Università degli studi di Pisa

Dipartimento di informatica
Corso di Laurea in informatica

Laurea specialistica in infromatica

Multilevel Attributed Probabilistic P System Implementation and Definition

Candidato:
Alessandro Bompadre
Matricola 264312

Relatori:
Prof. Roberto Barbuti
Dott. Pasquale Bove
Dott. Paolo Milazzo

Anno Accademico 2014-2015

Abstract

The aim of this thesis is to define a model based on P Systems called Multilevel Attributed Probabilistic P Systems (MAPPS) as a tool for modelling complex social structure of animal population. In the initial part are shown earlier models that represent the basis for the creation of MAPPS and which have been applied to social structures less complex than those handled in this thesis. In a detailed central section this thesis provides a formal definition of MAPPS, with semantics and simple examples. This thesis then presents a case of study suitable for MAPPS formalism representing the social structure of Serengeti lions. In appendix it is shown a commentary to the code developed to collect data and results of the proposed model. The software is a General purpose engine capable of take in input models described with the formalism of MAPP systems and models described with the formalism of Attributed Probabilistic P Systems (APPS) producing a simulation and giving in output results and log files.

Acknowledgements

I wish to remind all those who helped me in the writing of this thesis with suggestions, criticisms and comments: to them goes my gratitude, all persons mentioned on this page have been instrumental in the drafting of this thesis, but I want to clarify that any error or inaccuracy is due only to me. I first thank professor Roberto Barbuti, doctor Paolo Milazzo without their support and their wise guidance this thesis would not exist. I would like to thank doctor Pasquale Bove for supporting me during all this time and for his advice and all the evenings spent together on books. I would also like to thank doctor Giovanni Pardini for his comments, suggestions and for so many coffee offered.

I would also like to thank the people dearest to me: my parents Rossana and Severino, my sister Marzia, my friends Ugo and Francesco, my housemate Emanuele, and finally my girlfriend Elena who was with me till the end, to all of them and to my grandparents who watch over me from up there, this work is dedicated.

Contents

Chapter 1

Introduction

1.1 Motivation

The animal social behaviour is the suite of interactions that occur between two or more individual animals, usually of the same species, when they form simple aggregations, cooperate in sexual or parental behaviour, engage in disputes over territory and access to mates, or simply communicate across space. The social interactions refer to particular forms of externalities, in which the actions of a reference group affect an individual's preferences. The reference group depends on the context and is typically an individual's family, neighbors, relatives or peers. Ethology is the scientific and objective study of animal behaviour and social interactions, usually with a focus on behaviour under natural conditions, and viewing behaviour as an evolutionarily adaptive trait.

Understanding ethology or animal behaviour can be important in animal training. Considering the natural behaviours of different species or breeds enables the trainer to select the individuals best suited to perform the required task. It also enables the trainer to encourage the performance of naturally occurring behaviours and also the discontinuance of undesirable behaviours. Ethologists are typically interested in a behavioural process rather than in a particular animal group, and often study one type of behaviour, such as aggression, in a number of unrelated animals.

Using mathematical models of animal social structures, it is possible to systematically test and predict the effect of human interventions, such as reintroduction of animal species in a specific environment [1]. These models are typically built in an iterative cycle of experiment and refinement, by multidisciplinary research teams that include biologists, ethologists and computer scientists. Similarly, simulations produced by these models can give guidance on how to preserve a certain state of the system. For example researchers used this approach to investigate and better support the sustainability of the soles stock in a given portion of Adriatic sea. [2]

The study of biological macroscopic models, through stochastic simulations, is represented by a whole set of formalisms, methodologies and implementations, that allows to better understand the evolution and interaction of animal population of one or more species in a well defined environment. For a long time, ethologists, ecologists and social scientists have faced the challenge of how to handle the complexity of these systems, which have common elements with many other theoretical and practical problems. Well known biological phenomena, like chemical reactions or physical phenomena, partially share solutions and methodologies with social modelling, mathematical modelling and complex systems.

Many models used in the study of the social behavior of animals have born and evolved within the field of study of *systems biology* (SB), a science that studies living organisms. Traditionally, the study of biological systems has involved the development of mathematical models, often based on differential equations, which permit the description and analysis of their behaviour. The use of abstract models has many advantages such as, for example, it allows the development of simulators which, in turn, may reduce the need for laboratory experiments. However, as the complexity of the system increases, mathematical models become more difficult both in the specification and in the analysis [50]. Moreover, they may not be well-suited for modelling particular systems: for example, using differential equations for modelling the variation in concentrations of reactants in a solution is sufficiently accurate only when the number of reactants is high, whereas becomes less accurate when their number is low. This is because the number of reactants is abstracted as a continuous variable. A great effort to the modelling of biological processes has been given by the use of modelling formalisms coming from Computer Science. Many formalism developed for modelling and analysing interactive and communicating systems have been applied to Biology [16], as they allow for a finer specification of their behaviour than traditional models.

Among these, there are: the π-calculus [48, 19, 15, 21], automata-based models [3, 43] and rewrite systems [33, 46] In particular, these formalisms have the ability to overcome some of the limitations of traditional models: for example, they allow to represent faithfully even systems comprising a small number of components, and among which complex interactions take place.

There are also formal models developed for the study of macro-biological phenomena. For example, in [14, 17] the BlenX and LIME languages are used to compositionally construct models of ecosystems, while in [42] PALPS models are encoded in PRISM language a probabilistic model checker. PALPS (*Process Algebra with Locations for Population Systems*) is a process algebra described as "the first process-algebraic framework developed specifically for reasoning about ecological models" [39]. Another process algebra developed for *Modelling in Ecology with Location Attributes* is called MELA [67]. It is suitable for formally describing ecological systems, with focus on space abstraction and environmental description.

In the tradition of automata and formal language theory, a interesting formalism, which will be discussed in this thesis later are P Systems, introduced by Păun [45, 46, 47]. P Systems introduce the idea of membrane computing in the subject of natural computing. They represent a new computational paradigm which allow solving NP-complete problem in polynomial time (but in exponential space). They originated a very big mass of work and recently they have been also applied to the description of biological systems (see [79] for a complete list of references). The P Systems have been expanded and modified to be used in several case studies. The *Attributed Probabilistic P Systems* (APP) were used to model social interactions in primates [11] and *Minimal Probabilistic P Systems* were used to model the reproduction in frogs [6]. In [25] a variant of P systems is used to model the dynamics of some endangered species in the Pyrenees. In [5, 4] Spatial PSystems are proposed and used to model the behaviour of shoals of fish.

These extended versions of P systems have proved to be suitable formalisms for the previous examples. What seems to be lacking yet is an extension of these formalisms able to model social interactions of a species characterized by a complex social structure, where the hierarchy is not linear, where several groups interact each other sometimes cooperatively sometimes competitively sometimes merging each other. The human society is not the only example of complex social organization, we find other realities in nature too. For example, the lions are the most socially inclined of all wild felids. A pride of lions is characterized by a complex social structure composed of nomadic and stable social units. Such a complex and fluid structure would become hard to be represented without the aid of a formalism that can model interactions between group of individuals. Therefore, in this thesis we aim to present a formalism that meets this need.

1.2 Contributions

The continuous addition of new features to the standard models of P Systems and their evolution suggests that ethology requires specific models to manage the complexity of the case studies handled. In this Thesis we show how the study and modeling of a group of animals characterized by complex and diverse social structures, such as the lions of the Serengeti, requires a further evolution of P systems. Where rules are applied not only to terminal elements but also to membranes. We call this new model *Multilevel Attributed Probabilistic P Systems* (MAPPS). The purpose of MAPPS is to provide a more flexible tool than previous models, where the groups of individuals are not statically defined at the outset, but may evolve, increase in number or even disappear during the various stages of computation.

These groups can be seen both as specific groups of individuals, such as families, as that area groupings, such as the members of a species present in a specific area. They may also be used to identify different species such as a group of predators and a group of prey. In all these cases the groups may change or individuals within it can move from one group to another. For example, puppies can move out from their families when they grow up, groups of predators can disperse groups of prey, groups of animals can move from one area to another. This instability of the groups and the mobility of its members need to be kept into account in our formalism if we want to represent realistic models. MAPPS are designed to be suitable to describe social interactions of a complex social structure. In this thesis we produce a formal definition of MAPPS and its semantics.

In order to show a possible MAPPS application we propose a simulation based on our model that takes as a case of study the lions of the Serengeti and we produce the obtained results. As mentioned above, a pride of lions is composed by a complex and fluid hierarchy, where a subgroup of a pride, called *court* and formed only by females and cubs is permanently settled into a location while another subgroup, formed by males, called *coalition* could be nomadic for a long while. Social interactions between this two subgroups of lions represent the reading key that allows us to understand the complex dynamics of a pride. The need to model the interactions between groups and not just between individuals is one of the reasons that led us to choose the lions of the Serengeti as a case study to apply a model based on MAPPS.

Moreover, we have developed a simulator for MAPPS, it is actually a general purpose engine written in C#. The engine can take in input a model in MAPPS and return as output a log of the entire computation. The engine can be applied to several models. The obtained results can be processed to elaborate statistical surveys and make graphs

1.3 Published Material

The definition of APP systems presented in Section 2.2.3 has appeared in [11] and presented to the fourth international symposium on *Modelling and Knowledge Management applications: Systems and Domains* (MoKMaSD 2015). The published material is presented in this thesis in revised and extended form.

Chapter 2

State of the art

The state of the art in the formal modelling of ecosystems consists of studies in which formal notations are used to model and simulate population dynamics and ecosystems but also of some notable examples of formalisms used over the years and which are part of the classical methods for modeling macro-biological systems. Some of them have been defined with the specific purpose of describing biochemical networks and activity of membranes inside cells. Moreover, some of them have been inspired by the $\pi-$calculus process algebra of Milner [53], which is a standard foundational language for concurrency theory.

Cellular automata (1940s)
Were created by von Neumann and Ulam in the 40's [2]. They are discrete dynamic models that consist on a grid of cells with a finite number of states. A cellular automaton has an initial configuration that changes at each time step through a predefined rule that calculates the state of each cell as a function of the state of its neighbors at the previous step. They are specially suited for modeling complex phenomena in a scale-free manner and have been used in biological studies for a long time [81]. Due to their spatial features their main applications are related to molecular dynamics and cellular population dynamics.

Applications: An application example of cellular automata applied to macro-biological problems is DISPAS, Demersal fish Stocks Probabilistic Agent-based Simulator [26].It has been initially developed to investigate and support the sustainability of the soles stock in a given portion of the Adriatic sea where a wide study area is divided in hexagonal thanks to the introduction of a model based on Cellular Automata paradigm (CA), an idealization of a physical system in which space and time are discrete.

Petri nets (1962)

Were created by Carl Adam Petri in the 60's for the modeling and analysis of concurrent systems [59]. They are bipartite graphs with two types of nodes, places and transitions, connected by directed arcs. Places hold tokens that can be produced (respectively, consumed) when an input (respectively, output) transition fires. The execution of a Petri net is non-deterministic and specially suited for distributed systems with concurrent events.

Applications: Their application to biological processes began in 1993, by the work of Reddy and coworkers, to overcome the limitations in quantitative analysis of metabolic pathways [60]. There are currently several Petri net extensions (e.g.: coloured, timed, stochastic, continuous, hybrid, hierarchical, functional), forming a very versatile framework for both qualitative and quantitative analysis. Due to this versatility, they have been used in metabolic [118, 61, 63], gene regulatory [62, 66], and signaling networks [64, 44, 104, 65]. Also, they are suited for integrating different types of networks, such as gene regulatory and metabolic [120].

Lindenmayer systems (1968)

(or L Systems) are one of the oldest formalisms introduced and developed in 1968 by Aristid Lindenmayer [68].

Applications: An L system is a formal grammar most famously used to model the growth processes of plant development.

Boolean networks (1969)

In 1969 Kauffman introduced Boolean networks to model gene regulatory networks [72]. They consist on networks of genes, modeled by boolean variables that represent active and inactive states. At each time step, the state of each gene is determined by a logic rule which is a function of the state of its regulators. The state of all genes forms a global state that changes synchronously. For large network sizes (n nodes) it becomes impractical to explore all possible states (2^n).

Applications: This type of model can be used to find steady-states (called attractors), and to analyze network robustness [100]. Boolean networks can be inferred directly from experimental gene expression time-series data [95, 96]. They have also been applied in some studies to model signaling pathways [103, 122]. To cope with the inherent noise and the uncertainty in biological processes, stochastic extensions like Boolean networks with noise [97] and Probabilistic Boolean networks [98] were introduced.

Process algebras (1980s)

A family of formal languages for modeling concurrent systems are process algebras. They generally consist on a set of process primitives, operators for sequential and parallel composition of processes, and communication channels. The Calculus of Communicating Systems (CCS) was one of the first process algebras, developed during the 80's by Robin Milner [54]. In 1992 Robin Milner developed a process algebra called $\pi-$calculus [55].

Applications: In [41, 40] a process algebra is proposed and used to model population dynamics by taking spatial distribution of the individuals into account. In [29] the Bio-PEPA process algebra is used to describe epidemiological problems, again by including a notion of spatiality. Other relevant biological applications of process algebras include:

Bio-calculus (1999) Nagasaki, Onami, Miyano and Kitano in [117] define the Bio-calculus as an expression system designed to make a bridge between biology and computer science.
Applications: It can be used to describe and simulate some molecular interaction.

Applications of $\pi-$calculus (1992) More than a decade ago, Regev and Shapiro published their pioneer work on the representation of signaling pathways with $\pi-$calculus [48, 49]. Their idea is to describe metabolic pathways as $\pi-$calculus processes and in [20] they showed how the stochastic variant of the model (BioSpi), defined by Priami in [18], can be used to represents both qualitative and quantitative aspects of the systems described.

Beta binders (2005) Beta-binders is a language of processes with typed interaction sites which has been introduced by Priami and Quaglia [21]. A year later, Degano, Prandi, Priami, Quaglia, enriched syntax and semantics of Beta-binders to achieve a stochastic version of them, in order to obtain quantitative measures on biological phenomena [22].

Brane Calculi (2005) More details of membrane interactions have been considered by Cardelli in the definition of Brane Calculi [31, 32], which are elegant formalisms for describing intricate biological processes involving membranes. Moreover, a refinement of Brane Calculi have been introduced by Danos and Pradalier in [34].

SpacePi (2008) John, Ewald and Uhrmacher devolepd an extension of π-calculus called SpacePi [113]. In this formalism π-processes are embedded into a vector space and move individually. Only processes that are sufficiently close can communicate. The SpacePi extends π-calculus also in time, the operational semantics of SpacePi defines the interactions between movement, communication, and time-triggered events.

Bio-PEPA (2008) Developed by Ciocchetta and Hillston [27, 28], Bio-PEPA are a process algebra for the modelling and the analysis of biochemical networks. Bio-Pepa is an extension of PEPA, a process algebra originally defined for the performance analysis of computer systems.

BlenX (2008) Dematte, Priami, Romanel and Soyer used the Beta Workbench and its BlenX language to study the evolution of biological networks [23]. In 2009 Priami, Ballarini, Quaglia developed BlenX4Bio, an high-level interface for BlenX [24].

Differential equations

Differential equations are mathematical equations that contain derivatives, either ordinary derivatives or partial derivatives. They describe the rate of change of continuous variables. They are typically used for modeling dynamical systems in several areas.

Applications: Systems of non-linear ordinary differential equations (ODEs) have been used in systems biology to describe the variation of the amount of species in the modeled system as a function of time. They have been applied to all kinds of biological pathways [86, 91, 112, 92]. With a fully detailed kinetic model, one can perform time-course simulations, predict the response to different inputs and design system controllers. However, building ODE models requires insight into the reaction mechanisms to select the appropriate rate laws, and experimental data to estimate the kinetic parameters. Other types of differential equations, such as stochastic differential equations (SDEs) and partial differential equations (PDEs) can be used respectively to account for stochastic effects and spatial distribution [116]. Piecewise-linear differential equations (PLDEs) have been used to integrate discrete and continuous features in gene regulatory networks [99, 87].

Agent-based models(2000s)

This formalism describes the interactions among multiple autonomous agents. They are similar in concept to cellular automata, except in this case, instead of using a grid and synchronized time steps, the agents move freely within the containing space. Likewise, they are used to study complex phenomena and emergent dynamics using populations of agents with simple rules.

Applications: In ethology, Agent Based Model are used to study biological reality as in the paper of Martha Robbins and Andrew m. Robbins [52], which represents an excellent example of ABMs. The rules are related to individual monkeys where each monkey is represented by an agent characterized by a set of data such as age and gender. The sum of the states of every individuals make up the next state of entire model. After each iterative step, the system checks the status and decide whether to continue with the next iteration or stop. In this model, as well as in many others, the system does not produce a single simulation but a set of these that, in the final output, will be analysed with statistical methods, such as the classic Monte Carlo method [78].
Another example of ABMs applied to macro-biological problem is DISPAS [2]. In the model, the behaviour of a single agent simulates the behaviour of a single fish. By applying the entire model we can formally specify the behaviour of the whole stock as a function of time. In addition, we can easily reproduce the interactions with other species and the marine environment representing them as probabilities of natural death.

Rule-based modeling (2000s)

This category consists of the formalisms of non-deterministic parallel computing systems, also called Rewriting System [71]. A String Rewriting System (SRS), uses the free monoid structure of strings (words) on an alphabet which extends a relationship R for all strings in the alphabet which contains respectively the left and right side of certain rules as sub-strings. Formally an SRS is a tuple (Σ , R), where Σ is an alphabet, usually limited, and R is a binary relation between some strings in the alphabet, called rewriting rule. This formalism comprises a recent approach to the problem of multi-state components in biological models. In rule-based formalisms the species are defined in a structured manner and support multiple states. The reaction rules are defined as transformations of classes of species, avoiding the need for specifying one reaction per each possible state of a species. This high-level specification is then automatically transformed into a biochemical network with the set of species and reactions generated by the specification. The main advantage of the rule-based approach is that it can avoid the combinatorial explosion problem in the generation and simulation of the complete reaction network by performing stochastic simulations that only instantiate the species and reactions as they become available [73, 74] or by the generation of coarse-grained ODE systems [127] . Spatial simulation has been addressed recently by the inclusion of geometric information as part of the structure of the species [115].

Applications: This kind of formalism is implemented in BioNetGen [119] which generates an ODE model or a stochastic simulation from the ruled-based specification. It has been applied in the modeling of different signaling pathways [105, 123, 124, 126]. A similar rule-based formalism used for this kind of pathways is the $k-$language, where the species are defined by agents that have a structured interface for interaction with other agents [35, 36, 127]. The possible interactions are defined by a set of rules, which can be visualized by a contact map. BIOCHAM implements a rule-based approach for model specification which is complemented with a temporal logic language for the verification of the properties the biological models [121].

P Systems (1998)

The P Systems [45, 46, 47] are another example of rule-based system formalisms that have proved to be easily adaptable to the study of macro-biological problems. We will focus on its developments on the section Background.

$k-$calculus (2004)

This is an interesting example of rule-based model created by Danos and Laneve [33]. $k-$calculus is a pioneering formalism in the description of biological systemss. It is a formal language for protein interactions, it is enriched with an intuitive visual notation and it has been encoded into the $\pi-$calculus. The $k-$calculus idealizes protein-protein interactions, essentially as a particular restricted kind of graph$-$rewriting operating on graphs with sites. A formal protein is a node with a fixed number of sites, and a complex (i.e. a bundle of proteins connected together by low energy bounds) is a connected graph built over such nodes, in which connections are established between sites. The $k-$calculus has been recently extended to model also membranes [37].

Chapter 3

Background

3.1 Definition of multiset and related operations

In this section we present some of key concepts about multiset theory that will be used frequently during the discussion of the next topics: a multiset is an extension of the concept of a set. While a set can contain only one occurrence of any given element, a multiset may contain multiple occurrences of the same element. Note that by this definition, a set is also classified as a multiset. In a multiset the multiplicity of an element is the number of instances of the element in a specific multiset.

Definition 3.1 (multiset)**.** *Given a set $S = \{s_1, \dots, s_k\}$, a* multiset *is a set of pairs $w = \{(s_1, n_1), \dots, (s_k, n_k)\}$ where $\forall i \in \{1, k\}$ $n_i \in \mathbb{N}$.*

- *We represent a multiset w as $\{\!|w_1, \dots, w_n|\!\}$ where $w_j \in S$ and for each $s_i \in S$ there are exactly n_i elements of $w_1, \dots, w_n$ that are instance of s_i. The set S is called* support *of w;*
- *We denote with $|w|_{s_i}$ the value of n_i;*
- *We denote with $|w|$ the size (number of elements) of the multiset w, and with $-$ and $+$ the difference and the union of multisets, respectively;*

For the sake of simplicity we will often use strings to represent multisets. A character of a string represents an element of a multiset. For our purposes the strings *aaab*, *abaa*, *aaba* are equivalent and describe the same multiset $\{\!|a, a, a, b|\!\}$. We will also abbreviate multiset representations by using apices to denote repetitions. For example we will represent *aaab* as a^3b. This way to describe multisets is useful when we use the operator $+$ to append a string to another to describe the union over multisets. In this way, to represent the union of two multiset, we use $aab + ab = aabab$. Given a set S of elements, S^* denotes the universe of all multisets having S as support.

3.2 Notions of P Systems

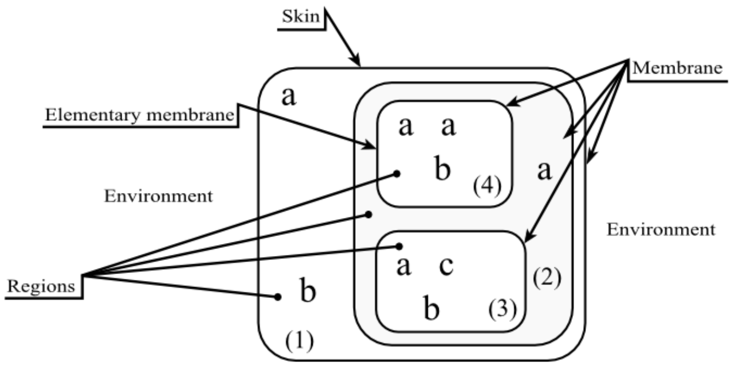


Figure 3.1: Membrane structure of a P System

The P Systems [46] are a form of rewriting systems originally defined as a bio-inspired model of computaton that have proved to be easily adaptable to the study of macro-biological problems. P Systems are an abstract model of parallel and distributed computing, which is inspired by molecular membranes and cellular compartments.

A P system consists of a hierarchy of membranes that do not intersect each other. The highest level membrane is called *skin membrane* whereas a membrane that does not contain any other membrane is called *elementary membrane*. See Figure 1 for an example of membrane structure. Membranes are seen as the demarcation lines between *regions*. Each membrane is associated with a single region. The space around the skin membranes is called the outer region or environment. Because of this one-to-one correspondence the terms membrane and region are used as synonyms sometimes.

Each region is associated with a multiset of objects, described by symbols in a given alphabet and with a set of evolution rules. According to Păun definition, multisets of objects in a region represent chemicals in the solution inside the cell compartment, while the rules correspond to the possible chemical reactions within the same compartment. The objects evolve according to the evolution rules, that can consume and produce some of them, or send some of them to internal membranes or to the upper one. The produced objects can not be created in a membrane that is not immediately contained or that does not contain directly the membrane in which the rules have been applied. Moreover, an evolution rule can cause a membrane to be dissolved with the effect of releasing its contents to the outer membrane.

