2009.

[87] C. Versari and R. Gorrieri. $\pi@$: A π-Based Process Calculus for the Implementation of Compartmentalised Bio-inspired Calculi. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *Formal Methods for Computational Systems Biology*, volume 5016 of *Lecture Notes in Computer Science*, pages 449–506. Springer Berlin / Heidelberg, 2008.

[88] D. J. Wilkinson. *Stochastic Modelling for Systems Biology*. Mathematical and Computational Biology Series. Chapman and Hall/CRC, 1st edition, 2006.