Evolution rules are applied with *maximal parallelism.* A maximal multiset of rules is chosen during each step and for any of these rules its reactant are consumed and new items are produced where required. The choice of the maximal multiset of rules is non-deterministic The computing process starts from an initial configuration, with a number of objects arranged within hierarchy of membranes. The system evolves by maximal parallel steps by taking into account that rules associated with the membrane act only on objects present in the same membrane.

A computation halts when there are no evolutionary rules that can be applied to the state reached by the system. The output of the system has been defined in several ways, for example we can consider as output the multiset present in a membrane when the computation stops, or the sequence of objects sent to the environment.

3.2.1 Formal definition of a P Systems

A P System has a tree-structure in which the skin membrane is the root and the membranes containing no other membranes are the leaves. The only change to the structure that may happen is the removal of some node of the tree (apart from the root) caused by evolution rules. A membrane structure can be represented graphically as a Venn diagram. P Systems as follows are formally defined as follow.

Definition 3.2 (P system)**.** *A* P System Π *is given by*

$$\Pi = (V, \mu, w_1, \ldots, w_n, (R_1, \rho_1), \ldots, (R_n, \rho_n))$$

where:

- V *is an* alphabet *whose elements are called* objects*;*
- $\mu \subset \mathbb{N} \times \mathbb{N}$ *is a* membrane structure*, such that* $(i, j) \in \mu$ *denotes that the membrane labeled by* j *is contained in the membrane labeled by* i*;*
- w_i *with* $1 \leq i \leq n$ *are strings from* V^* *representing multisets over* V *associated with the membranes* $1, 2, \ldots, n$ *of* μ*;*
- R_i *with* $1 \leq i \leq n$ *are finite sets of* evolution rules *associated with the membranes* $1, 2, \ldots, n$ *of* μ*; An evolution rule is a pair* (u, v)*, denoted* $u \to v$*, where* u *is a string over* V *and* v *is a string over* $(V \times \{here, out\}) \cup (V \times \{in_j | 1 \leq j \leq n\}) \cup \{\delta\}$ *and* δ *is a special symbol not in* V*.*

- *ρ_i is a partial order relation over R_i, specifying a* priority *relation between rules: $(r_1, r_2) \in \rho_1$ iff $r_1 > r_2$ (i.e. r_1 has a higher priority than r_2).*

Example computation

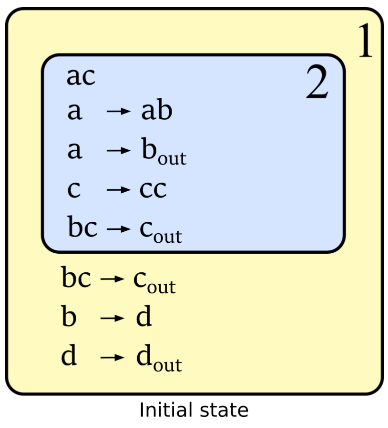


Figure 3.2: graphical representation of a P system

The image shown in figure 3.2 depicts the initial state of a P system with two membranes. Because of their hierarchical nature, P systems are often depicted graphically as Venn diagrams. The outermost membrane, is the skin membrane of this P system and contains three rules. The *out* tag represents that the target of products is out of the membrane. The innermost membrane, contains the multiset of symbols *ac* and four rules. In this initial state no rules outside of the most internal membrane are applicable: there are no symbols outside of that membrane. However, during evolution of the system, as objects are passed between membranes, the rules in other membranes will become active. This simple example does not have practical applications and serves only to provide an idea of the functioning of the computation of a P system.

Computation

Because of the non-deterministic nature of P systems, there are many different paths of computation a single P system is capable of, leading to different results. The following is one possible path of computation for the considered P system.

Step 1

From the initial configuration only the the most internal membrane has inside the following elements: ac. Object c is assigned to $c \to cc$, while a is assigned to $a \to ab$.

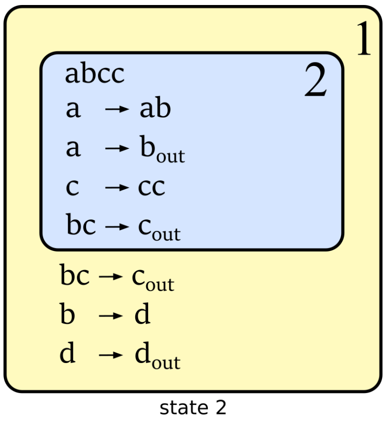


Figure 3.3: after step 1 of computation

Step 2

As we can see in figure 3.3, the most internal membrane now contains: $abcc$. Objects bc are assigned to $bc \to c_{out}$, while c is assigned to $c \to cc$, and a is assigned to $a \to b_{out}$. We remark that the choice of rules to be applied is non-deterministic.

Hence the application of the second rule $a \to b_{out}$ in place of to the first rule $a \to ab$ is non-deterministic due to the non-deterministic choice. The system could just as well have continued applying the first rule (and at the same time doubling the c particles) indefinitely. Since the *out* target has been encountered twice, in the second rule and in the fourth rule, the multiset of products bc is sent to membrane 1.

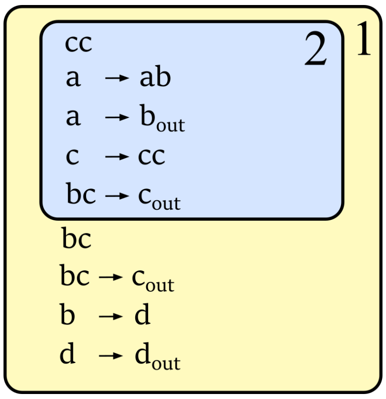


Figure 3.4: after step 2 of computation

Step 3

At this step, as shown in figure 3.4 we can see that the most internal membrane contains cc while the skin membrane contains bc. Notice the maximally parallel behaviour of rule application leading to the same rule being applied twice during one step. Each c in membrane 2 is assigned to $c \to cc$, while bc in the skin membrane is assigned to $bc \to c_{out}$. In figure 3.5

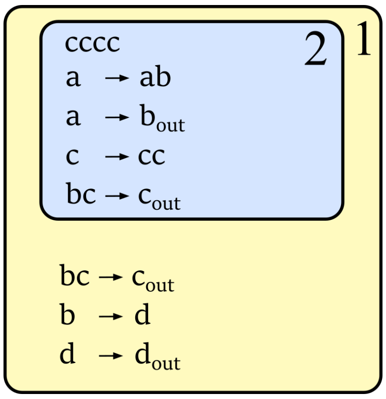


Figure 3.5: after step 3 of computationm

is shown the state after three steps. Notice that c is sent out of membrane 1, to the environment. In many cases the elements sent out of the skin membrane are seen as the output of the systems.

Notice that from now on the only rule applicable is $c \to cc$. We can not have other values sent out of the membrane 1 and of membrane 2.

3.2.2 Some relevant extensions

In this chapter we show some of the developments proposed by the scientific community to show some formalisms from which we took inspiration for the formulation of our paradigm. The P Systems are backed up by a very active community of researchers that has developed different versions and extensions to face different problems, both from the theoretical point of view and to respond to specific applications. Some, research teams have successfully used this type of solutions to face problems of animal species modelling. We will see in the examples below how the scholars introduced new elements in P Systems, such as explicit use of space or the introduction of probabilistic choices of maximal multiset of rules.

Dynamical Probabilistic P systems (DPPS)

are a variant of P Systems where each rule is associated with a constant

and rules are chosen according to a probability that is obtained at each computational step normalizing associated constants[69].
The origianl paradigm of P Systems proposed by Păun provides a non-deterministic choice of rules. According to the Păun definition[45], a rule cannot be selected if one of higher priority can be selected, that means that the non-determinism comes down to choosing between rules with equal priority. In DPPS we have a more sophisticated attempt to implement the non-determinism, in a probabilistic way, assigning to the rules a probabilistic rating.

Stochastic P systems

The Stochastic P Systems[56] are models derived from P Systems that implement a probabilistic choice of rules by using the *Stochastic Simulation Algorithm* (SSA), an algorithm of stochastic simulation of biochemical systems introduced by Gillespie in 1976. We see for instance, a similar implementation proposed in the article[58], in this example is shown as time has a fundamental importance in the algorithm of Gillespie. In SSA rules have a time duration, so we can know what is the next rule to be applied and how much time elapses between the application of a rule and another. This method may be useful for asynchronous systems or in general in all those systems in which it is important to know the duration of a given event in relation to others. The Stochastic P systems were implemented using the spatially explicit programming language MGS[57]. Examples of implementations include Lotka-Volterra models. auto-catalytic systems, and life cycle of Semliki Forest virus.

Spatial P systems

Spatial P systems are an extension of P systems where objects and membranes are included in a two-dimensional grid. They are characterized by the distinction between ordinary objects and mutually exclusive objects, with the requirement that each cell of the grid can contain any number of ordinary objects, and only one mutually exclusive object. In this work, what is emphasized is the concept of space, which is made explicit, and enriches the model. Space management and some features that belong the model increase the computational complexity in code of execution algorithm. It is possible to maintain low the computational cost by applying restrictions to the parameters of the system, retaining almost intact the expressiveness.

Multi environment Probabilistic Functional Extended P system

Monica Campbell and Angels Colomer present a rich model extension proposal in their paper. In this work several critical aspects of ecosystems modelling are faced such as: the introduction of a graph structure representing interconnected multi-habitats, the attribution of a probabilistic function to certain rules which work on the same set of reactants and the enrichment of membranes definition with specific attributes. This extension of P Systems is used in two cases studies: a study about European bearded vulture (Gypaetus barbatus) and one about zebra mussel in Ribarroja reserve.

3.2.3 Minimal Probabilistic P Systems

The variant of P systems defined below includes a minimal set of features useful for modelling population dynamics. This formalism together with that presented in the next section are the basis for what we propose in this thesis. For this reason we recallo here the formal definition of these two models. The first variant we consider are *Minimal Probabilistic P Systems* MPP systems are form of flat P systems[9], namely P systems consisting of a single membrane, since a membrane structure is not useful for the purposes of this formalism. The key ingredients that taken in account are:

- evolution rules with functional rates.
- probabilistic maximal parallelism
- rule promoters.

The aim of this variant of P systems is to make modelling of populations easier, by avoiding in the modelling formalism unnecessary functionalities.
In models of population dynamics, evolution rules are used to describe events such as reproduction, death, growth, and so on. In general there may be several rules describing one of these events and which can possibly applied, alternatively, to the same individuals. For instance, the same female individual may be involved in one of different reproduction rules, one rule for each possible kind of male it can mate with. Some of these rules may be more likely to be applied than others since the events they describe are more likely than others. (For instance, some females may have a sexual preference for some specific kind of males.) Associating rates with rules allows to choose rules in a probabilistic way, where probabilities are proportional to the rates. Moreover, by allowing rates to be functions, rather than constant values, the probability of applying a rule can depend on the current state of the system (for instance on the size of the population, or on the number of individuals of a specific kind). Probabilistic choice of rules have been considered in many formalisms for modelling biological systems [12, 8, 70, 13, 58, 10, ?].
In MPP systems probabilities of rules are used in conjunction with maximal parallelism.

Populations often evolve by stages (e.g. reproduction, selection, etc...) in which (almost) all of the individuals are involved. By combining maximal parallelism with probabilistic choice of reactions the systems allow the whole population to evolve in a coherent way and, at the same time, each individual to follow its own fate.
Finally, since in each stage of the evolution of a population different kinds of event may happen, it is necessary a way to enable different sets of rules depending on the current stage. For instance, during a reproduction stage only reproduction rules should be enabled, whereas during a selection stage only death/survival rules should be enabled. In order to obtain this result rule promoters are exploited, that can be used to enable/disable a set of rules by simply including/removing an object from the state of the system.

Definition 3.3 (MPP system). *A* Minimal Probabilistic P system *is a tuple* $\langle A, w_0, R\rangle$ *where:*

- *A is a possibily infinite alphabet of objects.*

- *$w_0 \in A^*$ is a multiset describing the initial state of the system;*

- *R is a finite set of evolution rules having the form*

$$u \xrightarrow{f} v \mid_{pr}$$

where $u, v, pr \in A^*$ are multisets (often denoted without brackets) of reactants*,* products *and* promoters*, respectively, and $f : A^* \mapsto \mathbb{R}^{\geq 0}$ is a* rate function.

A state (or configuration) of a MPP system is a multiset of objects in V^*. By definition, the initial state is w_0. We denote a generic state of the system as w, with $|w|$ the size (number of objects) of the multiset w, and with $|w|_a$ the number of instances of object a contained in multiset w.
The evolution of a MPP system is given by a sequence of probabilistic maximally parallel steps. In each step a maximal multiset of evolution rule instances is selected and applied as described by the following semantic rules.

$$\text{(rule application)}\quad \frac{\begin{array}{c} r_i:\ u \xrightarrow{k} v\ |_{pr} \in R \qquad u \subseteq w' \qquad pr \subseteq w \\ K = \{k' | u' \xrightarrow{k'} v'\ |_{pr'} \in R,\ u' \subseteq w',\ pr' \subseteq w\} \\ p = \frac{k}{\sum\limits_{k' \in K} k'} \end{array}}{(w', \overline{w}') \xrightarrow{r_i,\ p}_{(R,w)} (w' - u, \overline{w}' + v)}$$

$$\text{(single rule sequence)}\quad \frac{(w', \overline{w}') \xrightarrow{r_i,\ p}_{(R,w)} (w'', \overline{w}'')}{(w', \overline{w}') \xrightarrow{[r_i],\ p}{}^{+}_{(R,w)} (w'', \overline{w}'')}$$

$$\text{(multiple rules sequence)}\quad \frac{\begin{array}{c}(w', \overline{w}') \xrightarrow{r_i,\ p_i}_{(R,w)} (w'', \overline{w}'') \\ (w'', \overline{w}'') \xrightarrow{\overline{r},\ \overline{p}}{}^{+}_{(R,w)} (w''', \overline{w}''')\end{array}}{(w', \overline{w}') \xrightarrow{\overline{r}@[r_i],\ p_i \cdot \overline{p}}{}^{+}_{(R,w)} (w''', \overline{w}''')}$$

$$\text{(step rule)}\quad \frac{(w, \emptyset) \xrightarrow{\overline{r},\ \overline{p}}{}^{+}_{(R(w),w)} (w', \overline{w}') \qquad (w', \overline{w}') \not\rightarrow_{(R(w),w)}}{w \overset{\overline{r},\ \overline{p}}{\Longrightarrow}_R w' + \overline{w}'}$$

Where $[r_i]$ denotes the sequence composed by the single element r_i, and @ denotes the concatenation of sequences.

Given a system state, w, the (step rule) describes the evolution in a new state by the $\overset{\overline{r},\ \overline{p}}{\Longrightarrow}_R$ relation, where $\overline{p}$ is the probability of the transition, and $\overline{r}$ is the sequence of applied rules. (step rule) uses as a premise the transition $(w, \emptyset) \xrightarrow{\overline{r},\ \overline{p}}{}^{+}_{(R(w),w)} (w', \overline{w}')$ where $R(w)$ is the set of applicable rules in the state w, with their rates. The rates of the rules in $R(w)$ are normalized with respect to all the applicable rules in w, as defined as follows. Given a set of rules R we have:

$$\begin{array}{rcl} K(w) & = & \{k' \ \mid\ u' \xrightarrow{k'} v'\ |_{pr'} \in R,\ u' \subseteq w,\ pr' \subseteq w\}, \\ R(w) & = & \{u \xrightarrow{p} v\ |_{pr} \ \mid\ u \xrightarrow{k} v\ |_{pr} \in R,\ u \subseteq w,\ pr \subseteq w, p = \frac{k}{\sum\limits_{k' \in K(w)} k'}\} \end{array}$$

In the transition $(w', \varnothing) \xrightarrow{\overline{r},\ \overline{p}^{+}}_{(R,w)} (w', \overline{w}')$, $\overline{r}$ is a sequence representing the multiset of rules to be applies to w. (step rule) has also $(w', \overline{w}') \not\rightarrow_{(R(w),w)}$ as a premise in order to ensure that $\overline{r}$ corresponds to a maximal multiset of rules.

The transition relation $\xrightarrow{\overline{r},\ \overline{p}^+}_{(R,w)}$ is defined by rules (single rule sequence) and (multiple rule sequence) on the basis of transition relation $\xrightarrow{r_i,\ p}_{(R,w)}$. A transition $(w', \overline{w}') \xrightarrow{r_i,\ p}_{(R,w)} (w'-u, \overline{w}'+v)$ corresponds to the application of a single rule. When a rule is selected, its application consists in removing its reactants from w' and adding its products to $\overline{w}'$. The $\overline{w}'$ multiset will collect all products of all applied rules. Transition realtion $\overset{\overline{r},\ \overline{p}}{\Longrightarrow}_R$ is defined on the basis of two other transition relations: $\xrightarrow{r,\ p}_{(R,w)}$ and $\xrightarrow{\overline{r},\ \overline{p}}{}^{+}_{(R,w)}$. The former describes the application of a single rule as the latter describes a sequence of rule applications.

Note that each rule is applied with respect to the rates of the rules computed in the initial state w. This rates can change only if some rules cannot be applied to the remaining objects in w'. Note also that it is sufficient that the promoters are present in the initial state w, $pr \subseteq w$. Once objects in w' are such that no further rule in $R(w)$ can be applied to them, the (step rule) returns the new state of the system $w' + \overline{w}'$ (where w' are the unused objects and $\overline{w}'$ are the new products).

Intuitively, the semantic definition states that all the rules to be applied are selected in a probabilistic way from the set of applicable rules, their reactant are removed for the available reactants, w', and their product are added to a suspended multiset $\overline{w}'$. When no further rule can be applied to w' the new state, which is composed be the unused objects in w' plus the suspended products in $\overline{w}'$, is produced.

Finally we give the probability of a transition between two states by means of the following rule:

$$\boxed{\text{(state transition probability)} \quad \frac{PR = \{(\overline{r},\overline{p}) \mid w \overset{\overline{r},\ \overline{p}}{\Longrightarrow}_R w'\} \quad p = \sum_{(\overline{r},\overline{p})\in PR} \overline{p}}{w \overset{p}{\Longrightarrow}_R w'}}$$

There are different sequences of rules that go from w to w'. In $\overset{\overline{r},\ \overline{p}}{\Longrightarrow}_R$ these sequences correspond to different transitions. In $\overset{p}{\Longrightarrow}_R$ these sequences are grouped in a single transition with probability equal to the sum of the probabilities of single sequences. This leads to have a correct distribution of probability for a state w (the sum of its outgoing transition is 1) because the probabilities in $\overset{\overline{r},\ \overline{p}}{\Longrightarrow}_R$ were obtained through a normalization and in $\overset{p}{\Longrightarrow}_R$ these probabilities are grouped by summing the probabilities of those that lead to some next state.

Example computation

The figure 3.6 shown the initial state of a MPP system with only one membrane. Because of their flat nature, MPP systems are always depicted graph-

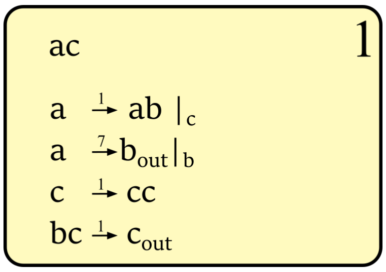

ac
1
a $\xrightarrow{1}$ ab $|_c$
a $\xrightarrow{7}$ $b_{out}|_b$
c $\xrightarrow{1}$ cc
bc $\xrightarrow{1}$ c_{out}

Figure 3.6: initial state Minimal Probabilistic P Systems

ically with only one membrane. The skin membrane is the container for all the elements of an MPP system. In this example, it contains four applicable rules. The *out* tag represents that the target of products is out of the skin membrane. This simple example does not have practical applications and serves only to provide an idea of the functioning of the computation of an MPP system.

Computation

Because of the probabilistic nature of MPP systems, there are many different paths of computation a single MPP system is capable of, leading to different results. The following is one possible path of computation for the considered MPP system.

Step 1

From the initial configuration the skin membrane has inside the following elements: ac. The element a is assigned to $a \xrightarrow{1} ab|_c$, the only rule applicable to the reagent a as the promoter b of rule $a \xrightarrow{7} b_{out}|_b$ is not present, while c is assigned to $c \xrightarrow{1} c_{out}$.

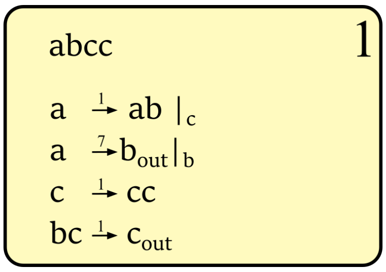

abcc
1
a $\xrightarrow{1}$ ab $|_c$
a $\xrightarrow{7}$ $b_{out}|_b$
c $\xrightarrow{1}$ cc
bc $\xrightarrow{1}$ c_{out}

Figure 3.7: after step 1 of computation

Step 2

At this step, as shown in figure 3.7, the skin membrane has inside the following elements: *abcc*. Both promoters b and c are presents, so both the first and the second rules can be applied. Notice that the probabilistic choice of the rules provides a probability of 1/8 to be chosen to the first rule, while the rule $a \xrightarrow{7} b_{out}|_b$ has a probability of 7/8 to be chosen. Assuming that the probabilistic choice assigns a to the second rule, bc to $bc \xrightarrow{1} c_{out}$ and c to $c \xrightarrow{1} cc$. Products c and b are sent out of membrane.

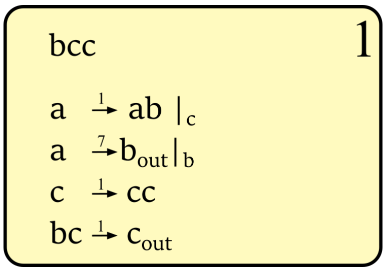

bcc
1
$a \xrightarrow{1} ab\ |_c$
$a \xrightarrow{7} b_{out}|_b$
$c \xrightarrow{1} cc$
$bc \xrightarrow{1} c_{out}$

Figure 3.8: after step 2 of computation

Step 3

As we can see in figure 3.8, there are no more a elements the only applicable rules are the third and the fourth, with same probabilities. Assuming that c is assigned to $c \xrightarrow{1} cc$ and bc is assigned to $bc \xrightarrow{1} c_{out}$. The product c is sent out of the membrane. From now on, the only applicable rule is $c \xrightarrow{1} cc$ and the number of c elements doubles step by step. Figure 3.9 show the state after three steps of computation.

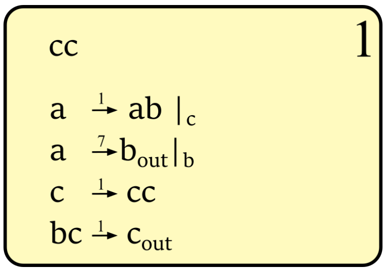

cc
1
$a \xrightarrow{1} ab\ |_c$
$a \xrightarrow{7} b_{out}|_b$
$c \xrightarrow{1} cc$
$bc \xrightarrow{1} c_{out}$

Figure 3.9: after step 3 of computation

3.2.4 Attributed Probabilistic P Systems

In the previous section, we have seen how the elements of MPP systems are seen as individuals of populations or as control elements. Control elements are used as promoters to model the subsets of active rules at various stages of computation. In MPP systems, we can model simple attributes of individuals adding symbols to the alphabet. For example we have an alphabet consisting of {a, b} that could represent two individual belonging to different species. If we want to specify the sex of each element we need to increase the number of symbols {am, af, bm, bf}.

When an attribute, as in this case the *sex*, can only have two possible values, such as *male* or *female*, it is easy to replace an element of the alphabet with two new elements, as in this case *am* and *af*. But if we want to insert an attribute such as age where possible attribute values are not only two, we may have to add a very large, possibly infinite, number of elements of the alphabet.

With the need to associate attributes to the elements, such as age, sex and strength, it is necessary to use a different model from MPP systems. The formulation of APP Systems is a natural consequence of this need. In this formalism we add attributes to enrich the model. The attributes of objects consumed by the rules greatly influence the functions of the rating associated with each rule generating a probabilistic system that is much more flexible and concise. Here we see in detail the APP Systems.

Definition 3.4 (APP system)**.** *An* Attributed Probabilistic P system*, ^aP, is a tuple $\langle A, arity, D_{a_1}, \ldots, D_{a_n}, w_0, R\rangle$ where:*

- *A is an ordered finite alphabet of symbols, $\{a_1, \ldots, a_n\}$;*
- *$arity : A \to \mathbb{N}$ is a function which for each $a_i \in A$ gives the arity of D_{a_i};*
- *each D_{a_i} is a set of tuples, $D_{a_i} = I_1 \times \ldots \times I_{arity(a_i)}$, where each I_j is a (possibly infinite) set of unstructured values; the set D_{a_i} is called the set of attributes of a_i;*
- *w_0 is a multiset of values in $\Sigma = \{\langle a_i, d_i\rangle \mid a_i \in A,\ d_i \in D_{a_i}\}$ describing the initial state of the system, where Σ is called the set of* objects *of P. In the following we will write $w_0 \in \Sigma^*$.*
- *given a set of variables V, R is a finite set of evolution rules having the form*

  $$u_{\mathrm{V}} \xrightarrow{f} v_{\mathrm{V}} \mid_{pr_{\mathrm{V}}}$$

  where $u_{\mathrm{V}}, pr_{\mathrm{V}} \in \Sigma_{\mathrm{V}}^*$ are multisets (often denoted without brackets) of objects and variables denoting reactants *and* promoters*, respectively;*

> $v_{\mathrm{V}} \in \Sigma^*_{\mathrm{EV}}$ *is a multiset of objects and* expressions *with variables denoting* products*; and* $f : \Sigma^* \mapsto \mathbb{R}^{\geq 0}$ *is a* weight function*. Precisely:*
>
> $$\Sigma_{\mathrm{V}} = \{(a_i, d_i) \mid a_i \in A, d_i \in D^{\mathrm{V}}_{a_i}\} \quad \Sigma_{\mathrm{EV}} \;\; = \{(a_i, e_i) \mid a_i \in A, e_i \in E^{\mathrm{V}}_{a_i}\}$$
>
> *where* $D^{\mathrm{V}}_{a_i} = (V \cup I_1) \times \ldots \times (V \cup I_{arity(a_i)})$*; and* $E^{\mathrm{V}}_{a_i} = Exp(V, I_1) \times \ldots \times Exp(V, I_{arity(a_i)})$*, with* $Exp(V, I)$ *denoting the set of well-typed expressions built from operators, variables* V*, and values of* I*. Moreover, we have* $Vars(v_{\mathrm{V}}) \subseteq Vars(u_{\mathrm{V}}) \cup Vars(pr_{\mathrm{V}})$*, where* $Vars(t)$ *denotes the set of variables occurring in* t*. Rules without variables are called* ground rules.

In what follows we will denote an (attributed) object $\langle a, d \rangle$ as $a_{(d)}$. A state (or configuration) of an APP system is a multiset of objects in Σ^*. By definition, the initial state is w_0, and we denote a generic state as w.
The evolution of an APP system is a sequence of probabilistic maximally parallel steps. We formally define the semantics of APP systems as a transition relation in the style of [7]. In each step a maximal multiset of evolution rule instances is selected and applied as described by the following semantic rules:

$$\text{(rule application)} \quad \frac{r_i = \;u \xrightarrow{k} v \in R \qquad u \subseteq w' \qquad\qquad K = \{\!|k' | u' \xrightarrow{k'} v' \in R,\; u' \subseteq w'|\!\} \qquad p = k / \sum_{k' \in K} k'}{(w', \overline{w}') \xrightarrow{r_i,\; p}_R (w' - u, \overline{w}' + v)}$$

$$\text{(single rule sequence)} \quad \frac{(w', \overline{w}') \xrightarrow{r_i,\; p}_R (w'', \overline{w}'')}{(w', \overline{w}') \xrightarrow{[r_i],\; p}{}^{+}_R (w'', \overline{w}'')}$$

$$\text{(multiple rules sequence)} \quad \frac{(w', \overline{w}') \xrightarrow{r_i,\; p_i}_R (w'', \overline{w}'') \qquad (w'', \overline{w}'') \xrightarrow{\overline{r},\; \overline{p}}{}^{+}_R (w''', \overline{w}''')}{(w', \overline{w}') \xrightarrow{\overline{r}@[r_i],\; p_i \cdot \overline{p}}{}^{+}_R (w''', \overline{w}''')}$$

$$\text{(step rule)} \quad \frac{(w, \emptyset) \xrightarrow{\overline{r},\; \overline{p}}{}^{+}_{R(w)} (w', \overline{w}') \qquad (w', \overline{w}') \not\rightarrow_{R(w)}}{w \overset{\overline{r},\; \overline{p}}{\Longrightarrow}_R w' + \overline{w}'}$$

where $[r_i]$ denotes the sequence composed of the single element r_i, and @ denotes the concatenation of sequences. The semantics of APP systems is similar to that of the MPP systems. What differs is the presence of variables and attributes. Given a system state, w, the (step rule) describes the evolution in a new state by the $\overset{\overline{r},\, \overline{p}}{\Longrightarrow}_R$ relation, where $\overline{p}$ is the probability of the transition, and $\overline{r}$ is the sequence of applied ground rules. (step rule) uses as a premise the transition $(w, \emptyset) \xrightarrow{\overline{r},\; \overline{p}}{}^{+}_{R(w)} (w', \overline{w}')$ where $R(w)$ is the set of applicable ground rules in the state w, with their weights, namely:

$$R(w) = \left\{ u_{\mathrm{V}}\sigma \xrightarrow{f(w)} v_{\mathrm{V}}\sigma \;\middle|\; u_{\mathrm{V}} \xrightarrow{f} v_{\mathrm{V}} \,|_{pr_{\mathrm{V}}} \in R,\; \exists \sigma.\, u_{\mathrm{V}}\sigma \subseteq w \wedge pr_{\mathrm{V}}\sigma \subseteq w \right\}$$

where (i) $\sigma : V \to flat(D_{a_1}) \cup \ldots \cup flat(D_{a_n})$, with $flat(D_{a_i}) = I_1 \cup \ldots \cup I_{arity(a_i)}$, for all $a_i \in A$; (ii) $u_V\sigma$ ($u_V \in \Sigma_V^*$) is the well-typed multiset obtained by substituting values for variables in u_V according to σ; and (iii) $v_V\sigma$ ($v_V \in \Sigma_{EV}^*$) is the well-typed multiset obtained by evaluating the expressions in v_V under the substitution σ. Transition relation $\xrightarrow{\overline{r},\, \overline{p}}{}^{+}_{R}$ is the transitive closure of $\xrightarrow{r,\, p}_R$.

A transition $(w', \overline{w}') \xrightarrow{r_i,\, p}_R (w'-u, \overline{w}'+v)$ corresponds to the application of a single rule. When a rule is selected, its application consists in removing its reactants from w' and adding its products to $\overline{w}'$. The $\overline{w}'$ multiset will collect all products of all applied rules. Note that $R(w)$ takes into account that each rule is applied with respect to the weights of the rules computed in the initial state w. Moreover, $R(w)$ contains only the ground rules the promoters of which are present in the initial state w ($pr_V\sigma \subseteq w$). Once objects in w' are such that no further rule in $R(w)$ can be applied to them, by (step rule) the new system state is $w' + \overline{w}'$ (where w' are the unused objects and $\overline{w}'$ are the new products).

Intuitively, the semantic definition states that all the rules to be applied are selected in a probabilistic way from the set of applicable rules, their reactant are removed for the available reactants, w', and their product are added to a suspended multiset $\overline{w}'$. When no further rule can be applied to w' the new state, which is composed be the unused objects in w' plus the suspended products in $\overline{w}'$, is produced. Finally we give the probability of a transition between two states by means of the following rule:

$$\boxed{\text{(state transition prob.)} \qquad \frac{PR = \{(\overline{r}, \overline{p}) \mid w \overset{\overline{r},\, \overline{p}}{\Longrightarrow}_R w'\} \qquad p = \sum_{(\overline{r},\overline{p}) \in PR} \overline{p}}{w \overset{p}{\Longrightarrow}_R w'}}$$

As in the case of MPP systems we have that the sum of probabilities of transitions for a given state is one since the probabilities in $\overset{\overline{r},\, \overline{p}}{\Longrightarrow}_R$ were obtained through a normalization and in $\overset{p}{\Longrightarrow}_R$ these probabilities are grouped by summing the probabilities of those that lead to the next state.

Example computation

The image shown in in figure 3.10 depicts the initial state of a APP system with only one membrane. Because of their flat nature, APP systems are always depicted graphically with only one membrane that is the container for all the elements of an APP system. The skin membrane contains four applicable rules. The *out* tag represents that the target of products is out of the skin membrane. This simple example does not have practical applications and serves only to provide an idea of the functioning of the computation of a APP system

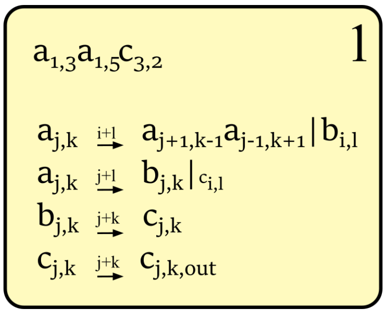

$a_{1,3}a_{1,5}c_{3,2}$
1
$a_{j,k} \xrightarrow{i+l} a_{j+1,k-1}a_{j-1,k+1}|b_{i,l}$
$a_{j,k} \xrightarrow{j+l} b_{j,k}|c_{i,l}$
$b_{j,k} \xrightarrow{j+k} c_{j,k}$
$c_{j,k} \xrightarrow{j+k} c_{j,k,out}$

Figure 3.10: initial state Attributed Probabilistic P Systems

Computation
Because of the probabilistic nature of APP systems, there are many different paths of computation a single APP system is capable of, leading to different results. The following is one possible path of computation for the considered APP system.

Step 1
From the initial configuration the skin membrane has inside the following elements: $a_{(1,3)}a_{(1,5)}c_{(3,2)}$. The absence of promoter b prevents the application of the rule $a_{(j,k)} \to a_{(j+1,k-1)}a_{(j-1,k+1)}|b_{(i,l)}$. The elements $a_{(1,3)}$ and $a_{(1,5)}$ are assigned to $a_{(j,k)} \to b_{(j,k)}|c_{(i,l)}$, the element $c_{(3,2)}$ is assigned to $c_{(j,k)} \to c_{(j,k,out)}$. Notice the maximally parallel behaviour of rule application leading to the same rule being applied twice during one step.

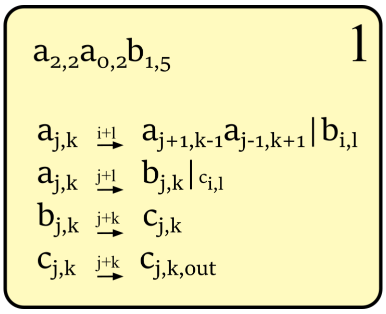
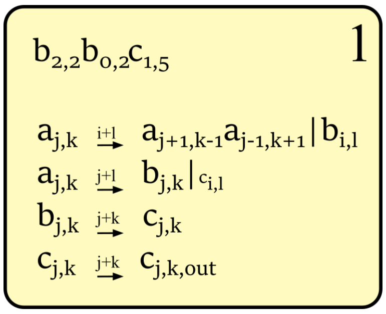

$a_{2,2}a_{0,2}b_{1,5}$
1
$a_{j,k} \xrightarrow{i+l} a_{j+1,k-1}a_{j-1,k+1}|b_{i,l}$
$a_{j,k} \xrightarrow{j+l} b_{j,k}|c_{i,l}$
$b_{j,k} \xrightarrow{j+k} c_{j,k}$
$c_{j,k} \xrightarrow{j+k} c_{j,k,out}$

Figure 3.11: after step 1 of computation

Step 2
As we can see in figure 3.11, the skin membrane has inside the following elements: $a_{2,2}a_{0,2}b_{1,5}$. The absence of promoter c prevents the application of the rule $a_{j,k} \to b_{j,k}|c_{i,l}$. The elements $a_{0,2}$ and $a_{2,2}$ are assigned to $a_{j,k} \to a_{j+1,k-1}a_{j-1,k+1}$, while $b_{1,5}$ is assigned to $b_{j,k} \to c_{j,k}$.

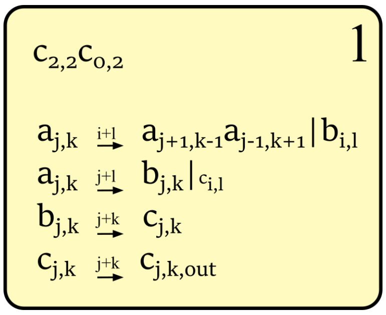


$b_{2,2}b_{0,2}c_{1,5}$
1
$a_{j,k} \xrightarrow{i+l} a_{j+1,k-1}a_{j-1,k+1} | b_{i,l}$
$a_{j,k} \xrightarrow{j+l} b_{j,k} | c_{i,l}$
$b_{j,k} \xrightarrow{j+k} c_{j,k}$
$c_{j,k} \xrightarrow{j+k} c_{j,k,out}$

Figure 3.12: after step 2 of computation

Step 3

In figure 3.12, the skin membrane has inside the following elements: $b_{2,2}b_{0,2}c_{1,5}$. The elements $b_{2,2}$ and $b_{0,2}$ are assigned to $b_{j,k} \to c_{j,k}$, the element $c_{1,5}$ is assigned to $c_{j,k} \to c_{j,k,out}$.



$c_{2,2}c_{0,2}$
1
$a_{j,k} \xrightarrow{i+l} a_{j+1,k-1}a_{j-1,k+1} | b_{i,l}$
$a_{j,k} \xrightarrow{j+l} b_{j,k} | c_{i,l}$
$b_{j,k} \xrightarrow{j+k} c_{j,k}$
$c_{j,k} \xrightarrow{j+k} c_{j,k,out}$

Figure 3.13: after step 3 of computation

Step 4

In figure 3.13 we can see the state after three steps. The computation halts with the output of $c_{2,2}$ and $c_{0,2}$ from the skin membrane.

Chapter 4

Multilevel Attributed Probabilistic P Systems

In this chapter we propose a new formalism aimed at representing social interactions between groups of individuals. We intend to use membranes to represent groups of individuals grouped by their age, sex, spatial position or other characteristics of social nature. To this end we decide not to use *Flat P systems*, used in the case of *APP systems.*, and we use the hierarchical membranes structure of the original paradigm of Păun.

We intend to study interactions between groups, then we must also redefine the semantics of rules to ensure that they can operate not only on the usual elements, but also on the membranes, to have the opportunity to create them, destroy them and change them.

4.1 Definition of MAPP systems

We intend to extend APP systems adding attributes to the membranes. As already happens for objects, these attributes are used and modified by the rules. Some of these attributes are updated by update functions. For example an update function can calculate the number of elements in a membrane and update the value of a membrane attribute that serves as a counter.

We present a variant of P systems called *Multilevel Attributed Probabilistic P Systems* (MAPP systems). MAPP systems are an extension of APP systems where we include the following additional features:

- an ordered finite set of membranes Γ. The membranes are sorted depending on which one can incorporate the others. The first one, named

Env can contain all the others but itself, and so on, the last in order cannot contain other membranes but only simple objects.

- a set of domains D_Γ that defines the types of the attributes associated with membranes.

- A set of rules is associated to each membrane. Each rule is composed by a multiset of reagents and promoters, a multiset of products, a multiset of the promoters and a rating function. Reagents and products are multisets of both membranes and objects. Variables can be uses to represent attribute values in an abstract way. The multiset of reagents cannot be empty. A rating function takes as input the attributes of both promoters and reagents and returns as output a positive number used as weight of the rule.

- a set of update functions U in one to one correspondence with the attributes of the membranes.Each function takes as input all membrane's attributes, the inner state of the membrane and returns as an output a new value for its corresponding attribute.

- Finally, the computational step is no longer parallel, a membrane to be able to calculate its new internal state must wait until all the states of contained membranes have been calculated. We define a recursive computation of the membranes that rises from the inner to the outer membrane which describes the environment of the system. In the next section we show in detail this formalism that incorporates features of APP such as probabilistic elements and promoters.

4.1.1 Formal definition

Definition 4.1 (MAPPS). *A Multilevel Attributed Probabilistic P System is a tuple* $\langle \Gamma, \Sigma, arity, D_\Sigma, D_\Gamma, \omega_0, R, U\rangle$
that consists of the following elements:

- $\Gamma = \{Env\} \cup \{\gamma_1, .., \gamma_n\}$ *is an ordered finite alphabet of symbols, representing membranes of our system. Membranes* $\gamma_1, .., \gamma_n$ *are assumed to be ordered and the order influences the containment of one membrane into another. In particular, a membrane* γ_i *can be contained (nested) into another membrane* γ_j *only if* $j < i$*. The Env (enviroment) element is called skin membrane, it cannot be contained by any element of* Γ *while it can contain any element of* Γ*.*

- $\Sigma = \{\sigma_1, .., \sigma_m\} \cup \{\epsilon\}$ *is an ordered finite alphabet of symbols, representing standard objects.*
  The special symbol ϵ *represents no objects and will be used to define the inner state of an empty membrane.*

- *arity:* $\Sigma \cup \Gamma \to \mathbb{N}$ *is a function which for each* $a_i \in \Sigma \cup \Gamma$ *gives the lenght of the tuple* D_{a_i} *that describes the attributes of* a_i*.*

- $D_\Gamma = \{D_{\gamma_1}, \dots, D_{\gamma_n}\}$ *Ordered set of domains, in correspondence with elements of set* $\{\gamma_1, .., \gamma_n\}$*. Notice that Env has no attributes therefore does not exist a* D_{Env}*.*
  EachD_{γ_i} *is a set of tuples,* $D_{\gamma_i} = I_{\gamma_i,1} \times \cdots \times I_{\gamma_i,arity(\gamma_i)}$*, where each* $I_{\gamma_i,j}$ *is a (possibly infinite) set of values for the j-th attribute of* γ_i*. The sequence* D_{γ_i} *is called the set of attributes of* γ_i*.*

- $D_\Sigma = \{D_{\sigma_1}, \dots, D_{\sigma_m}\}$ *Ordered set of domains, in correspondence with elements of set* Σ*.*
  Each D_{σ_i} *is a set of tuples,* $D_{\sigma_i} = I_{\sigma_i,1} \times \cdots \times I_{\sigma_i,arity(\sigma_i)}$*, where each* $I_{\sigma_i,j}$ *is a (possibly infinite) set of values for the j-th attribute of* σ_i*. The sequence* D_{σ_i} *is called the set of attributes of* σ_i*.*

- *The initial state* $\omega_0 = (Env, s)$ *is the state of the system at the beginning of the computation. The state of a generic membrane* $\gamma_i \in \Gamma_{i,j}$ *is a multiset over* S_i=$\{\langle\gamma_j, \overline{d}, \overline{s}\rangle | j > i, \gamma_j \in \Gamma, \overline{d} \in D_{\Gamma_j}, \overline{s} \in S_j^*\} \cup \{\langle\sigma, \overline{d}\rangle | \sigma \in \Sigma, \overline{d} \in D_{\Sigma_j}\} \cup \{\epsilon\}$ *with* $i, j \in \{0, \dots, n\}$*. The state s is the state of membrane Env and it is a multiset over* S_0*. It represents membranes and objects contained in the internal state of Env.*

- *The set* $R = \{R_{\gamma_1}, \dots, R_{\gamma_n}\}$ *is an ordered set of sets of evolution rules, in corrispondence with the elements of* Γ*. For each element* $\gamma_i \in \Gamma$ *there is an associated set of rules* $R_{\gamma_i} \in R$*. Evolution rules are defined in def. 4.2*

- *The set* $U = \{U_{\gamma_1}, \dots, U_{\gamma_n}\}$ *is an ordered set of sets of functions (called update functions) in correspondence with elements of* Γ*. Each set of functions* U_{γ_i} *is composed by* $arity(\gamma_i)$ *functions.*
  $U_{\gamma_i} = \{U_{\gamma_i,1}, \dots, U_{\gamma_i,arity(\gamma_i)}\}$ *The generic function* $U_{\gamma_i,j}$ *belonging to* U_{γ_i} *is typed as:*

$$U_{\gamma_i,j} : S_i^* \times D_{\Gamma_i} \rightarrow \quad D_{\Gamma_{i,j}}$$

The generic internal state of a membrane γ_i is a multiset over the set:

S_i=$\{\langle\gamma_j, \overline{d}, \overline{s}\rangle | j > i, \gamma_j \in \Gamma, \overline{d} \in D_{\Gamma_j}, \overline{s} \in S_j^*\} \cup \{\langle\sigma, \overline{d}\rangle | \sigma \in \Sigma, \overline{d} \in D_{\Sigma_j}\} \cup \{\epsilon\}$
with $i, j \in \{0, \dots, n\}$
The generic element of the internal state of membrane γ_i is part of the resulting union of three sets.
An element of first set is a tuple $\langle\gamma_j, \overline{d}, \overline{s}\rangle$ and describes a membrane with its attributes and its internal state, with the restriction $j > i$.
An element of second set is formed by the terminal symbols belonging to Σ with an instance of its attributes.
The last set consists of the singlet ϵ that is used to represent the absence of elements within a membrane.

With this definition of internal state we define the generic state of the system $\omega = (Env, s)$ composed by a couple consisting of the symbol Env representing the skin membrane, and its internal state s.

Each individual set of functions U_{γ_i} consists of a number of functions equivalent to the number of attributes of the membrane associated with it, then $|U_{\gamma_i}| = arity(\gamma_i)$. If from the set of update functions of the i-th membrane we consider the j-th update function this will take as input one internal state of level i and one istantiation of the attribute domain of the membrane Γ_i and will give as output one value of type $D_{\Gamma_{i,j}}$. For example from the j-th set of values $I_{\gamma_i,j}$ from $D_{\Gamma_i} = \{I_{\gamma_i,1} \times \dots \times I_{\gamma_i,j} \times \dots \times I_{\gamma_i,arity(\gamma_i)}\}$.

In rules definition are present variables and expressions, as in the formalism, membranes and objects are attributed and the evolution rules concur to change attributes values. Variables and expressions are used to write rules applicable for different values of attributes

Given a set of variables V we define:

$S_j^{*V} = V \cup \{\varnothing\}$
$D_{\Gamma_j}^V = (V \cup I_{\gamma_j,1}) \times \dots \times (V \cup I_{\gamma_j,arity(\gamma_j)})$
$D_{\Sigma_l}^V = (V \cup I_{\sigma_l,1}) \times \dots \times (V \cup I_{\sigma_l,arity(\sigma_l)})$

$E_{\Gamma_j}^V = Exp(V \cup I_{\Gamma_j(1)}) \times \dots \times Exp(V \cup I_{\Gamma_j(arity(\gamma_j))})$
$E_{\Sigma_l}^V = Exp(V \cup I_{\sigma_l,1}) \times \dots \times Exp(V \cup I_{\sigma_l,arity(\sigma_l)})$

where $Exp(V \cup I_j)$ denotes the set of well-typed expressions built from operators, variables V, and values of I_j. $Exp(V \cup S_j)$ denotes a well typed expression built on strings operator "+", variables V and elements and strings of set S_j^V.

Definition 4.2. *Given a set of variables V, an evolution rule associated with membrane γ_i of a MAPPS has the form:*

$$u_V \xrightarrow{f} v_V|_{pr_V}$$

where:

- $u_V \in (\{\langle\gamma_j, d_j, s\rangle|\ j > i, \gamma_i \in \Gamma, d_j \in D_{\Gamma_j}^V, s \in S_j^{*V}\} \cup \{\langle\sigma_l, d_l\rangle|\sigma_l \in \Sigma, d_l \in D_{\Sigma_l}^V\})^+$ *is a non empty multiset of objects each one with its own attributes made explicit by values or by variables.*

- $pr_V \in (\{\langle\gamma_j, d_j, s\rangle|\gamma_i \in \Gamma, s \in S_j^{*V}, d_j \in D_{\Gamma_j}^V, j > i\} \cup \{\langle\sigma_l, d_l\rangle|\sigma_l \in \Sigma, d_l \in D_{\Sigma_l}^V\})^*$ *is a possibly empty multiset of objects with its own attributes explicited by values or by variables.*

- $v_V \in$*((* $\{\ \langle\gamma_j, d_j, s\rangle|\gamma_i \in \Gamma, d_j \in E_{\Gamma_j}^V, s \in ES_j^{*V}, j > i\} \cup \{\langle\sigma_l, d_l\rangle|\sigma_l \in \Sigma, d_l \in E_{\Sigma_l}^V\} \times \{out, \_\}) \cup (\{\langle\gamma_k, d_k, s\rangle|\gamma_i \in \Gamma, s \in S_k^*, d_k \in E_{\Gamma_k}^V, k > j\} \cup \{\langle\sigma_l, d_l\rangle|\sigma_l \in \Sigma, d_l \in E_{\Sigma_l}^V\} \times \{in(\langle\gamma_j, d, s\rangle\}))^*$ *where:* $j > i$, $Vars(v_V) \subseteq Vars(u_V) \cup Vars(pr_V)$ *and* $|\langle\sigma, d\rangle| + |\langle\gamma, d, s\rangle| \leq 1$ *with* $\langle\sigma, d\rangle, \langle\gamma, d, s\rangle \in v_V$

- f *is a rate function typed as:*
  $f : S_j^+ \times S_j^* \to \mathbb{R}^{\geq 0}$
  where S_j is the state of membrane γ_j as defined in def. 4.1. The domain is composed by reagent (S_j^+) and the promoters (S_j^*), the output is a positive real number (the weight of the rules).

Multiset v_V is a possibly empty multiset of pairs consisting of one object with its own attributes explicited by value or functions which can take as input attributes both from u_V and pr_V and a flag from the set $\{in(\langle\gamma_i, d, s\rangle), out, \_\}$ where $\langle\gamma_i, d, s\rangle \in u_V$. For the pairs flagged by "*out*" or "_" the membranes of the multiset v_V have an index that is greater than i, where γ_i is the membrane that owns the rule. Otherwise for the pair flagged by $in(\langle\gamma_{j>i}, d, s\rangle)$ the membranes have an index that is greater than j.

4.2 Semantics, formal definition

We define the semantics of MAPP systems using inference rules. We have seen that the system elements can be membranes or objects. A membrane is identified by a tuple $\langle \gamma_i, d, s\rangle$. An object is identified by a tuple $\langle \sigma, d\rangle$. In what follows , especially in the examples, for greater readability, we could denote an (attributed) object $\langle \sigma, d\rangle$ as $\sigma_{(d)}$. We represent the structure of the internal state of *Env* as a tree where each node is a membrane, its children are the membranes contained in its internal state. The computing of internal state of each membrane is not in parallel, as in regular P System, but sequentially in the order induced a postorder traversal of the tree. One by one the states of the membranes are computed back to the root of the tree represented by *Env* that concludes the complete computation of the new state of the MAPPS.

The computation of the new state of a membrane is managed by the single internal step rule, A single internal step consists of the following operations:
- Get the objects sent out by inner membranes

- Choose a maximal multiset of rules considering internal membranes as reactants ready to be used. Then apply in parallel the chosen multiset of rules to the current state of the membrane.

- Apply the update functions on the inner state and attributes of the membrane to recalculate the value of its attributes

After these three steps, the membrane has a new internal state and set of attributes. Here below we describe, one by one, the inference rules that describe the computative process of the entire system.

In the first three inference rules we describe the internal computative step of a single membrane. In order to achieve our purpose, we define three multiset: w' that indicates the multiset of objects ready to be used by rules, $\overline{w}'$ the multiset of products created by the rules application and w_{out} the multiset of products created by rules with ”out” flag.

The evolution of the internal state of a generic membrane γ_j is a sequence of probabilistic maximally parallel steps. We formally define the semantics of that computation as a transition relation in the style of [7]. The transition relation is $\xrightarrow{r_i,p}_{\overline{R}}$ where $\overline{R}$ is the multiset of applied rules,r_i is a generic rule belonging to $\overline{R}$ and $p$is the probability that r_i can be applied. In each step a maximal multiset of evolution rule instances is selected and applied as described by the following semantic rules, in this rules are not present variables and promoters because we are working on applied rules where variables are instantiated and the presence of promoters has already been verified:

($single\ rule\ application$)

$$\frac{r_i{=}u\xrightarrow{k}v_{(\_)}+v_{(out)}\in\overline{R}\quad u\subseteq w'\quad K{=}\{\!|\ k'|u'\xrightarrow{k'}v'\in\overline{R},u'\subseteq w'\ |\!\}\quad p{=}k/\sum_{k'\in K}k'}{(w',\overline{w}',w_{out})\xrightarrow{r_i,p}_{\overline{R}}(w'-u,\overline{w}'+v_{(\_)},w_{out}+v_{(out)})}$$

single rule application - the rule application describes the application of a single rule that is associated with a probability p. The rule is applied starting from an internal state $(w', \overline{w}, w_{out})$. The multiset $v_{(\_)}$ subtracts reagents from w' and adds products to $\overline{w}$. The multiset $v_{(out)}$ subtracts reagents from w' but adds products to w_{out} to be sent outside the membrane. The probability p is evaluated by normalizing the weight of the rule r_i compared to the weights of all the applicable rules of the set R_{γ_j} to reageants in w'.

($single\,rule\,application\,with\,in\,flag$)

$$\frac{\begin{array}{r} r_i = u \xrightarrow{k} v_{(\_)} + v_{(out)} + \sigma_{in(\gamma_j)} \in \overline{R}u \subseteq w' \quad \langle\gamma_j, d, s\rangle \in w' \\ w'_{\gamma_j} = \{\!|\langle\gamma_j, d', s'\rangle \in w'|d' \in D_{\Gamma_j}, s' \in S_j|\!\} \\ K = \{\!|\ k'|u' \xrightarrow{k'} v' \in \overline{R}, u' \subseteq w'\ |\!\} \quad p = k/\sum\limits_{k'\in K} k' \cdot \dfrac{|w'|_{\langle\gamma_j,d,s\rangle}}{|w'_{\gamma_j}|} \end{array}}{(w',\overline{w}',w_{out})\xrightarrow{r_i,p}_{\overline{R}}(w'-u-\{\!|\langle\gamma_j,\sigma,s\rangle\langle\gamma_j,\sigma,s+\{\!|\sigma|\!\}\rangle|\!\},\overline{w}'+v_{(\_)}+\{\!|\sigma|\!\},w_{out}+v_{(out)})}$$

single rule application with in flag - describes rules where products are sent into another membrane. The target membrane must be among the reagents. The rule definition ensures us that it is not possible to send into a membrane another membrane of higher or of the same level

These two single rule applications are similar to those of APP systems with the difference that these sets of rules are applied inside a membrane j, then R_{γ_j} is the set of rules related to the j-th element of the set Γ. K is the set of weights associated to all the applicable rules. The probability is given by the weight k of a single rule divided by the sum of all the applicable weights in K multiplied by $\frac{|w'|_{\langle\gamma_j,d,s\rangle}}{|w'_{\gamma_j}|}$. Where |w'|$_{\langle\gamma_j,d,s\rangle}$ is the number of membranes $\langle\gamma_j, d, s\rangle$ present in w' while $|w'_{\gamma_j}|$ is the number of elements in w'_{γ_j}

($single\,rule\,sequence\,rule$)

$$\frac{(w',\overline{w}',w'_{out})\xrightarrow{r_i,p}_{\overline{R}}(w'',\overline{w}'',w''_{out})}{(w',\overline{w}',w'_{out})\overset{[r_i],p^+}{\longrightarrow}_{\overline{R}}(w'',\overline{w}'',w''_{out})}$$

single rule sequence rule - describes the application of a sequence of rules composed by a single rule. The sequence is associated with the probability of the single rule. The set of rules of our interest is limited to those visible within a membrane of the type corresponding to the j-th element of Γ. The rule, conditioned by the respective probability, makes a transition from (w', $\overline{w}'$, w'_{out}) to (w'', $\overline{w}''$, w''_{out}).

In the next two inference rules, it is present the transition relation $\xrightarrow{\overline{r},\overline{p}^+}_{\overline{R}}$ where the sequence $\overline{r}$ represents the multiset of the instances of rules.

($multiple\,rules\,sequence$)

$$\frac{(w',\overline{w}',w'_o)\xrightarrow{r_i,p_i}_{\overline{R}}(w'',\overline{w}'',w''_o) \quad (w'',\overline{w}'',w''_o)\xrightarrow{\overline{r},\overline{p}^+}_{\overline{R}}(w''',\overline{w}''',w'''_o)}{(w',\overline{w}',w'_o)\overset{\overline{r}@[r_i],p_i\cdot\overline{p}^+}{\longrightarrow}_{\overline{R}}(w''',\overline{w}''',w'''_o)}$$

multiple rules sequence - describes the application of a sequence of rules using as basic case the *(single rule sequence)*, the probability of the sequence is the product of the individual probabilities of the rules of the sequence. Our multiple rules sequence is an update of an APP Systems rule, Also here the set of rules is limited to R_{γ_j} In our definition $[r_i]$ denotes the sequence composed of the single element r_i, and @ denotes the concatenation of sequences.

($single\,membrane\,internal\,step\,rule$)

$$\frac{(s,\varnothing,\varnothing)\xrightarrow{\overline{r},\overline{p}^+}_{R(s)_d}(w',\overline{w'},w'_{out}) \quad (w',\overline{w'},w'_{out})\not\rightarrow_{R_{\gamma_j}(s)_d-}}{\langle\gamma_j,d,s,o\rangle\xrightarrow{\overline{r},\overline{p}}_{R_{\gamma_j}}\langle\gamma_j,d,w'+\overline{w}',w'_{out}\rangle}$$

single membrane internal step rule Given a membrane internal state, s, the *(single membrane internal step rule)* describes the evolution in a new membrane internal state by the $\overset{\overline{r},\,\overline{p}}{\to} R_{\gamma_j}$ relation, where $\overline{p}$ is the probability of the transition, and $\overline{r}$ is the sequence of applied rules. (single membrane internal step rule) invokes $(s,\varnothing,\varnothing) \xrightarrow{\overline{r},\,\overline{p}^{\,+}}_{R(s)_d} (w',\overline{w}',w_{out})$ where $R(s)_d$ is the set of applicable rules in the state s and with membrane attribute d, with their weights, namely:

$$R(s)_d = \left\{ u\lambda \xrightarrow{f(s)} v\lambda \;\middle|\; u \xrightarrow{f} v \mid_{pr} \in R_{\gamma_j},\ \exists\lambda.\, u\lambda \subseteq s \wedge pr\lambda \subseteq s \right\}$$

where (i) $\lambda : V \to flat(D_{a_1}) \cup \ldots \cup flat(D_{a_n})$, with $flat(D_{a_i}) = I_{a_1} \cup \ldots \cup I_{a_{arity(a_i)}}$, for all $a_i \in \Gamma \cup \Sigma$; (ii) $u_{\mathrm{V}}\lambda$, $pr_{\mathrm{V}}\lambda$ and $d_{\mathrm{V}}\lambda$ are well-typed multiset obtained by substituting values for variables in u_{V}, pr_{V} and d_{V} according to λ; and (iii) $v_{\mathrm{V}}\lambda$ is the well-typed multiset obtained by evaluating the expressions in v_{V} under the substitution λ.
A transition $(w',\overline{w}',w_{out}) \xrightarrow{r_i,\,p}_R (w'-u,\overline{w}'+v,w_{out}+v_{out})$ corresponds to the application of a single rule. When a rule is selected, its application consists in removing its reactants from w' and adding its products to $\overline{w}'$ or w_{out} according to the flags of the products. The two multiset $\overline{w}'$ and w_{out} will collect all products of all applied rules. Note that $R(s)$ takes into account that each rule is applied with respect to the weights of the rules computed

in the initial state s. Moreover, $R(s)$ contains only the rules the promoters of which are present in the initial state s ($pr_{\mathrm{V}}\lambda \subseteq s$). Once objects in w' are such that no further rule in $R(s)$ can be applied to them, by (single membrane step rule) the new membrane is $(\gamma_j, d, w' + \overline{w}'$,w$_{out})$ (where w' are the unused objects and $\overline{w}'$ are the new products).

(*update*)

$$\frac{\forall i \in \{1 ... arity(\gamma_j)\} \quad U_{\gamma_j,i}(d,s)=d'_i}{\langle \gamma_j, d, s, o\rangle \overset{update(d,s)_j}{\longrightarrow} \langle \gamma_j, (d'_1, ..., d'_{arity(\gamma_j)}), s, o\rangle}$$

update - The update rule is a feature not present in APP systems, here we obtain a new domain of attributes d' from previous attributes and state of γ_j, thanks to a set of functions that change the value of each attribute

(*getting on rule*)

$$\frac{s'=s-\biguplus_{\langle\overline{\gamma},\overline{d},\overline{s},\overline{o}\rangle\in s}\langle\overline{\gamma},\overline{d},\overline{s},\overline{o}\rangle+\biguplus_{\langle\overline{\gamma},\overline{d},\overline{s},\overline{o}\rangle\in s}\langle\overline{\gamma},\overline{d},\overline{s},\varnothing\rangle+\biguplus_{\langle\overline{\gamma},\overline{d},\overline{s},\overline{o}\rangle\in s}\overline{o}}{\langle\gamma_j,d,s,o\rangle\overset{gettingon(\gamma)}{\longrightarrow}\langle\gamma_j,d,s',o\rangle}$$

getting on rule - The *(getting on rule)* applied on a target membrane searches all membranes in its internal state, then, for each membrane found, keeps the elements present in o and adds them to internal state (s) of target membrane. Moreover this rule makes o empty for each internal membrane. The new state of target membrane s' is obtained subtracting the union of all internal membranes $[-\biguplus_{(\overline{\gamma},\overline{d},\overline{s},\overline{o})\in s}(\overline{\gamma},\overline{d},\overline{s},\overline{o})]$, adding the union of all membranes where o has been emptied $[+\biguplus_{(\overline{\gamma},\overline{d},\overline{s},\overline{o})\in s}(\overline{\gamma},\overline{d},\overline{s},\varnothing)]$and adding the union of all elements taken from the o multiset of internal membranes $[+\biguplus_{(\overline{\gamma},\overline{d},\overline{s},\overline{o})\in s}\overline{o}]$.

(*single membrane steprule*)

$$\frac{\begin{array}{c}\langle\gamma_j, d, s, \varnothing\rangle \overset{gettingon(\gamma_j)}{\longrightarrow} \langle\gamma_j, d, s', \varnothing\rangle \\ \langle\gamma_j, d, s', \varnothing\rangle \overset{\overline{r},\overline{p}}{\longrightarrow}_{R_{\Gamma_j}} \langle\gamma_j, d, s'', o\rangle \\ \langle\gamma_j, d, s'', o\rangle \overset{update(d,s'')_j}{\longrightarrow} \langle\gamma_j, d', s'', o\rangle\end{array}}{\langle\gamma_j,d,s,\varnothing\rangle\overset{\overline{r},\overline{p},update(d,s')_j}{\Longrightarrow}\langle\gamma_j,d',s'',o\rangle}$$

single membrane step rule - describes a composition of the getting on rules ,an internal computational step and a following application of update functions.
These three steps are performed in the correct order in a membrane element. At the end of this transition the object membrane will have a new internal state, updated attribute values and a new multiset of elements o. The membrane is now ready for the computational step at upper level.

$(recursive\, membrane\, computational\, step\, rules)$

(1)

$$\frac{\sigma\in\Sigma \;\; d\in D_\Sigma}{\langle\sigma,d\rangle \Rrightarrow^1 \langle\sigma,d\rangle}$$

(2)

$$\frac{s = x_1.x_2.\ldots.x_l \quad \forall i, 1<i<l, x_i \overset{p_i}{\Rrightarrow} \overline{x}_i \qquad \overline{s} = \overline{x}_1.\overline{x}_2.\ldots.\overline{x}_m \quad \langle\gamma_j, d, \overline{s}, o\rangle \overset{\overline{r},\overline{p},update(d,s')_j}{\Longrightarrow} \langle\gamma_j, d^*, s^*, o^*\rangle}{\langle\gamma_j,d,s',o'\rangle \overset{\overline{p}\cdot\prod_{i\in\{1\ldots l\}} p_i}{\Rrightarrow} \langle\gamma_j,d^*,s^*,o^*\rangle}$$

(3)

$$\frac{s = x_1.x_2.\ldots.x_l \quad \forall i, 1<i<l, x_i \overset{p_i}{\Rrightarrow} \overline{x}_i \quad \overline{s} = \overline{x}_1.\overline{x}_2.\ldots.\overline{x}_m \qquad \langle\gamma_j, \varnothing, \overline{s}, \varnothing\rangle \xrightarrow{gettingon(\gamma_j)} \langle\gamma_j, \varnothing, \overline{s}', \varnothing\rangle \qquad \langle\gamma_j, \varnothing, \overline{s}', \varnothing\rangle \xrightarrow{\overline{r},\overline{p}}_{R_{\Gamma_j}} \langle\gamma_j, \varnothing, s^*, o\rangle}{(Env,s) \overset{\overline{p}\cdot\prod_{i\in\{1\ldots l\}} p_i}{\Rrightarrow} (Env,s^*)}$$

where

$$x_i \in \{\langle\gamma_k, \overline{d}, \overline{s}, o\rangle | k > i, \gamma_k \in \Gamma, \overline{d} \in D_{\Gamma_k}, \overline{s} \in S_k^*, o \in S_k^*\} \cup \{\langle\sigma_k, \overline{d}\rangle | \sigma_k \in \Sigma, \overline{d} \in D_{\Sigma_k}\}$$

recursive membrane computational step rules - The three recursive membrane computational step rules are applied to all elements of the systems: membranes represented by $\langle\gamma, d, s, o\rangle$, objects represented by $\langle\sigma, d\rangle$, *Env* represented by (Env, s). The rule (1) tell us that the terminal elements are unaltered by this transformation. The (1) represents the basic rule of the recursion. The rule (2) is applied to membranes. The rule (2) is recursively called to each elements (x_i) of internal state (s) of the membrane to obtain a new state $\overline{s}$ and then the single membrane step rule is applied over the tuple $\langle\gamma_j, d, \overline{s}, o\rangle$ to obtain $\langle\gamma_j, d^*, \overline{s}^*, o^*\rangle$. The rule (3) describes the application of the rule to *Env* to obtain a new step of entire system. Instead to apply a full single membrane step rule to *Env*, the rule (3) applies to *Env* only the gettingon and the *single membrane internal step rule*. The update rules is not required because *Env* has not attributes. In Both the rules (2) and (3) the states s and $\overline{s}$ are represented by sequence of x_i elements. Each x_i can be a membrane represented by a tuple $\langle\gamma_k, \overline{d}, \overline{s}\rangle$ or a terminal element represented by $\langle\sigma_k, \overline{d}\rangle$.

In the description of these rules we use the following sets , operator and conventions:
$R_{\gamma_n}(s) = \{u_V\phi \overset{f(s)}{\rightarrow} v_V\phi \mid u_V \overset{f}{\rightarrow} v_V|_{pr_V} \in R_{\gamma_n}()\exists\phi.u_V \subseteq s \wedge pr_V\phi \subseteq s\}$

where $\phi : V \rightarrow flat(D_{a_1} \cup \cdots \cup flat(D_{a_n})$, with $flat(D_{a_i}) = I_{a1} \cup \cdots \cup I_{arity(a_i)}$, for all a$_i \in A$; u$_V\phi(u_V \in \Sigma_V^*)$ is the well-typed multiset obtained by substituting values for variables in u_V according to ϕ; and $(v_V \in \Sigma_{EV}^*)$

The probability that a membrane $\langle\gamma_j, d, s\rangle$ makes a transition to $\langle\gamma_j, d, s^*\rangle$ is $p = \overline{p} \cdot \prod p_i \forall i = \{1 \ldots l\}$ If the membrane contains no other membrane but only objects $p_i = 1$ then $p = \overline{p}$. Where $\overline{p}$ is a probability given by the normalization of weight k and hence the sum of all alternative transitions is 1. If the membrane contains only membranes which contain only objects, the probability is $p = \overline{p}' \cdot \prod p_i'$ where p_i' are the probability $\overline{p}$ of internal membranes , and $\overline{p} isagainobtainedbynormalizationofruleweights.Theprobabilitiesp'_i$ are indipendent each others. Therefore $\overline{p}' \cdot \prod p_i'$ is a product of probabilities that is a probability itself. We can continue in this way leveling up from the innermost membrane to Env and say that (Env, s) makes a transition to (Env, s^*) with probability $p'' = \overline{p''} \cdot \prod p_i''$ where p_i'' is still a probability.

4.3 MAPPS a simple example

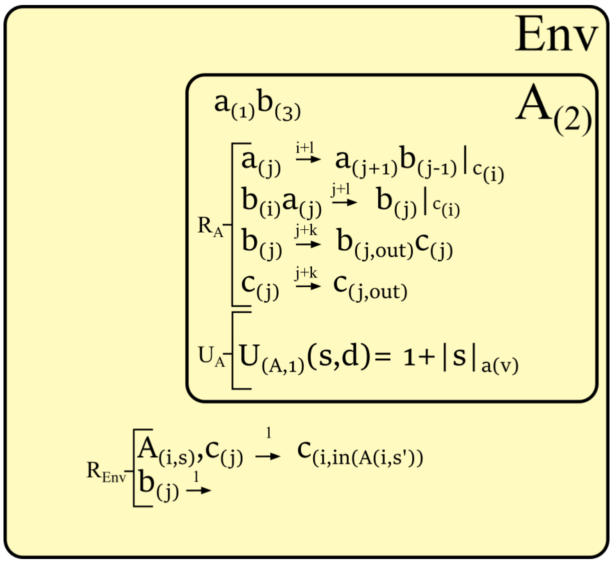


Figure 4.1: graphical representation of a P system

The image shown in figure 4.1 depicts the initial state of a MAPP system with two membranes. The outermost membrane, is the Env of this MAPP system and contains two rules. The innermost membrane, contains the multiset of symbols $a_{(1)}c_{(3)}$ and four rules. Membrane A as an attribute and its value is 2. There is only one update function present in A. The *out* tag represents that the target of products is out of the membrane. The *in* tag represents that the target of products is an internal membrane. In this initial state only the rules of the most internal membrane are applicable: there are no symbols outside of that membrane. However, during the evolution of the system, as objects are passed between membranes, the rules in other membranes will become active. This simple example does not have practical applications and serves only to provide an idea of the functioning of the computation of a MAPP system. The initial state of a MAPP system is represented as below:

MAPPS $= \langle \Gamma, \Sigma, arity, D_\Gamma, D_\Sigma, w_0, R, U \rangle$
$where:$
$\Gamma = \{A\}$
$\Sigma = \{a, b, c, d\}$
$D_\Gamma = \{D_A\}$
$D_\Sigma = \{D_a, D_b, D_c, D_d\}$
$\omega_0 = (Env, \{\!|(A, (2), a_{(1)}b_{(3)})|\!\})$
$R = \{R_{Env}, R_A\}$

$U = \{U_A\}$

Computation

Because of the probabilistic nature of MAPP systems, there are many different paths of computation a single MAPP system is capable of, leading to different results. The following is one possible path of computation for the considered MAPP system.

Step 1

From the initial configuration the most internal membrane has inside the following elements: $a_{(1)}b_{(3)}$. Object $b_{(3)}$ is assigned to $b_{(j)} \xrightarrow{j+k} b_{(j,out)}, c_{(j)}$, while $a_{(1)}$ is not assigned to any rule because the only one that takes $a_{(j)}$ as reactant requires $c_{(i)}$ as promoter. At the and of the rule application, the update functions are applied. In this simple example the only update function is $1 + |s|_{a_{(v)}}$ that sum 1 to the number of elements a present into membrane A, and set $D_{(A,1)} = 2$.

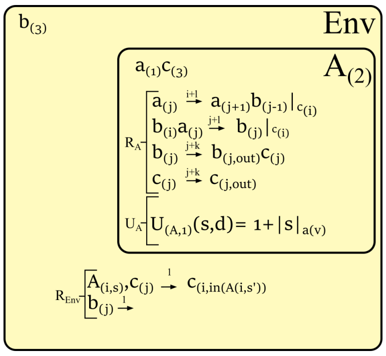

$b_{(3)}$
Env
$a_{(1)}c_{(3)}$
$A_{(2)}$
R_A
$a_{(j)} \xrightarrow{i+1} a_{(j+1)}b_{(j-1)}|_{c_{(i)}}$
$b_{(i)}a_{(j)} \xrightarrow{j+1} b_{(j)}|_{c_{(i)}}$
$b_{(j)} \xrightarrow{j+k} b_{(j,out)}c_{(j)}$
$c_{(j)} \xrightarrow{j+k} c_{(j,out)}$
U_A
$U_{(A,1)}(s,d) = 1+|s|_{a(v)}$
R_{Env}
$A_{(i,s)}, c_{(j)} \xrightarrow{1} c_{(i,in(A(i,s')))}$
$b_{(j)} \xrightarrow{1}$

Figure 4.2: after step 1 of computation

Step 2

As we can see in figure 4.2, the most internal membrane now contains: $a_{(1)}c_{(3)}$ while the Env contains $b_{(3)}$. Object $a_{(1)}$ is assigned to $a_{(j)} \xrightarrow{i+j} a_{(j+1)}, b_{(j-1)}|c_{(i)}$, while $c_{(3)}$ is assigned to $c_{(j)} \xrightarrow{j+k} c_{(j,out)}$, In Env, $b_{(3)}$ is assigned to $b_{(j)} \xrightarrow{1}$. We remark that the choice of rules to be applied is probabilistic. In this case however only one rule can be applied for each present reactant. At the end of the rules application, update functions are applied, $D_{(A,1)} = 2$.

Step 3

At this step, as shown in figure 4.3 we can see that the most internal membrane contains $a_{(2)}b_{(0)}$ while the Env contains $c_{(3)}$. The reactant $a_{(2)}$ is not

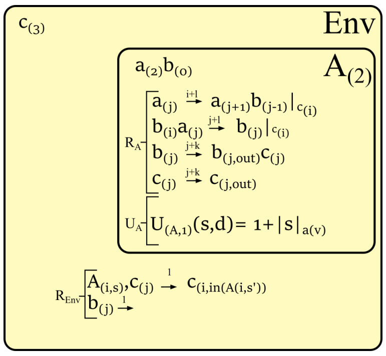


Figure 4.3: after step 2 of computation

assigned to any rules because the only one that takes $a_{(j)}$ as reactant requires $c_{(i)}$ as promoter. The reactant $b_{(0)}$ is assigned to $b_{(j)} \overset{j+k}{\rightarrow} b_{(j,out)}, c_{(j)}$. In Env, $c_{(3)}$ is assigned to$A_{i,s}c_{(j)} \xrightarrow{1} c_{(i,in(A_{(i,s')}))}$. At the and of the rules application, update functions are applied, $D_{(A,1)} = 2$.

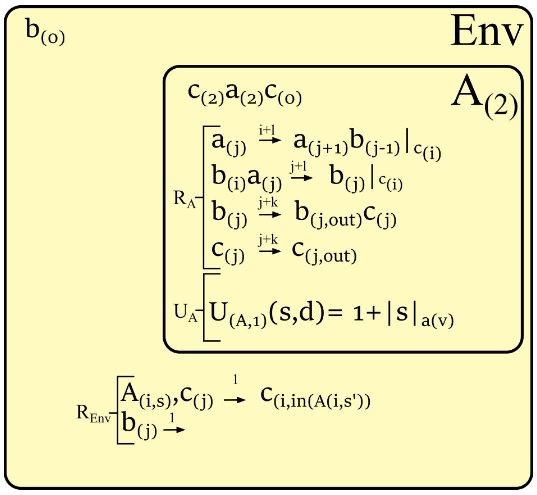


Figure 4.4: after step 3 of computationm

In figure 4.4 is shown the state after three steps. Notice that these few steps are intended as simple examples of computation, but the computation continues to proceed with the next steps.

4.4 MAPPS another example: Predator / Prey

This brief example is made to demonstrate the functioning of MAPP Systems on a classical case study of population dynamics: the predator/prey interaction. A predator is an organism that eats another organism. The prey is the organism which the predator eats. Some examples of predator and prey are lion and zebra, bear and fish, and fox and rabbit. In the following three subsections we show, the sets, the rules and the update functions of a predator/prey model, written with our formalism.

4.4.1 Sets

- $\Gamma \cup \{Env\} = \{Env, Loc, Predators\}$
  is the set of symbols of alphabet of membranes. It Includes *Env*, the external environment, *Loc*, the locations reachable by groups of predators, and *Predators*, a group of predators that hunt in packs.

- $D_\Gamma = \{D_{Loc}, D_{Predators}\}$
  is the set of domains of membranes as defined below.

  – $D_{Loc} = \{fecundity\}$ where $fecundity \in \mathbb{N}$ is an attribute used to indicate the quantity of food for preys produced in a certain location.

  – $D_{Predators} = \{members, supplies\}$ where $members \in \mathbb{N}$ and *supplies* $\in \mathbb{N}$. *members* represents the number of elements within a group, and supplies the number of supplies collected by a pack.

- $\Sigma = \{m, p, f, r, summer, winter, spring, autumn\}$ is the set of objects. Includes m,which is a member of the pack of predators, p representing a prey, f representing the food, r that is the source from which the food is spawned and four control elements, named *summer*, *spring autumn* and *winter* representing the cycle of the seasons. The use of a source from which the food is spawned prevents the situation in which prey eat all the food and there is no more possibility to reproduce it. The food reproducing speed is related to the fecundity of a location.

- $D_\Sigma = \{D_m, D_p, D_f, D_r, D_{summer}, D_{spring}, D_{winter}, D_{autumn}\}$ is the set of domains of objects as defined below.

  - $D_m = \{\epsilon\}$
  - $D_p = \{\epsilon\}$
  - $D_f = \{\epsilon\}$
  - $D_r = \{\epsilon\}$
  - $D_{summer} = \{month\}$ where $month \in \mathbb{N}$ Months are used to allow more iterations within the same season, so that *prey* reproduce themselves more than once.
  - $D_{spring} = \{month\}$ where $month \in \mathbb{N}$ Months are used to allow more iterations within the same season, so that *food* increases more than once.
  - $D_{winter} = \{month\}$ where $month \in \mathbb{N}$ Months are used to allow more iterations within the same season, so that *predators* reproduce themselves more than once.
  - $D_{autumn} = \{month\}$ where $month \in \mathbb{N}$ Months are used to allow more iterations within the same season, so that *predators* hunt more than once.

4.4.2 Rules

$$
\begin{aligned}
R_{Env} = \{ & \\
& autum_{month<3} \rightarrow autum_{month++} \\
& autum_{month=3} \rightarrow winter_{month=0} \\
& winter_{month<3} \rightarrow winter_{month++} \\
& winter_{month=3} \rightarrow spring_{month=0} \\
& spring_{month<3} \rightarrow spring_{month++} \\
& spring_{month=3} \rightarrow summer_{month=0} \\
& summer_{month<3} \rightarrow summer_{month++} \\
& summer_{month=3} \rightarrow autum_{month=0} \\
& \\
& Predators, Loc_{fecundity} \overset{fecundity}{\rightarrow} Loc, Predators_{in(Loc)}|_{winter} \\
\} &
\end{aligned}
$$

- In *Env* the seasons pass cyclically. Every four months, the season changes and the system pass from autumn to winter, from winter to spring, from spring to summer, from summer to autumn and so on

- During winter, predators move from environment (*Env*) to a location (*Loc*) according to the value of attribute *fecundity* of *Loc*.

$R_{Loc} = \{$
$autum_{month<4} \rightarrow autum_{month++}$
$autum_{month=4} \rightarrow winter_{month=0}$
$winter_{month<4} \rightarrow winter_{month++}$
$winter_{month=4} \rightarrow spring_{month=0}$
$spring_{month<4} \rightarrow spring_{month++}$
$spring_{month=4} \rightarrow summer_{month=0}$
$summer_{month<4} \rightarrow summer_{month++}$
$summer_{month=4} \rightarrow autum_{month=0}$

$Predators \rightarrow Predators_{out}|_{winter}$

$Predators_{members-supplies>0}, prey \rightarrow Predators_{supplies++}|_{autumn}$
$Predators_{members-supplies>0}, prey \rightarrow Predators_{supplies++}|_{autumn}$

$r \rightarrow r, f|spring, Loc_{fecundity>0}$
$r \rightarrow r, f, f|spring, Loc_{fecundity>1}$
$r \rightarrow r, f, f, f|spring, Loc_{fecundity>2}$
$r \rightarrow r, f, f, f, f|spring, Loc_{fecundity>3}$

$p, f \rightarrow p|_{summer}$
$p, f \rightarrow p, p|_{summer}$
$p, f, f \rightarrow p, p, p|_{summer}$
$p, f, f \rightarrow p, p, p, p|_{summer}$
$\}$

- In *Loc* the seasons pass cyclically. Every four months, the season changes and the system pass from autumn to winter, from winter to spring, from spring to summer, from summer to autumn and so on

- During winter, predators move from a location (*Loc*) to environment (*Env*), from there they will choose a new location.

- During autumn, the groups of predators eat the prey. Each eaten prey increases the number of supplies stored by the groups

- During spring, food grows according to the attribute *fecundity* of *Loc*.

- During summer the number of preys grows according to the quantity of food they eat.

$R_{Predators} = \{$
$autum_{month<4} \rightarrow autum_{month++}$
$autum_{month=4} \rightarrow winter_{month=0}$
$winter_{month<4} \rightarrow winter_{month++}$

$winter_{month=4} \rightarrow spring_{month=0}$
$spring_{month<4} \rightarrow spring_{month++}$
$spring_{month=4} \rightarrow summer_{month=0}$
$summer_{month<4} \rightarrow summer_{month++}$
$summer_{month=4} \rightarrow autum_{month=0}$

$m \rightarrow m, m, m | Predators_{(supplies/members)>0,75}, winter$
$m \rightarrow m, m | Predators_{(supplies/members)>0,5}, winter$
$m \rightarrow m | Predators_{(supplies/members)>0,25}, winter$
$m \rightarrow \_ | Predators_{(supplies/members)=0}, winter$
}

- In *Predators* the seasons pass cyclically. Every four months, the season changes and the system pass from autumn to winter, from winter to spring, from spring to summer, from summer to autumn and so on
- During the winter, predators number grows according to the stored supplies.

4.4.3 Functions

$U_{Env} = \{\epsilon\}$
$U_{Loc} = \{\epsilon\}$
$U_{Predators} = \{$
$U_{(Predators,members)}(s, d) = |s|_m$
}

The only nonempty set of update functions is of membrane *Predators*, where the attribute *members* is updated based on how many elements of type m are present within the membrane.

Example computation

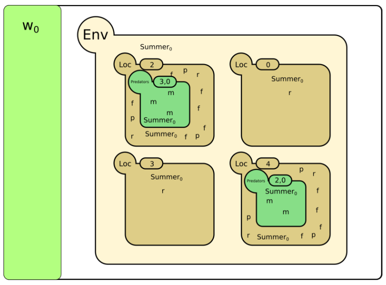


Figure 4.5: Initial state predator/prey

$$
\begin{aligned}
\omega_0 = (&Env, \{\!|\\
&\quad (Loc, (2), \{\!| f, f, f, f, f, f, p, p, p, r, r, summer_0,\\
&\quad\quad\quad (Predators, (3, 0), \{\!| m, m, m, summer_0 |\!\})|\!\}),\\
&\quad (Loc, (0), \{\!| summer_o, r |\!\}),\\
&\quad (Loc, (3), \{\!| summer_o |\!\}),\\
&\quad (Loc, (4), \{\!| f, f, f, f, p, p, p, r, r, summer_0,\\
&\quad\quad\quad (Predators, (2, 0), \{\!| m, m, summer_0 |\!\})|\!\}),\\
|\!\})
\end{aligned}
$$

The image shown in figure 4.5 depicts the initial state of the system. The outermost membrane, is the *Env* and contains four *Loc*. The innermost membranes are *Predator*. According to the rules, the presence of promoter $summer_{(0)}$ allows the application only of those rules with summer as promoter. In each membrane the promoter $summer_{(0)}$ is associated with $summer_{month<4} \to summer_{month++}$ In $Loc_{(2)}$ p, f are associated to $p, f \to p|_{summer}$, p, f, f are associated to $p, f, f \to p, p, p|_{summer}$ and $p, f \to p, p|_{summer}$. In $Loc_{(4)}$ p, f are associated to $p, f \to p|_{summer}$ and p, f, f are associated to $p, f, f \to p, p, p|_{summer}$. The number of m within each membrane *Predators* has not changed, then the update function leaves unaltered the values of *members*.

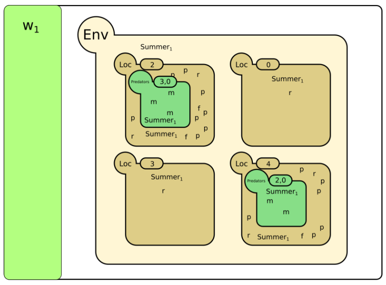


Figure 4.6: after step 1

The image shown in figure 4.6 depicts the state of the system after one step. According to the rules, the presence of promoter $summer_{(1)}$ allows the application only of those rules with summer as promoter. In each membrane the promoter $summer_{(1)}$ is associated with $summer_{month<4} \to summer_{month++}$ In $Loc_{(2)}$ p, f are associated to $p, f \to p, p|_{summer}$ and p, f, f are associated to $p, f, f \to p, p, p|_{summer}$. In $Loc_{(4)}$ p, f are associated to $p, f \to p, p|_{summer}$. The number of m within each membrane *Predators* has not changed, then the update function leaves unaltered the values of *members*.

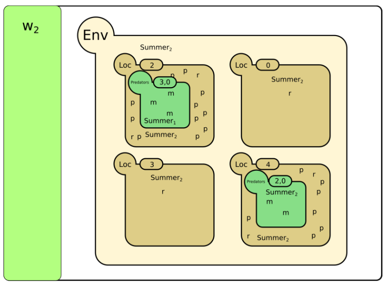


Figure 4.7: after step 2

The image shown in figure 4.6 depicts the state of the system after two steps. These few steps will only give an idea of how the system evolves, following the rules of the example. The system evolves according to the rules of evolution, where the rules that modify the promoters simulate the seasons and membranes *predators* are used as reactants while the promoters *winter* or *autumn* are present.

Chapter 5

A case of study: Serengeti lions.

Lion is the only feline that is truly social, living in prides and coalitions, the size and dynamics of which are determined by an intricate balance of evolutionary costs and benefits. Why social behaviour, lacking in other cats, become so important in this species? Is it a necessary adaptation for hunting large prey such as wildebeest? Does it facilitate the defence of young cubs? Has it arisen from the imperatives of competing for territory? As details of leonine sociality have emerged, mostly over the past 40 years, many of the key revelations have come from a continuous study of lions within a single ecosystem: the Serengeti.

As we can read in [138, 139, 140, 141], Serengeti National Park encompasses 5,700 square miles of grassy plains and woodlands near the northern border of Tanzania. The park had its origin as a smaller game reserve under the British colonial government in the 1920s and was established formally in 1951. The greater ecosystem, within which vast herds of wildebeest, zebra, and gazelle migrate seasonally, following the rains, includes several game reserves (designated for hunting) along the park's western edge, other lands under mixed management regimes (including the Ngorongoro Conservation Area) along the east, and a trans-boundary extension (the Masai Mara National Reserve) in Kenya. In addition to the migratory herds, there are populations of mice, reedbuck, waterbuck, eland, impalas, buffalo, warthogs, and other herbivores living less peripatetic lives. Nowhere else in Africa supports quite such a concentrated abundance of hoofed meat, amid such open landscape, and therefore the Serengeti is an important place for lions and an ideal site for lion researchers.

For this animals, life is hard and precarious, and casualties are numerous, for them as well as for their prey, life spans tend to be short, more often terminating abruptly than in a graceful decline. An adult male lion, if he is lucky and durable, might attain the advanced age of 12 in the wild. Adult females can live longer, even to 19. A tipical case of mortality in Serengeti is caused by Masai farmers and shepherds who identify which the lions a source of danger to their herds and their crops[133]. Life expectancy at birth is much lower, for any lion, if we consider the high mortality among cubs, half of which die before age two. But surviving to adulthood is no guarantee of a peaceful demise. Continual risk of death, even more than the ability to cause it, is what shapes the social behaviour of this ferocious but ever jeopardized animal.

Male lions, not strictly belonging to any pride, instead form coalitions with other males and exert controlling interest over a pride, fathering the cubs and becoming resident, loosely associated with the pride[129]. They also play an important role in helping to kill prey, especially with larger and more dangerous animals, such as cape buffalos or hippos, thereby contributing something besides sperm and protection to the life of the pride as we can see in [132].

Usually lion coalitions make a challenge for controlling rights to a pride. In this situation the roars play an important role, serve to indicate totheir opponents their numerical strength. In some cases, conflicts between lions finish before beginning when a group of lions realizes to be inferior to the other for size [136]. If a coalition of males took over, it would kill the young of their rivals to bring the females quickly back into aestrum. Mostly lions die because they kill each other. The number one among the causes of death for lions, in an undisturbed environment, are other lions. At least 25 percent of cub loss is due to infanticide by incoming males. Females too, given the chance, will sometimes kill cubs from neighbouring prides. They will even kill another adult female, if she unwisely wanders into their ambit. Resources are limited, prides are territorial, a lot of bite wounds visible on lions, reflecting the competitive struggle for food, territory, reproductive success, sheer survival.

It is not just the need for joint effort in making and defending kills, that drives lionesses to live in prides[137]. It's also the need to protect offspring and retain those premium territories. Although pride size varies widely, from just one adult female to as many as 18. Prides in the middle range succeed best at protecting their cubs and maintaining their territorial tenure. Prides that are too small tend to lose cubs. Periods of oestrus for the adult females often are synchronized especially if an episode of male infanticide has killed off all their young and reset their clocks. In this case that cubs of different mothers are born at about the same time. This allows the formation of créches, lion nursing groups in which females suckle and protect not just their own cubs but others too. Such cooperative mothering, efficient in itself, is further encouraged by the fact that the females of a pride are related as mothers, daughters, sisters and aunts, sharing a genetic interest in one another's reproductive success. But prides that are too large do poorly also, because of excessive within-pride competition. A pride of two to six adult females seems to be optimal on the plains[135].

Male coalition size is governed by similar logic. Coalitions are formed, typically, among young males who have outgrown the natal pride and gone off together to cope with adulthood. One pair of brothers may team with another pair, their half-siblings or cousins, or even with unrelated individuals that turn up, solitary, nomadic, and needing partnership.
Too many of such males together are roving posse, each hungry for food and for chances to mate may be the prodrome of an internal conflict. But a lone male, or a coalition that is too small just a pair, will suffer disadvantages also.

As we can see in this little preamble, life is very hard for a lion, but more interesting for a realization of a model based on MAPPS. Because of the many challenges, life for pride proves to be really complex. We tried to put the total amount of problems faced by lions during their lifetime into this model, and we formalized MAPP Systems as main tool that would help us in this purpose. Here below we present a formal definition of our model for Serengeti lions, taken as example because in Serengeti live the most studied and known pride of lions.

5.1 Serengeti lions - Informal description

1. Environment: the place of simulation, where are contained the hunting territories controlled by prides

2. Hunting ground: Locations where there is a pride and may be subjected to invasion by coalitions of stranger males

3. Pride: the largest social organization of lions consisting of several females with their cubs, young members and a small number of males

4. Court: group of females kindred with each other by matrilineality, and their young or cubs

5. Coalition: group of males kindred with each other, which are part of a pride or wander in search of a pride to take over

6. Family: a female with cubs under 18 months

7. Subcourt: a family of cubs between 18 months and 4 years, differs from the family, because, in the case of a takeover of pride, cubs are not killed by the new males

8. Subcoalition: a group of male cubs between 18 months and 4 years within the pride, differs from the family, because, in the case of a pride's takeover, cubs are not killed by new males

Objects of simulation are:

male: age, health female: age, health cube: resources: health, casualty

In addition, we provided a collection of control objects, to manage the events and seasonal cycles of lions' life.

The simulation of a year of life of the lions is as follows:

1. a finite number of hunting iterations under favourable conditions:
   In each iteration the pride seeks to acquire prey, individual survival is tested as a function of the preys captured, during the phases of hunting a shortage of prey can cause divisions

2. take over:
   At this phase a coalition of nomadic males tries to chase away the breakfast resident and replace it, if they succeed all the little lions are killed to make the females fertile again and available.

3. A mating season:
   Where the lions of the coalition within the pride mate with all adult females who haven't sons under the age of 18 months

4. a finite number of hunting iterations under adverse conditions:
   Even in this phase, the pride seeks to acquire prey, individual survival is tested as a function of the captured prey, so in this case the shortage of food is greater, and this favours internal divisions

5. Births and ageing:
   At the end of the season of adversities and with the beginning of the abundance's season, new cubs born and new families are formed within cohorts and each lion goes through the ageing rule

In our simulation we also manage events like:

1. creation of coalitions formed by peripatetic males unrelated

2. Expulsion of members from pride

3. Expulsion of members from location

4. Casualties during hunting phase

We could add complexity to the system implementing many other features, but we have tried to capture only the most important events, leaving out what would occur either on too specific conditions or on sporadic events or what would be of little interest for the simulation.

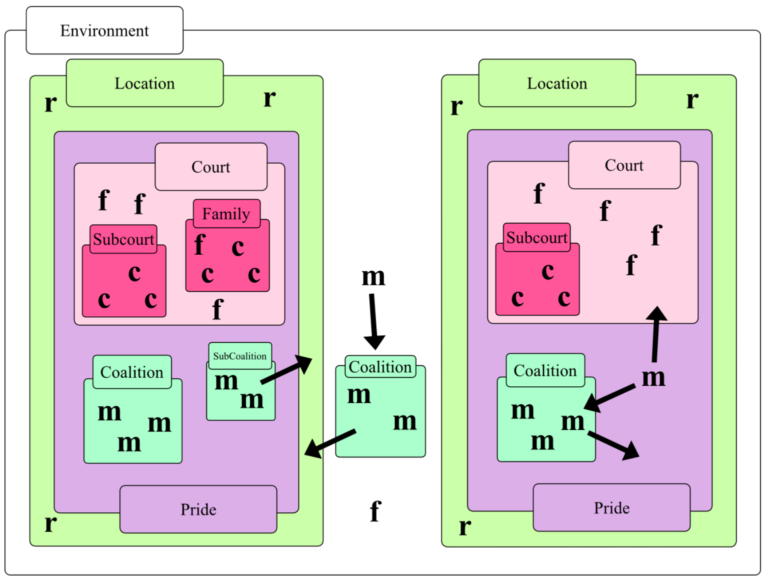


Figure 5.1: initial state ω_2

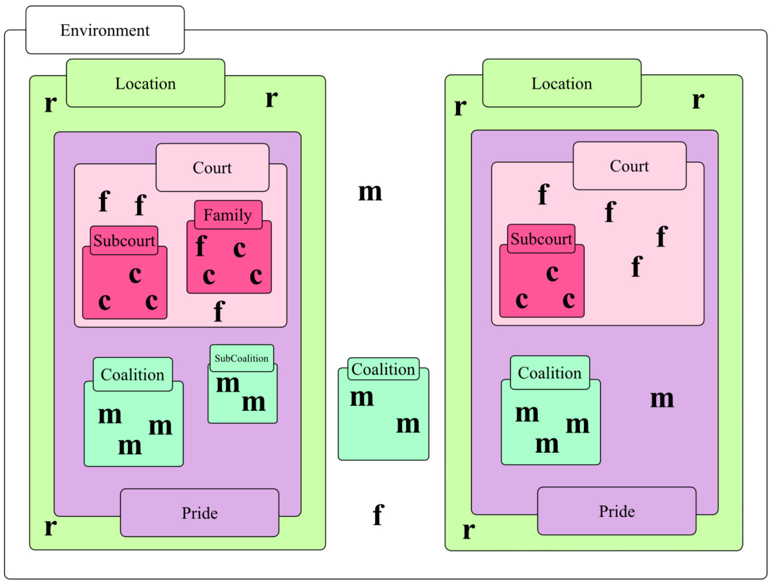
As we see in the picture above, our model is composed by eight membranes and five elements, as shown by arrows, our items can pass from a membrane to another, the male, for example, a lion can leave its coalition, goes to pride membrane, and, from there, enters the court, in order to mate with the lionesses of that court. A male in the environment can also join a coalition in order to find allies with the aim of trying a take over in the future. Also a membrane can move from a parent membrane to another, for instance, a coalition of males, can enter in a pride, fight with the resident coalition and stay there, driving out the loser coalition. A membrane could also change in another one, copying its state, a Family for example, when it grows enough, turns into court.

5.2 Serengeti lions - Formal definition

P is the tuple: $\{\ \Theta, \Gamma, \Sigma, \omega_0, R\}, where:$

1. Σ is a finite alphabet of symbols {m,f,c,r,s,winter,spring,autumn,summer} representing elements present inside the membranes. They respectively stand for male, female, cub, resources, sources and seasons, season are used as control elements

2. Γ is a finite alphabet of symbols { E,L,P,Cr,Cl,Scr,Scl,F } representing possible type of membranes:
   (a) E : Environment
   (b) L : Location
   (c) P : Pride
   (d) Cr: Court
   (e) Cl: Coalition
   (f) Scr: SubCourt
   (g) Scl: SubCoalition
   (h) F : Family
3. Θ represents membranes' order:
   (a) $L \subset E$
   (b) $P \subset L$
   (c) $Cr \subset P$
   (d) $Cl \subset Cr$
   (e) $Scr \subset Cl$
   (f) $Scl \subset Scr$
   (g) $F \subset Scl$



4. ω_0 is a tuple of values in $\Sigma, \Gamma = \{\dots\}$ which describes initial state of a system, where Σ is the set of elements that we can find in a membrane of P and Γ is the set of membrane of P.

5. $D_\Gamma = \{ \mathrm{D}_{\Gamma_E}, D_{\Gamma_L}, D_{\Gamma_P}, D_{\Gamma_{Cr}}, D_{\Gamma_{Cl}}, D_{\Gamma_{Scr}}, D_{\Gamma_{Scl}}, D_{\Gamma_F} \}$ Ordered set of domains, in a one-to-one correspondence with elements of set Γ.

   (a) $D_{\Gamma_E} = \{geneticcode, privation\}$
   Genetic code indicates the most suitable lion which can survive as a peripatetic, Privation indicates a percentage of weakness that can affect lions health

   (b) $D_{\Gamma_L} = \{geneticcode\}$
   Genetic code indicates the most suitable lion which can survive in a specific location.

   (c) $D_{\Gamma_P} = \{Health, Strength, Requirements, Coalitions\}$
   Health indicates the total amount of *health* property taken from every member of a Pride, *Strength* indicates the total amount of *Strength* property taken from every members of a Pride, *Requirements* indicates total food requirements for a pride, *Coalitions* indicates the number of coalitions present in a Pride.

   (d) $D_{\Gamma_{Cr}} = \{Health, Strength, Requirements\}$
   Health indicates the total amount of *Health* property taken from every members of a Court, Strength indicates the total amount of *Strength* property taken from every members of a Court, Requirements indicates total food requirements for a pride.

   (e) $D_{\Gamma_{Cl}} = \{Health, Strength, Requirements, Resident\}$
   Health indicates the total amount of *Health* property taken from every members of a Coalition, Strength indicates the total amount of Strength property taken from every members of a Coalition, Requirements indicates total food requirements for a pride. Resident indicates if a Coalition is resident or not.

   (f) $D_{\Gamma_{Scr}} = \{Health, Strength, Requirements\}$
   Health indicates the total amount of Health property taken from every members of a SubCourt, Strength indicates the total amount of Strength property taken from every members of a SubCourt, Requirements indicates total food requirements for a pride.

   (g) $D_{\Gamma_{Scl}} = \{Health, Strength, Requirements\}$
   Health indicates the total amount of Health property taken from every members of a SubCoalition, Strength indicates the total amount of Strength property taken from every members of a SubCoalition, Requirements indicates total food requirements for a pride.

   (h) $D_{\Gamma_F} = \{Health, Cubs, Age\}$
   Health indicates the total amount of Health property taken from every members of a Family, Cubs indicates the total number of Cubs within the Family, Age indicates the age of the cubs, all cubs have the same age.

6. $D_\Sigma = \{ \mathrm{D}_{\Sigma_m}, D_{\Sigma_f}, D_{\Sigma_c}, D_{\Sigma_r} \}$ Ordered set of domains, in a one-to-one correspondence with elements of set Σ.

   (a) $D_{\Sigma_m} = \{Age, Strength, Geneticcode, resident\}$
   Age indicates how a Lion is old, Strength indicates the percentage of success for a lion to defeat other lions, according to strength of opponents, Genetic code indicates if a lion is suitable for the location in which on lives, resident is a flag, we need it to know if a male is resident into a pride or if it is just arrived after a take over, in the second case he will kill the cubs of a female.

   (b) $D_{\Sigma_f} = \{Age, Strength, Geneticcode, Secondarygeneticcode, \}$
   Age indicates how a lioness is old, Strength indicates the percentage of success for a lioness to defeat other lions, according to strength of opponents, Genetic code indicates if a lioness is suitable for the location in where she lives while secondary genetic code indicates the genetic code of the male, if she is pregnant, it is empty otherwise.

   (c) $D_{\Sigma_c} = \{Geneticcode, Sex\}$
   Cubs don't need an age, it is recorded in the attributes of $Family$, Genetic Code will be useful once they will became adult, sex is necessary to produce a male or a female from a cub.

   (d) $D_{\Sigma_r} = \{Rating, Amount\}$
   Rating indicates how fast a resource grow up, Amount indicates how much food requirements can satisfy.

   (e) $D_{\Sigma_s} = \{\epsilon\}$
   this item has no attributes it is necessary only to produce new resources.

   (f) $D_{\Sigma_{spring}} = \{duration\}$
   duration indicates how many iterations this control item waits before than expire.

   (g) $D_{\Sigma_{summer}} = \{duration\}$
   duration indicates how many iterations this control item waits before than expire.

   (h) $D_{\Sigma_{autumn}} = \{duration\}$
   duration indicates how many iterations this control item waits before than expire.

   (i) $D_{\Sigma_{winter}} = \{duration\}$
   duration indicates how many iterations this control item waits before than expire.

7. The system evolves within a cycle of four seasons:

   In each season we can see a different set of rule for each membrane:

	F	Scl	Scr	Cl	Cr	P	L	E
Season 1					Birth		Coalition enter in pride Food collecting	
Season 2					Family became Subcourt	Internal fight Expulsion	Expulsion	Coalition enter in location
Season 3				Expulsion	mating Expulsion	Male enter in Court		
Season 4	Selection Aging	Selection Aging	Selection Aging	Family become Court Selection Aging	Selection Aging	Subcoalition become Coalition Selection Aging	Selection Aging	Selection Aging

(a) Season 1:
During season 1, new lions born; families are created; a coalition, stationing in the same location of a Pride, where there is only a coalition, can enter inside; in the same season lions provide to collect food, during the hunt some lion could die.

(b) Season 2
During season 2, there could be a take over; a Subcolation too old to be considered a cub could be expelled from pride, Coalition or Subcoalition present in a location could be expelled.

(c) Season 3
This is the mating season, males leave their membranes and, from Pride membrane enter in court membrane, where mate with females; this season takes eight iterations, to allow this shifting of lions and their return to their pride

(d) Season 4
Winter has come, this is the harder season for lions, in this season, who survive get older otherwise dies, we use this season to simulate ageing, families old enough become subcourts, other families become subcoalitions, subcoalition get stronger and become coalition, ready to try an internal take over in the next season;.

8. The set of rules R is divided in subsets by membranes: *(E, L, P, Cr, Cl, Scr, Scl, F)* to make reading easier we decided to divide them even further according to the seasons.

   (a) Season 1

   i. $R_E = \{$
   $spring \rightarrow summer$
   $Cl, L_{coalitions<2} \rightarrow Cl_{in(L)}, L$

}
During Season1, coalitions move from environment to locations where number of coalitions is less than 2 to. In each membrane we have an element usually used as promoter, that changes from spring to summer.

ii. $R_L = \{$
$spring \rightarrow summer$
$P, r \rightarrow P, r$
$P_{health}, r_{amount} \rightarrow P_{health+=amount}$
$P_{health}, r_{amount} \rightarrow P_{health+=amount}, w_{in(P)}$
$P, r \rightarrow P, r, w_{in(P)}$
$Cl, P_{coalitions<2} \rightarrow Cl_{in(P)}$
}
During Season1, a Pride hunts a resource, there are four rules that managed this event and which represent the possibility of capturing the food, or not, and the possibility that during the hunt there would be casualties, or injuries. The object w represent a possible casualty into a pride. The element w will enter into an internal membrane that could be a coalition or a court or a subcourt and there consumed by rules. In the same season a Coalition move into a pride where number of coalitions is less than 2.

iii. $R_P = \{$
$spring \rightarrow summer$
$w, Cl \rightarrow w_{in(Cl)}, cl$
$w, Cr \rightarrow w_{in(Cr)}, cl$
$w, Scl \rightarrow w_{in(Scl)}, cl$
$w, Scr \rightarrow w_{in(Scr)}, cl$
}
During Season1, a wound received by Pride goes into a coalition, a subcoalition, a subcourt or into a court.

iv. $R_{Cr} = \{$
$spring \rightarrow summer$
$f_{code,code^2,strength,age} \rightarrow F_{health,age=0,litter=2}(\{f, c_{code=mix(code_f,\, code^2_f)}, c_{code=mix(code}$
$f_{code,code^2,strength,age} \rightarrow F_{health,age=0,litter=3}(\{f, c_{code=mix(code_f,\, code^2_f)}, c_{code=mix(code}$
$c_{code=mix(code_f,\, code^2_f)}\})$
$f_{code,code^2,strength,age} \rightarrow F_{health,age=0,litter=4}(\{f, c_{code=mix(code_f,\, code^2_f)}, c_{code=mix(code}$
$c_{code=mix(code_f,\, code^2_f)}, c_{code=mix(code_f,\, code^2_f)}\})$
$f_{code,code^2,strength,age} \rightarrow F_{health,age=0,litter=6}(\{f, c_{code=mix(code_f,\, code^2_f)}, c_{code=mix(code}$
$c_{code=mix(code_f,\, code^2_f)}, c_{code=mix(code_f,\, code^2_f)}, c_{code=mix(code_f,\, code^2_f)} c_{code=mix(code_f,\, code}$
$Scl \rightarrow Scl_{out};$
}

During Season1, a pregnant female, who copied in *code*[2] the genetic code of a male, generates a Family, four different rules concur to decide whether the number of cubs in the family will be 2, 3, 4 or 6, in this season a subcoalition, generated from a Family, goes out from the court to the Pride.

v. $R_{Cl} = \{$
$spring \rightarrow summer$
$w, m \rightarrow m$
$w, m \rightarrow \_$
$\}$
During Season1, in each membrane we have an element usually used as promoter, that changes from spring to summer. An element w, produced during the hunt, can be assigned to $w, m \rightarrow \_$ producing the death of a male.

vi. $R_{Scr} = \{$
$spring \rightarrow summer$
$w, f \rightarrow f$
$w, f \rightarrow \_$
$c \rightarrow f$
$\}$
During Season1, in each membrane we have an element usually used as promoter, that changes from spring to summer. An element w, produced during the hunt, can be assigned to $w, m \rightarrow \_$ producing the death of a female.

vii. $R_{Scl} = \{$
$spring \rightarrow summer$
$w, m \rightarrow m$
$w, m \rightarrow \_$
$c \rightarrow m$
$\}$
During Season1, in each membrane we have an element usually used as promoter, that changes from spring to summer. An element w, produced during the hunt, can be assigned to $w, m \rightarrow \_$ producing the death of a male.

viii. $R_F = \{$
$spring \rightarrow summer$
During Season1, in each membrane we have an element usually used as promoter, that changes from spring to summer.

(b) Season 2

i. $R_E = \{$
$summer \rightarrow autumn$
$m \rightarrow m_{in(Cl)} | Cl$
$\}$

During Season 2, in each membrane we have an element usually used as promoter, that changes from summer to autumn.

ii. $R_L = \{$
$summer \rightarrow autumn$
$Cl \rightarrow Cl_{out}$
$\}$
During Season 2, a Coalition exits from location to environment.

iii. $R_P = \{$
$summer \rightarrow autumn$
$Cl_{strenght}, Cl_{strenght'} \overset{strenght/(strenght+strenght')}{\rightarrow} Cl_{resident=true}, Cl_{out}$
$Cl_{strenght}, Cl_{strenght'} \overset{strenght'/(strenght+strenght')}{\rightarrow} Cl_{resident=true}, Cl_{out}$
$\}$
During Season 2, two coalition fight and one is expelled from pride who won becomes resident.

iv. $R_{Cr} = \{$
$summer \rightarrow autumn$
$w, f \rightarrow f$
$w, f \rightarrow$
$\}$
During Season 2, a wound kill a female or is consumed.

v. $R_{Cl} = \{$
$w, m \rightarrow m$
$w, m \rightarrow$
$summer \rightarrow autumn$
$\}$
During Season 2, a wound kill a male or is consumed.

vi. $R_{Scr} = \{$
$summer \rightarrow autumn$
$w, f \rightarrow$
$\}$
During Season 2, a wound kill a female

vii. $R_{Scl} = \{$
$summer \rightarrow autumn$
$w, c \rightarrow$
$\}$
During Season 2, a wound kill a male

viii. $R_F = \{$
$summer \rightarrow autumn$

(c) Season 3

i. $R_E = \{$

$autumn_{duration>8} \rightarrow winter$
$autumn_{duration\leq 8} \rightarrow autumn_{duration+1}$
$L, Cl \rightarrow Cl_L, L$
$\}$
During Season 3, in each membrane we have an element usually used as promoter, that changes from autumn to winter, if duration $>$ 8 else duration is increased by one.

ii. $R_L = \{$
$autumn_{duration>8} \rightarrow winter$
$autumn_{duration\leq 8} \rightarrow autumn_{duration+1}$
$\}$
During Season 3, in each membrane we have an element usually used as promoter, that changes from autumn to winter, if duration $>$ 8 else duration is increased by one.

iii. $R_P = \{$
$autumn_{duration>8} \rightarrow winter$
$autumn_{duration\leq 8} \rightarrow autumn_{duration+1}$
$m, Cr \overset{duration \pmod 2=0}{\rightarrow} m_{in(Cr),Cr}$
$m, Cl \overset{duration \pmod 2=1}{\rightarrow} m_{in(Cl),Cl}$
$\}$
During Season 3, males exit from coalition and enter into court to mate with females.

iv. $R_{Cr} = \{$
$autumn_{duration>8} \rightarrow winter$
$autumn_{duration\leq 8} \rightarrow autumn_{duration+1}$
$m \overset{duration \pmod 2=1}{\rightarrow} m_{out}$
$f \overset{strength_m}{\rightarrow} f_{code^2=code_m}|m$
$f \rightarrow f|m$
$F \rightarrow f|m_{resident=false}$
$\}$
During Season 3, males exit from court to pride, after that they mated with females, if males are not resident (the coalition was set as resident but not males are inside) a Family produces a female.

v. $R_{Cl} = \{$
$autumn_{duration>8} \rightarrow winter$
$autumn_{duration\leq 8} \rightarrow autumn_{duration+1}$
$m \overset{duration \pmod 2=0}{\rightarrow} m_{out}$
$\}$

vi. $R_{Scr} = \{$

$autumn_{duration>8} \to winter$
$autumn_{duration\leq 8} \to autumn_{duration+1}$
$\}$
During Season 3, in each membrane we have an element usually used as promoter, that changes from autumn to winter, if duration > 8 else duration is increased by one.

vii. $R_{Scl} = \{$
$autumn_{duration>8} \to winter$
$autumn_{duration\leq 8} \to autumn_{duration+1}$
$c \to m;$
$\}$
During Season 3, a cube become a male.

viii. $R_F = \{$
$autumn_{duration>8} \to winter$
$autumn_{duration\leq 8} \to autumn_{duration+1}$

(d) Season 4

i. $R_E = \{$
$winter \to spring$
$\}$
During Season 4, in each membrane we have an element usually used as promoter, that changes from winter to spring.

ii. $R_L = \{$
$winter \to spring$
$\}$
During Season 4, in each membrane we have an element usually used as promoter, that changes from winter to spring.

iii. $R_P = \{$
$winter \to spring$
$Scl \to Scl_{age+1}$
$m_{resident} \overset{\phi(age)}{\to} m_{resident=resident'}, age+1|Cl_{resident'}$
$m \overset{\phi(age)}{\to} \_$
$\}$
During Season 4, a male ages or dies according to the rules and a subcoalition ages.Value of resident attribute of males is set with the value of resident of Coalition, so males that was not resident in Season 3 become resident.

iv. $R_{Cr} = \{$
$winter \to spring$
$Scr \to Scr_{age+1}$
$F \to F_{age+1}$

$F_{age>3} \rightarrow Cl$
$F_{age>3} \rightarrow Cr$
$f \overset{\phi(age)}{\rightarrow} f_{age+1}$
$f \overset{\phi(age)}{\rightarrow} \_$
$\}$
During Season 4, a female ages or dies according to the rules and a subcourt ages.

v. $R_{Cl} = \{$
$winter \rightarrow spring$
$m \overset{\phi(age)}{\rightarrow} m_{age+1}$
$m \overset{\phi(age)}{\rightarrow} \_$
$\}$
During Season 4, a male ages or dies according to the rules.

vi. $R_{Scr} = \{$
$winter \rightarrow spring$
$f \overset{\phi(age)}{\rightarrow} f_{age+1}$
$f \overset{\phi(age)}{\rightarrow} \_$
$\}$
During Season 4, a female ages or dies according to the rules.

vii. $R_{Scl} = \{$
$winter \rightarrow spring$
$m_{age} \overset{\phi(age)}{\rightarrow} m_{age+1}$
$m_{age} \overset{\phi(age)}{\rightarrow} \_$
$\}$
During Season 4, a male ages or dies according to the rules.

viii. $R_F = \{$
$winter \rightarrow spring$
$f_{age} \overset{\phi(age)}{\rightarrow} f_{age+1}$
$f_{age} \overset{\phi(age)}{\rightarrow} \_$
$c \overset{\phi(age)}{\rightarrow} \_|F_{age}$
$c \overset{\phi(age)}{\rightarrow} c|F_{age}$
$\}$
During Season 4, a female ages or dies according to the rules, a cubs dies or survives according to the rules.

where $\phi(age)$: is a function that return 1 if $age < 10$, 0 if $age = 16$, $0,5$ if $age \geq 10$, $0,5$ if $age \leq 3$

9. The set of update functions U is divided in subsets by membranes: (E,L,P,Cr,Cl,Scr,Scl,F)

(a) $U_E = \{$
Empty
$\}$

(b) $U_L = \{$
$\phi_{Coalitions} = arity(Cl) \in L$
$\}$
this function calculates the number of Coalitions into a location.

(c) $U_P = \{$
$\phi_{Coalitions} = arity(Cl) \in P$
$\phi_{Strength} = Strength_{Cr} + Strength_{Cl} + \Sigma_{\forall Scl \in w_P} Strength_{scl}$
$\}$
this functions calculate the number of Coalitions into a Pride and the strength of a Pride.

(d) $U_{Cr} = \{$
$\phi_{Strength} : \Sigma_{\forall f \in w_{Crl}} Strength_f + \Sigma_{\forall Scr \in w_{Cr}} Strength_{Scr}$
$\}$
this function calculates the strength of a Court.

(e) $U_{Cl} = \{$
$\phi_{Strength} : \Sigma_{\forall m \in w_{Cl}} Strength_m$
$\}$
this function calculates the strength of a Coalition.

(f) $U_{Scr} = \{$
$\phi_{Strength} : \Sigma_{\forall f \in w_{Scr}} Strength_f$
$\}$
this function calculates the strength of a subCourt.

(g) $U_{Scl} = \{$
$\phi_{Strength} : \Sigma_{\forall m \in w_{Scl}} Strength_m$
$\}$
this function calculates the strength of a subCoalition.

(h) $U_F = \{$
$\phi_{Cubs} : arity(Cubs) \in w_F$
$\}$ this function calculates the number of cubs into a Family.

Chapter 6

Experimental results

We studied the dynamics of the MAPP model described before by running simulations. In particular, we implemented a MAPP systems interpreter in *C#* that allows attributed objects and evolution rules to be represented as instantiations of specific *C#* classes. Once a MAPP systems model is specified, the interpreter simulates it by performing a number of iterations to be given as a parameter. At each iteration, a maximally parallel step is performed according to the MAPP systems semantics. The result of a simulation is the sequence of configurations reached by the interpreter at each iteration. In order to show easy readable measurements, we processed the simulation results and produced graphical representations by using the statistical framework R.

6.1 Data and results

From our model we can observe that, once we reach a certain level of complexity, which includes a great number of elements and iterations, the system reaches a certain stability. Where the number of elements of pride in one location does not grow and the coalition within is strong enough to no suffer take over from foreign coalitions.

In figure 6.1 we can see a Pride that starts with only one male and one female and a subcourt of three young females. At time t_{12} young females grow and become adult, so the number of young females decreases up to 0 and the number of adult females increases to 3, while an adult female die. The pride grows as females reproduce and, between time t_{25} and time t_{58}, the number of cubs increases. At time t_{83} we can see 5 females, 2 young females 1 male and 4 cubs. At time t_{84} three old females die because of their old age, three young females become adult, and a family changes in one subCoalition. At time t_{85} 10 new cubs born and at time t_{96} a subCoalition evolves in coalition, ready for an inner take over. Graph shows the evolution of pride within a number of 120 iterations, that signifies 10 years because every 12 iterations our system pass through all seasons. It is important to note that most of the significant changes taking place in seven years. During this time, females become pregnant, families grow and mutate into cohorts or coalitions, lions begin to die. This example shows a period of only ten years to describe what happens from the beginning of the birth of a pride, and it gives a comprehensive idea of the dynamics of a pride. Longer simulations show that it reaches a situation where the deaths reduce the population growth and leads to a substantial stability of the system.

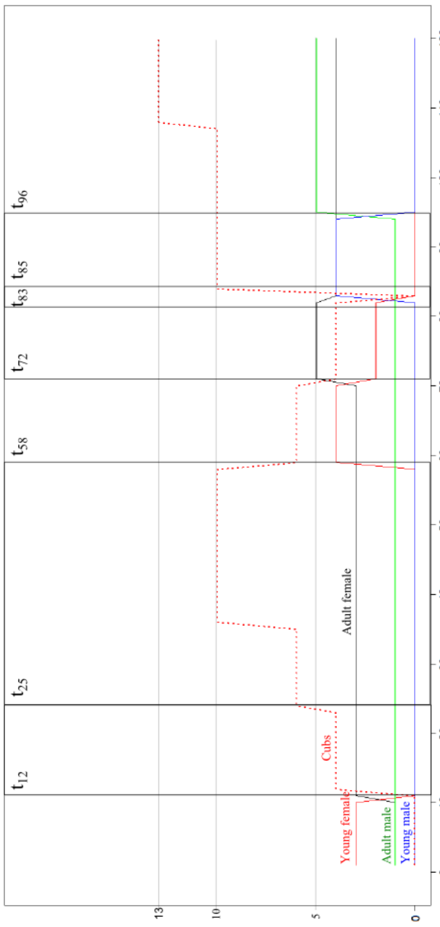


Figure 6.1

In figure 6.2 we can see a Pride starting with only one family composed by an adult female ad three cubs, six young females and two coalitions each one consisting of one male. At time t_{12} we see the effect of a take over, a family dies with its cubs. The young females become adult and one new family with four cubs born. At time t_{25} we see another two families born and we arrive to have twelve cubs. The older ones four grew up and become young female, forming a subcourt at time t_{59} while at time t_{72} four young female become adults, four cubs become young males, forming a subpride and four cubs become young females. At t_{85} the lone adult male dies, young females become adult, and young males become adult. At time t_{88} four females die. At time t_{95} six new cubs born.

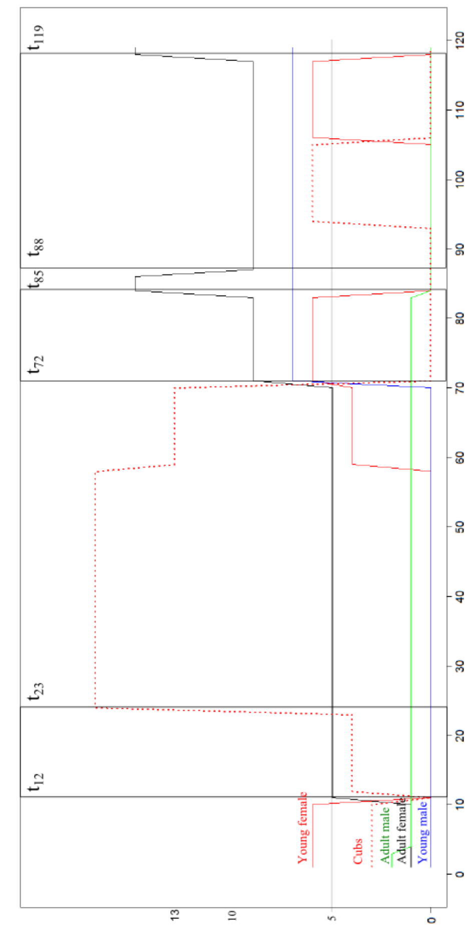


Figure 6.2

These are just two examples of what we can produce as output, using our model and our software, we wanted to propose a comprehensive and at the same time easy to read overview of the social dynamics of Serengeti lions. Changing certain settings may offer different solutions, such as the difficulty in finding food, or the rate of death or birth of lions, the different life expectancy between males and females, all of these are parameters that can affect results.

This model can be used to predict what happens in a pride when a coalition outside is introduced in the same environment. Lions adult residents sufficiently strong can stand up to any foreign coalitions and bring their own cubs to reach the age of maturity, unlike a weak coalition is unable to maintain control of the pride and makes it liable to take over.

Chapter 7

Final Conclusions

We proposed an extension of Attributed Probabilistic P systems (APP systems), called Multilevel Attributed Probabilistic P systems (MAPP systems), in which membranes are annotated with attributes and are consumed by the rules as elements. MAPP systems are intended to be used to model the dynamics of complex social behaviours in groups of animals. In this context, attributes can be used to represent characteristics of the population, such as the number of member, position, and so on. Apart from attributing membrane, the feature that mainly makes a difference between MAPP systems and other proposals is the use of membrane as items consumed by the application of rules. This feature is particularly suitable for the modelling of populations where groups are, and sometimes merge with others or even change in different group. We used MAPP systems for modelling the social behaviours of some species of Lions. In particular, as an application we developed a model to compare behaviours in different group of Lions. Such a kind of social systems is usually approached by means of agent-based models that are often poorly documented and ambiguous. On the contrary, since both the syntax and the semantics of MAPP systems are formally defined, the model based on MAPP systems is unambiguous. The model has been inspired by the behaviours of lions of Serengeti. We plan to adapt our general model to the modelling of the behaviour of particular groups of lions by changing the values of the parameters.

Appendix A

General purpose implementation

Our work includes an implementation part, it is a a software engine that takes as input an encoded formal definition of our models including an instance of the initial state of the system and gives as output a log file where are stored information of what happens during each iteration. Firstly this code was an ad-hoc code, written to produce statistical data for a model based on APP systems describing social Interactions in primates[11]. Later, the code has become a general purpose engine with the aim to receive, roughly, every kind of possible model based on APP systems.Finally, by adding some extensions, the code has been expanded in order to accept model based on MAPP systems.

A.1 Overview

A.1.1 Programming language used in the project

At the design stage, we choose to use C# as programming language. C# is a type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework. The code editor used is Microsoft Visual Studio 2010 professional. The main aspect of C# considered useful for our purposes was the *reflection* functionality.

The Reflection is one of the features of .NET Framework that has a significant importance in the development of our application. It is basically a way to extract and manipulate the information of an object at runtime. from the metadata of an application, These metadata contain all the information inherent the types of objects used by an application.

Basically we produce a parser that takes as input a little number of classes representing rules and elements of our model. These classes are written in a pseudocode based on *C#* by users and enriched by software with some simple and automatically reproducible controls and additions. The need to read the code, written into a file by users, to produce objects at runtime, fits perfectly with the reflection feature and the strong type checking of *C#*.

A.2 Commentary to code

A.2.1 software engine

From the beginning we divided the code in order to keep well separated the part relating to APP Engine from those which were specific classes related a particular model. For this reason the main part of the code is composed by few files briefly explained below, while the classes related a particular model are stored in a few number of input files:

Rule.cs
This is the class for a generic rule. Each rule written by users inherits from this class.

Item.cs
This is the generic class of an element, Each element written by users inherits from this class.

List.cs
This is a class created to manage a list of objects (reactants or promoters) is primarily used to save and change the status of the system. We use three lists: one called *currenState* that represent the current state of system, one called *dynamicState*, that is initially a copy of *currentState* from where we delete consumed objects and one called *producedItems* in which we add new objects produced by rules. At the end of a computational step, we append the list called *producedItems* to that one called *dynamicState*, then we copy the result to *currentState*, so at the beginning of every step *dynamicState* is a perfect copy of *currentState*.

clsGlobalDefs.cs
This class contains all global variables, they represent system parameters used in the simulation, such as the number of individuals present in the initial state, the maximum distance into which two individuals fight each other, and so on. Parameters are global values used by the rules and are what we usually change to characterize our simulations. Changing parameter we produce different models where, for instance, males should be more aggressive, the hierarchy should stable or not, females go into heat simulta-

neously or alternately and so on. Parameters are the way in which we tune the rules for different versions of our models. With different parameters, we can simulate different type of population with different behaviours. In this class there are also some global variables used by the engine, like input and the output file path, number of iterations, output string etc.

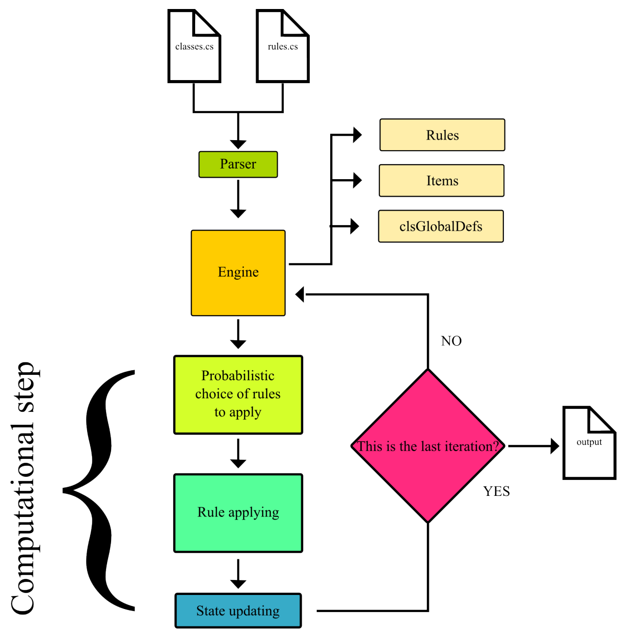


Figure A.1: Flow chart of our engine

Program.cs

This is the core of this program, the engine itself, is responsible for initializing the values in *clsGlobalDefs*, initialize elements and rules initialize lists, to launch the various computational steps and generate output. As we can see in the flow chart in Figure A.1, our engine, receives from parser a code, written in pseudocode and ready to be used, dynamically. Thanks to *reflection* feature, the engine updates the list of rules, the list of objects and global values, including *currentState*, *dynamicState* and *producedItems*, then it starts computational step. During computational step, it choices in a probabilistic manner the multiset of rules to be applied. The engine applies sequentially the multiset of chosen rules, emulating the parallelism of the APP Models, it updates the state and finally it runs the new step. At the end it produce log files and outputs. Eventually the user can define functions for particular kind of calculation.

A.2.2 input files

Input files are written in a pseudocode based on *C#* the users don't need a deep knowledge of this language because it is very similar to a great number of common imperative languages. The number of input files is four: *rules.cs*, *classes.cs*, *environment.cs* and *parameters.cs*. They compose that part of code written by user to implement a particular model.

- The file *classes.cs* contains classes defining promoters and reactant similar of our rules, below is shown an example of a simple promoter.

```
public class Season : item
{
public int duration;

public override void print()
{
Console.Write("Duration:"+this.duration+" ");
}

public string tostring()
{
return "Duration:"+this.duration+" ";
}
public override void printf()
{
GlobalDefs.Text += "Duration:"+this.duration+" ";
}
}
```

  Where the variable duration of type int is an attribute of object Season. We define the class *item* including three virtual method: *print()*, *printf()*, and *tostring()*, to constrain users to write three output methods used to obtain outputs from their model.

- In the file *environment.cs* we store the initial state of environment. For instance, we can see below a code example.

```
public class environment : item
{
private static int _Summer = 1;
private static int _Autumn = 0;
private static int _Winter = 0;
private static int _Spring = 0;
}
```

In the example above is shown a simple initial state representing the following $w_0 = \{Summer\}$ where *Autumn*, *Summer*, *Winter* and *Spring* are control items, usually used like promoter that are used to simulate the changing of seasons.

- The file *rules.cs* include all rule classes written by users. Essentially, what a user need to write into the file *rules.cs* is just a little class for each rule, with very few information like in the example below.

```
 public class [ruleName] : Rule
{
 public [ruleName]()
 {
  Type1 = Winter;
  Type2 = Male;
 }
 public override int rating(int i, int j)
 {
  int month = a.month + 1;
  if (month < 4 )
   return 1;
  else
   return 0;
 }
 public override void apply(int i, int j)
 {
  int month = a.month + 1;
  Winter o = new Winter(month);
  Console.WriteLine("userMessage"));
  GlobalDefs.producedItems.add(o);
 }
}
```

Where, [ruleName] stands for a simple name, chosen by the user as an identifier for rule class. As we can see, the commands Type1 = Winter; and Type2 = Male; are not *C#* commands, This syntax has been simplified to make it easier to write rules for users. Actually the command "Type1 = Winter;" corresponds to the command $this.Type1 = typeof(Winter)$; $Type1$ and $Type2$ are two variable that we use to set the types (in the formalism the symbols of the alphabet) of the reactants of our rules. Once these variables are set, our parser provides to make some little control like to control if at least one object for each type is present in our current state. The i-th occurrence of an object of type $Type1$ is always referred by variable a while the first occurrence of an object of type *Type*2 is always referred by variable *b*. The word *userMessage* into the method *WriteLine* is just an example of what a user can write as output. *GlobalDefs* is the static class that contains the global variables. In *GlobalDefs* are present three lists, *currentState*, *dynamicState* and *producedItems*. The list *producedItems* is a list of reactants and promoters representing the products of our rule.

- The input file *parameters.cs* is another important input file, here a user can write all the usable parameters by our system. for instance, here below we can see a simple example.

```
public class parameters : item
{
private static int _iterations = 48;
}
```

The variable *_iterations* stands for the number of computational steps performed by our application, this is a crucial parameter that we have to set, by default the value of this parameter is 10; the character _ before the word iterations, is used by our parser to know when the name of a variable starts, in this way it can recognize the string placed between *static* and _ as the name of the type, the string placed between _ and = as the name of a variable, and the string between = and ; as the vale of a variable.

A.2.3 apply method

Rule classes was implemented respecting the formal definition. The apply method definition is used by the user to define which new objects are created and set the values of the attributes of each object. If a rule does not produce any products, the apply method does not create any new objects . This method provides the rule application. For example, if we have a rule like this below:

$a_x, b_y \rightarrow k_{e=f(x,y,z)}|_{cz}$ where $f(x, y, z) = x + y + z$

the code for the corresponding apply method will be as follow

```
public override void apply(int i, int j)
{
k o = new k();
o.e = a.x + b.y + c.z;
GlobalDefs.producedItems.add(k);
}
```

As said above, *GlobalDefs* is the static class that contains the global variables. While *producedItems* is the list where we place new items produced by our rules. We remark that a and b are the first and the second reactants used by our rules, while c is always the first promoter of a rule,k is a type of a class written by the users.

A.2.4 Rating method

The rating method is crucial for the implementation of probabilistic choice. As the name suggests, it provides the rating function implementation, and its application return a weight $\in \mathbb{R}^+$ this weight will be normalized to obtain the probability of any rule to be picked up and placed in multiset of chosen rules. Taking as an example an update function like that.

$a_x, b_y \overset{f(x,y,z)}{\rightarrow} k|c_z$ where $f(x, y, z) = x + y + z$

our code will be:

```
public override int rating(int i, int j)
{
return a.x + b.y + c.z
}
```

eventually a rating could return a constant like here below:

```
public override int rating(int i, int j)
{
return 1
}
```

or be more complex and offer different values as for instance:

```
public override int rating(int i, int j)
{
switch (a.x + b.y + c.z)
{
case 10:
return 1;
break;
case 5:
return 2;
break;
default:
return 0;
break;
}
```

A.2.5 Matrices of choice

The formalism provides, a mechanism for the probabilistic choice of rules. We implemented this mechanism into our code. A probability value is obtained by a weight received from the rating method, this weight is used by our system to produce, at the beginning of each computational step, a set of matrices as we can see in the graph in figure A.2.

$w_0 = \{a, a, b, b, c, d, e, e, e, \}$

$rule_0 : a, b, \xrightarrow{2} c$
$rule_1 : d, e, \xrightarrow{1} d$
$rule_2 : b, c, \xrightarrow{3} a$

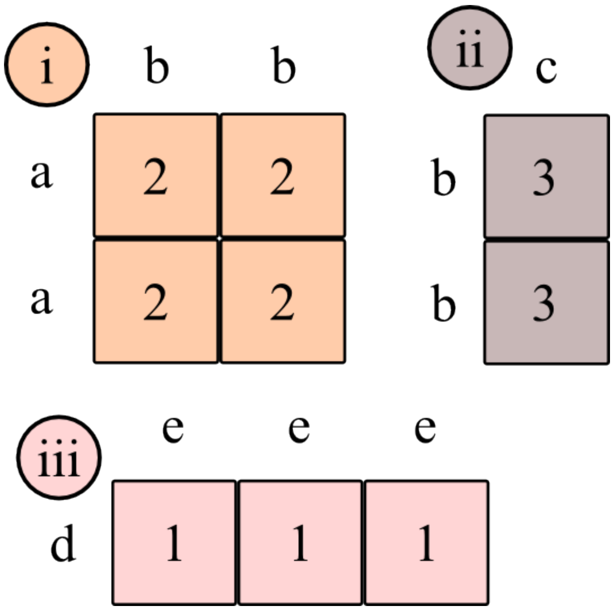


Figure A.2: matrices of choice

To each matrix we assign a score given by the sum of each value in their elements. So for instance, from graph above we have: for (i) $score = 8$, for (ii) $score = 6$, for (iii) $score = 3$ the total score is 17.

The engine receives from a random function a number between 0 to 17 (the total score), reads the score of each matrix to identify in which matrix corresponds to the score then finds the corresponding matrix row and column. For example if the number obtained from random function is 2 the corresponding matrix matrix is the first matrix, the corresponding row is 0 and the corresponding column is 0. It means that we will take from *dynamicState* the first occurrence of b and the first occurrence of a. Figure A.3 shows a flow chart that summarizes what has just been said.

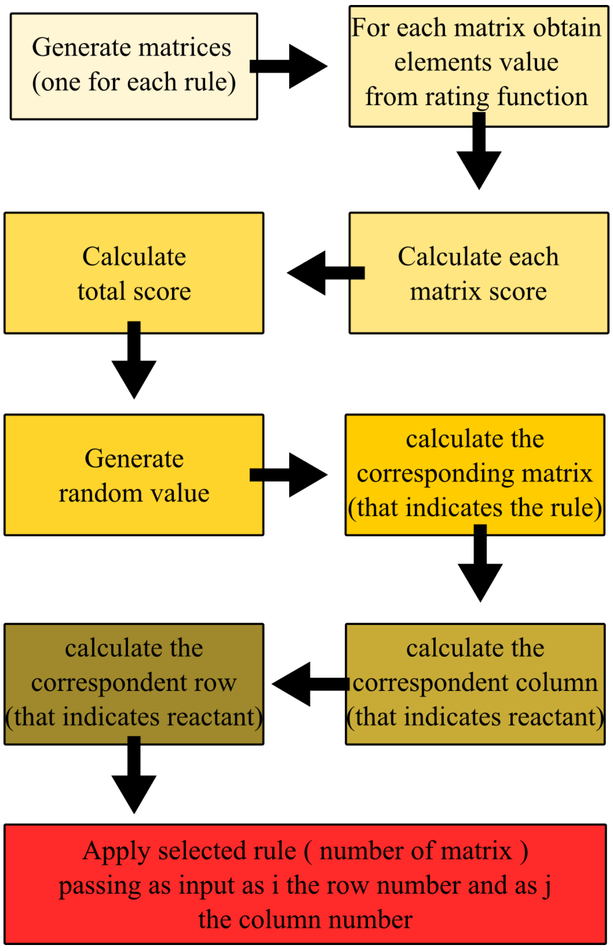
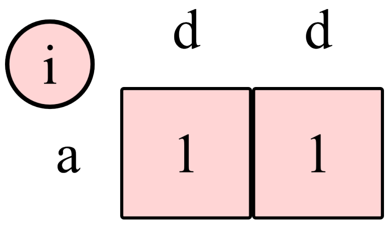

Generate matrices (one for each rule)
For each matrix obtain elements value from rating function
Calculate each matrix score
Calculate total score
Generate random value
calculate the corresponding matrix (that indicates the rule)
calculate the correspondent column (that indicates reactant)
calculate the correspondent row (that indicates reactant)
Apply selected rule (number of matrix) passing as input as i the row number and as j the column number

Figure A.3: Flow chart of matrices of choice

The second reactant should be replaced by a promoter, it works in the same way but, obviously, promoter cannot be consumed by the method apply. For instance, if we have a rule like this: $a \to d|_c$ and state would be like this $w_0 = \{a, b, b, c, d, e, e, e, \}$, then we can see the following matrix generated



Matrix score will be 2 and if we obtain, as random value, 1 then we will obtain 0 as number of matrix, 0 as number of column and 0 as number of row.

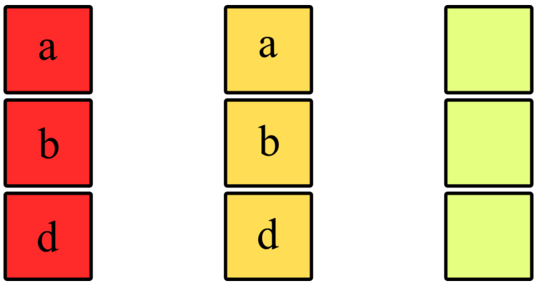
A.2.6 State updating

According to the requirements of formalism, we have three different lists, one representing current state, one representing the state that we change little by little as we apply each rules, that we call *dynamicState*, and one that stores every produced items. The way in which we manage and modify these lists represents the way in which we emulate a sort of parallelism of our rules application. As we are not able to apply all the rules at the same time, as the formal model would like, we have to apply them sequentially, but we want to avoid losing the parallelism. To give a concrete example, if our state is:
$w_0 = \{a, b, d\}$ and we have two rules as the following:

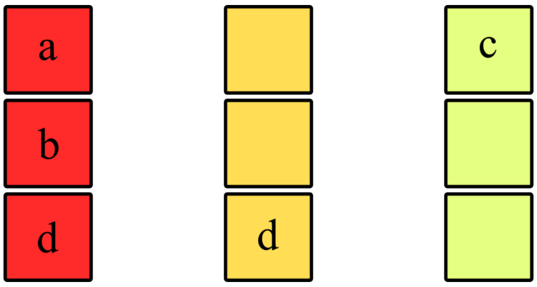
$Rule_1 : a, b \to c$

$Rule_2 : d, \to e|_a$

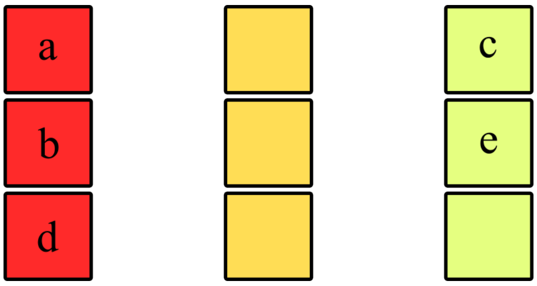
If we calculate these rules sequentially we obtain the following state $w_1 = \{c, d\}$ while in a parallel computation of each rule we obtain $w_1 = \{c, e\}$. In a parallel computation, the item b is consumed by the first rule, but can still be viewed as promoter by the second one. We need to remember what we have in our current state while we are saving in somewhere our produced items and in the meanwhile, we need to know which items we are consuming during our rules application. In the *currentState* we store our items before than we start our rules application, we don't modify this list until the end of our computational step. The *dynamicState* is used to remember us which items are consumed by our rules, to not use the same element twice. In *producedItems* we store the products of our rules and this list is updated as new rules are applied. For instance, if our initial state, as said above is $w_0 = \{a, b, d\}$, our list, at the beginning of our computational step looks like here below.
Where in red we have*currentState* correspondig to w, in yellow *dynamic-*



State corresponding to w' and in green *producedItems* corresponding to $\overline{w}'$. applying the $Rule_1$ we delete a and b from *dynamicState* (w').



Applying the $Rule_2$ we can still see a in *currentState* and produce e deleting d from *dynamicState*.



According to the formalism, this is the way in which we emulate parallelism and the reason because we use three different lists of items.

A.3 Software engine extension

In a second moment, during the development of the theory behind the MAPPS, we began to transform our software engine in one suited to handle multilevel models, such as the one presented in the case of study of Serengeti Lions. Notice that the software has been extended keeping the firm intention of being able to accept both models: *APP Systems* and *MAPP Systems*. We decided to create a new class that extends the software engine, we did not modify old parts of the code, in this way the engine can work on models expressed in both the formalisms.

A.3.1 class Membrane

Membrane is new class that inherits from class *item*, this class is meant to reproduce a membrane of MAPP Systems that could be not only the membrane in which are located elements but also an element itself. The membrane class, contains member functions that initialize inner related rules.

The membrane class, contains member functions that initialize inner state Lists. At this point, we find three internal lists that we already seen in previous version of code as global lists: *currentState*, *dynamicState* and *producedItems* they are used to produce the internal computational step of a membrane. Global lists remain and are used as the lists of Environment Membrane, while each membrane has its own lists with the addition of one more list called *ejectedItems* ad used as a list where we put items ready to be expelled by membrane.

MAPPS formalism requires to reproduce a new kind of rule like this below.

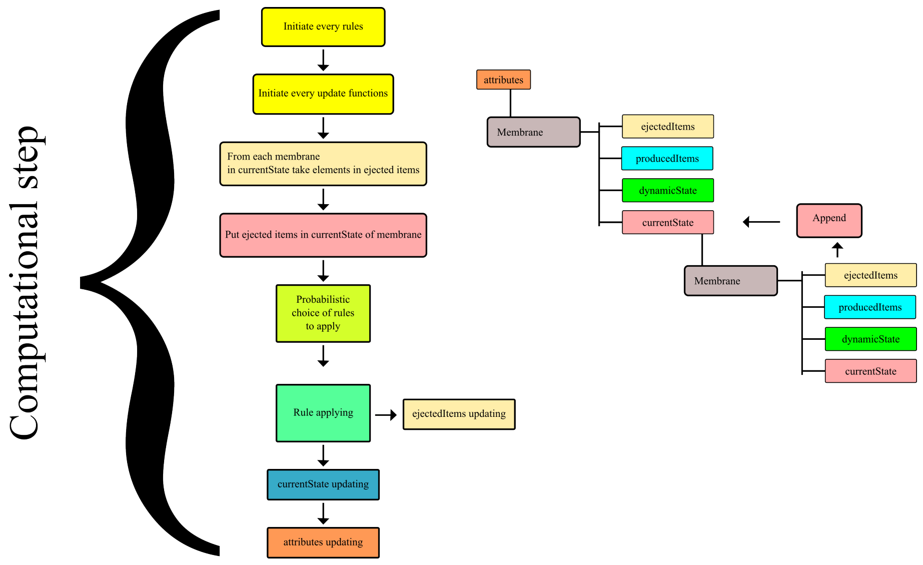
$a \rightarrow k_{out}$

in this case the apply of MAPPS model would be like this:

```
public override void apply(int i, int j, m)
{
k o = new k();
m.ejectedItems.add(o);
}
```

Where *ejectedItems* correspond to w_o of our formalism. To understand better how the computational step of a single membrane works, and how we manage items stored in *ejectedItems* we can give a look to the following graph.

As we can see in graph above, each membrane have four lists: *ejectedItems*, *producedItems*, *dynamicState* and *currentState*. During a computational step, a membrane class loads the entire set of rules that can use, loads the entire set of update function that needs, controls if in its *currenState* there are one or more membranes and if there are, take all elements stored in the *ejectedItems* list of each membrane, and append them in its own currentState, chooses the set of rules to be applied, applies the rules, updates *currentState*, put in *ejectedItems* the elements to be ejected in parent membrane, and update the attributes, according to the loaded set of functions.



According to the MAPPS formalism, the method *computationalStep* of our code, which implements the computational step within a single membrane, implements a visit post-order of the entire tree of membranes, in this way, before the computational step is performed in current membrane, we are sure that is performed in the inner membranes. Gradually, the computational steps are being made in all the membranes, levelling up starting from the bottom.

A.3.2 input files

Even input files are changed in this version of our general purpose engine. While the previously mentioned input files still remain, we need now an entire set of new files, for each membrane we need to read an *update.cs* file and a *rules.cs* file, where are stored, respectively, as saw above, update functions for our membrane and rules of our membrane. So, the user have to create, for each membrane, a folder with the same name of the membrane, where are located *update.cs* and *rules.cs*. We will discuss about *update.cs* file later.

A.3.3 class Membrane implementation

Stepping back to the code we can see something new: the apply function takes no more two variables as input, it takes three variables, the third one is a variable of type membrane, used by us to refer to the membrane that call this method. This because, the same rule should be used in many different membrane with the same type. So we need to know which one called this method to access to its *currentState*. Here below we can see how it is made a rule that belongs to the software engine for MAPP Systems.

```
public class [nameRule] : Rule
{
public [nameRule]()
{
_cat = new Type[1];
Type1 = Female;
cat[0] = Winter;
}
public override int rating(int i, int j, membrane m)
{
return 1;
}
public override void apply(int i, int j, membrane m)
{
Female k = new(Female)
k.Age = k.Age + 1;
Console.WriteLine("I add female to produced items"));
m.producedItems.add(k);
}
}
```

As seen before, [nameRule] stands for the name of our rule, chosen by the user. *_cat* is an array of promoters, the command $\_cat = newType[1]$; sets the number of promoters, in this case one. The command $cat[0] = Winter$; set the type of the promoter. *membranem*, as said above, is necessary to to access attributes and state of the membrane that call the apply or the rating. *producedItems*, as we see, is no more a member of $GlobalDefs$, now it is a member of m.

A.3.4 Membrane Attributes

In MAPPS model, and in MAPP implementation, we have frequently said that membrane are attributed, contrary to what is seen in APP systems, we also said that there are two way to change the value of a membrane attributes: A membrane attribute should be modified by a rule as below:

$r : a_x, B_y \rightarrow B_{x+y}$

this is made by our model, in the apply method as below

```
public override void apply(int i, int j, membrana m)
{
```

```
a.x += b.y;
m.producedItems.add(a);
}
```

the second way is by an update function like this:

$Male_{strenght}((strenght), s) = \Sigma_{a \in S} 1$

We will see in the section below ho we manage this kind of functions in our code. It is important to understand that usually not all the attributes are meant to be changed by rules and not all the attributes are meant to be changed by updating functions. Often, we can see that our set of attributes should be divided in two separated subset, one composed by those attributes changed by updating function, and another one composed by that attributes used only by inner rules when a membrane is used as promoter. In any case the formalism tells us that for each attribute there is an update function, even if in many cases it may be the identity function. In our code, we did not implement a function for each attribute, we just proved a way to easy access membrane attributes letting the users write necessary code to modify them.

A.3.5 Updating functions

In general purpose code for MAPP Systems we need to implement functions update. As we have already said, we have decided not to implement method used as update functions, leaving users free access to the attributes of a membrane and allowing them to write the code needed.

the list *currentState* is a member attribute that we can find in every membrane, so it is easy to refer to it as m.currentState, but it is not so simple to refer to other attributes created by users, because our code can know them only at runtime. The reflection feature comes in our help giving us an easy solution. So for example if we need to access the attribute Strength of our membrane, we can refer to it like using the variable *attributes*["*Strength*"] where *attributes* is a variable of type Dictionary accessed by string that contains object (not items).
once we have gained access to any variable of our membranes due to the attribute *attributes* we can easily modify their contents as here below, where we take all the males present in the *currentState* of a Pride and calculate the total strength of the pride itself

```
int strength=0;

List<object> currentState;
```

```
currentState=m.currentState.getTypeList(typeof(Male));

currentState.ForEach(delegate(object element)
{
Male e = (Male)element;
strength += e.Strength;
});

attributes["Strength"] = strength;
```

as we see above, we obtain from the method *getTypeList* of *currentState* a list of object we launch a simple foreach statement in which we sum the strengths of all male and then we assign to *attribues*["*Strength*"] the resulting value.

A.3.6 Membrane example

We saw before, how was made a class inheriting from item class, now we can see how is made a class inheriting from membrane.

```
public class Coalition : membrane
{
public int health;
public int strength;
public int requirement;

public Coalition(){
    Male o = new Male();
    base.currentState.add(o);
    base.dynamicState.add(o);

    Summer e = new Summer();
    base.currentState.add(e);
    base.dynamicState.add(e);
    }
}
```

As we see above, we just need to write which attributes we want in our class. The items created in constructor method and added to *currentState* and *dynamicState* are that ones used in the initial state of our system.

Bibliography

[1] Bustamante, Javier. "Use of simulation models to plan species reintroductions: the case of the bearded vulture in southern Spain." Animal Conservation 1.4 (1998): 229-238.

[2] Penna, P., Paoletti, N., Scarcella, G., Tesei, L., Marini, M., Merelli, E.: Dispas: An agent-based tool for the management of shing e ort. In: Software Engineering and Formal Methods, pp. 362-367. Springer (2014)

[3] R. Alur, C. Belta, F. Ivanˇci´c, V. Kumar, M. Mintz, G. J. Pappas, H. Rubin, and J. Schug. Hybrid Modeling and Simulation of Biomolecular Networks. Lecture Notes in Computer Science, 2034:19–32, 2001.

[4] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G.: Simulation of spatial P system models. Theoretical Computer Science 529, 11-45 (2014)

[5] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Pardini, G., Tesei, L.: Spatial P systems. Natural Computing 10(1), 3-16 (2011)

[6] Barbuti, R., Bove, P., Milazzo, P., Pardini, G. (2015). Minimal probabilistic P systems for modelling ecological systems. Theoretical Computer Science.

[7] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: An overview on operational semantics in membrane computing. International Journal of Foundations of Computer Science 22(01), 119–131 (2011)

[8] Barbuti, R., Cataudella, S., Maggiolo-Schettini, A., Milazzo, P., Troina, A.: A probabilistic model for molecular systems. Fundamenta Informaticae 67(1), 13–27 (2005)

[9] Barbuti, R., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: A p systems flat form preserving step-by-step behaviour. Fundamenta Informaticae 87(1), 1–34 (2008)

[10] Barbuti, R., Caravagna, G., Maggiolo–Schettini, A., Milazzo, P., Pardini, G.: The calculus of looping sequences. In: Formal Methods for Computational Systems Biology, Lecture Notes in Computer Science, vol. 5016, pp. 387–423. Springer Berlin Heidelberg (2008)

[11] Barbuti, R., Bompadre, A., Bove, P., Milazzo, P., Pardini, G. Attributed Probabilistic P Systems and their Application to the Modelling of Social Interactions in Primates. (2015)

[12] Madhu, M.: Probabilistic rewriting p systems. International Journal of Foundations of Computer Science 14(1), 157–166 (2003)

[13] Barbuti, R. and Levi, F. and Milazzo, P. and Scatena, G., Maximally parallel probabilistic semantics for multiset rewriting, Fundamenta Informaticae, 2011, pp. 1-17

[14] Ciobanu, G., Cornacel, L.: Probabilistic transitions for p systems. Progress in Natural Science 17(4), 432–441 (2007)

[15] Jordán, F., Scotti, M., Priami, C.: Process algebra-based computational tools in ecological modelling. Ecological Complexity 8(4), 357-363 (2011)

[16] M. Curti, P. Degano, C. Priami, and C. T. Baldari. Modelling biochemical pathways through enhanced π-calculus. Theor. Comput. Sci., 325(1):111–140, 2004.

[17] D. Prandi, C. Priami, and P. Quaglia. Process Calculi in a Biological Context. Bulletin of the EATCS, 85:53–69, 2005.

[18] Kahramano gullar, O., Lynch, J.F., Priami, C.: Algorithmic systems ecology: ex-periments on multiple interaction types and patches. In: Information Technology and Open Source: Applications for Education, Innovation, and Sustainability, pp. 154-171. Springer (2014)

[19] C. Priami. “Stochastic π–Calculus”. The Computer Journal, volume 38, number 7, pages 578–589, 1995.

[20] 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, October 2001.

[21] C. Priami, A. Regev, W. Silvermann, and E. Shapiro. “Application of a Stochastic Name–Passing Calculus to Representation and Simulation of Molecular Processes”. Information Processing Letters, volume 80, pages 25–31, 2001.

[22] C. Priami and P. Quaglia. Beta Binders for Biological Interactions. Lecture Notes in Computer Science, 3082:20–33, 2005.

[23] P. Degano, D. Prandi, C. Priami, P. Quaglia "Beta-binders for biological quantitative experiments." Electronic Notes in Theoretical Computer Science 164.3 (2006): 101-117.

[24] Dematte L, Priami C, Romanel A, Soyer O. Evolving BlenX programs to simulate the evolution of biological networks. Theor Comput Sci. 2008;408(1):83–96.

[25] Priami C, Ballarini P, Quaglia P. Computational Methods in Systems Biology. Springer, Berlin Heidelberg; 2009. BlenX4Bio-BlenX for Biologists; pp. 26–51.

[26] Cardona, Campbell, Colomer, M.A., Margalida, A., Palau, A., Pérez-Hurtado, I., Pérez- Jiménez, M.J., Sanuy, D.: A computational modeling for real ecosystems based on P systems. Natural Computing 10(1), 39-53 (2011)

[27] Coria, C.A.N., Tesei, L., Scarcella, G., Russo, T., Merelli, E.: Sea-scale agent-based simulator of solea solea in the adriatic sea. In: Software Engineering and Formal Methods, pp. 259275. Springer (2014)

[28] Ciocchetta F, Hillston J. Bio-PEPA: an extension of the process algebra PEPA for biochemical networks. Electron Notes Theor Comput Sci. 2008;194(3):103–117.

[29] Ciocchetta F, Hillston J. Bio-PEPA: a framework for the modelling and analysis of biological systems. Theor Comput Sci. 2009;410(33-34):3065–3084.

[30] Ciocchetta, F., Hillston, J.: Bio-pepa for epidemiological models. Electronic Notes in Theoretical Computer Science 261, 4369 (2010)

[31] L. Cardelli and A.D. Gordon. “Mobile Ambients”. Theoretical Computer Science, volume 240, number 1, pages 177–213, 2000.

[32] L. Cardelli. “Brane Calculi. Interactions of Biological Membranes”. CMSB’04, LNCS 3082, pages 257–280, Springer, 2005.

[33] Cardelli L. Computational Methods in Systems Biology. Springer, Berlin Heidelberg; 2005. Brane calculi; pp. 257–278.

[34] V. Danos and C. Laneve. “Formal Molecular Biology”. Theoretical Computer Science, volume 325, number 1, pages 69–110, 2004.

[35] V. Danos and S. Pradalier. “Projective Brane Calculus”, Computational Methods in Systems Biology (CMSB’04), LNCS 3082, pages 134–148, Springer, 2005.

[36] Danos V, Feret J, Fontana W, Harmer R. CONCUR 2007 - Concurrency Theory. Vol. 4703. Springer, Berlin Heidelberg; 2007. Rule-Based Modelling of Cellular Signalling; pp. 17–41.

[37] Danos V, Feret J, Fontana W, Harmer R, Krivine J. Rule-based modelling and model perturbation. T Comput Syst Biol XI. 2009;5750:116–137.

[38] C. Laneve and F. Tarissan. A Simple Calculus for Proteins and Cells. Workshop on Membrane Computing and Biological Inspired Process Calculi (MeCBIC’06), to appear on ENTCS.

[39] Internal coarse-graining of molecular systems. Feret J, Danos V, Krivine J, Harmer R, Fontana W Proc Natl Acad Sci U S A. 2009 Apr 21; 106(16):6453-8.

[40] A. Philippou, M. Toro, M. Antonaki, Simulation and Verification for a Process Calculus for Spatially-explicit Ecological Models, Scientific Annals of Computer Science 21, 2011, pp. 1-42

[41] Philippou, A., Toro, M., Antonaki, M.: Simulation and verification in a process calculus for spatially-explicit ecological models. Sci. Ann. Comp. Sci. 23(1), 119-167 (2013)

[42] Philippou, A., Toro, M.: Process ordering in a process calculus for spatially-explicit ecological models. In: Software Engineering and Formal Methods, pp. 345-361. Springer (2014)

[43] M. Antonaki, A. Philippou, A process calculus for spatially-explicit ecological models, Electronic Proceedings in Theoretical Computer Science (EPTCS) 100, 2012, pp. 14-28

[44] H. Matsuno, A. Doi, M. Nagasaki, and S. Miyano. Hybrid Petri Net Representation of Gene Regulatory Network. Pacific Symposium on Biocomputing, pages 341–352, 2000.

[45] Modelling and simulation of signal transductions in an apoptosis pathway by using timed Petri nets. Li C, Ge QW, Nakata M, Matsuno H, Miyano S J Biosci. 2007 Jan; 32(1):113-27.

[46] G. Păun. “Computing with Membranes”. Journal of Computer and System Sciences, volume 61, number 1, pages 108–143, 2000.

[47] G. Păun. Membrane Computing: An Introduction. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[48] G. Păun, G. Rozenberg. “A Guide to Membrane Computing”. Theoretical Computer Science, volume 287, number 1, pages 73–100, 2002.

[49] A. Regev, W. Silverman and E.Y. Shapiro. “Representation and Simulation of Biochemical Processes Using the pi-calculus Process Algebra”. Pacific Symposium on Biocomputing, World Scientific Press, pages 459–470, 2001.

[50] A. Regev and E. Shapiro. “Cells as Computation”. Nature, volume 419, page 343, 2002.

[51] A. Regev and E. Y. Shapiro. Cells as Computation. In CMSB ’03: Proceedings of the First International Workshop on Computational Methods in Systems Biology, pages 1–3, London, UK, 2003. Springer-Verlag.

[52] A. Regev, E.M. Panina, W. Silverman, L. Cardelli and E. Shapiro. “BioAmbients: An Abstraction for Biological Compartments”. Theoretical Computer Science, volume 325, number 1, pages 141–167, 2004.

[53] Robbins, Martha M., and Andrew M. Robbins. "Simulation of the population dynamics and social structure of the Virunga mountain gorillas." American Journal of Primatology 63.4 (2004): 201-223.

[54] Milner R. “Communicating and Mobile Systems: the –Calculus”. Cambridge University Press, 1999.

[55] Milner R. A calculus of communicating systems. Springer; 1980.

[56] Milner R, Parrow J, Walker D. A calculus of mobile processes. I. Inf Comput. 1992;100(1):1–40.

[57] Bortolussi, L., Policriti, A.: Modeling biological systems in stochastic concurrent constraint programming. Constraints 13(1-2), 66 90 (2008)

[58] Spicher, Antoine, Olivier Michel, and Jean-Louis Giavitto. "Spatial Computing in MGS." UCNC. 2012.

[59] Spicher, A., Michel, O., Cieslak, M., Giavitto, J.L., Prusinkiewicz, P.: Stochastic p systems and the simulation of biochemical processes with dynamic compartments. BioSystems 91(3), 458–472 (2008), cited By 23

[60] Petri C. Kommunikation mit Automaten. Ph.D. thesis, Rheinisch-Westfälisches Institut f. instrumentelle Mathematik an d. Univ; 1962.

[61] Reddy V, Mavrovouniotis M, Liebman M. Petri Net Representations in Metabolic Pathways. In: Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology. AAAI/MIT Press, Menlo Park, CA; 1993. pp. 328–336.

[62] Topological analysis of metabolic networks based on Petri net theory. Zevedei-Oancea I, Schuster S In Silico Biol. 2003; 3(3):323-45.

[63] Chaouiya C, Remy E, Ruet P, Thieffry D. Qualitative modelling of genetic networks: From logical regulatory graphs to standard petri nets. Lect Notes Comput Sc. 2004;3099:137–156.

[64] Application of Petri net theory for modelling and validation of the sucrose breakdown pathway in the potato tuber. Koch I, Junker BH, Heiner M Bioinformatics. 2005 Apr 1; 21(7):1219-26.

[65] Application of Petri net based analysis techniques to signal transduction pathways. Sackmann A, Heiner M, Koch I BMC Bioinformatics. 2006 Nov 2; 7():482.

[66] Petri net-based method for the analysis of the dynamics of signal propagation in signaling pathways. Hardy S, Robillard PN Bioinformatics. 2008 Jan 15; 24(2):209-17.

[67] Chaouiya C, Remy E, Thieffry D. Petri net modelling of biological regulatory networks. J Discrete Algorithms. 2008;6(2):165–177.

[68] Vissat, Ludovica Luisa, et al. "MELA: Modelling in Ecology with Location Attributes.

[69] P. Prusinkiewicz, A Lindenmayer. “The Algorithmic Beauty of Plants”. Springer, 1990.

[70] Pescini, Dario, et al. "Dynamical probabilistic P systems: Definitions and applications." Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla (Spain), January 31st-February 4th (2005): 275-288.

[71] Pescini, D., Besozzi, D., Mauri, G., Zandron, C.: Dynamical probabilistic p systems. International Journal of Foundations of Computer Science 17(1), 183–204 (2006)

[72] Book, Ronald V., and Friedrich Otto. String-rewriting systems. Springer New York, 1993.

[73] Metabolic stability and epigenesis in randomly constructed genetic nets. Kauffman SA J Theor Biol. 1969 Mar; 22(3):437-67.

[74] Simulation of large-scale rule-based models. Colvin J, Monine MI, Faeder JR, Hlavacek WS, Von Hoff DD, Posner RG Bioinformatics. 2009 Apr 1; 25(7):910-7.

[75] RuleMonkey: software for stochastic simulation of rule-based models. Colvin J, Monine MI, Gutenkunst RN, Hlavacek WS, Von Hoff DD, Posner RG BMC Bioinformatics. 2010 Jul 30; 11():404.

[76] Rule-based spatial modeling with diffusing, geometrically constrained molecules. Gruenert G, Ibrahim B, Lenser T, Lohel M, Hinze T, Dittrich P BMC Bioinformatics. 2010 Jun 7; 11():307.

[77] Dynamics of HIV infection: a cellular automata approach. Zorzenon dos Santos RM, Coutinho S Phys Rev Lett. 2001 Oct 15; 87(16):168102.

[78] Dynamics of HIV infection studied with cellular automata and conformon-P systems. Corne DW, Frisco P Biosystems. 2008 Mar; 91(3):531-44.

[79] Metropolis, Nicholas, and Stanislaw Ulam. "The monte carlo method." Journal of the American statistical association 44.247 (1949): 335-341.

[80] The P Systems web page: http://psystems.disco.unimib.it/.

[81] Weimar J. Cellular automata approaches to enzymatic reaction networks. Cellular Automata. 2002;2493:294–303.

[82] Cellular automata approaches to biological modeling. Ermentrout GB, Edelstein-Keshet L J Theor Biol. 1993 Jan 7; 160(1):97-133.

[83] Von Neumann J, Burks A. Theory of self-reproducing automata. University of Illinois Press; 1966.

[84] A cellular automaton model of cellular signal transduction. Wurthner JU, Mukhopadhyay AK, Peimann CJ Comput Biol Med. 2000 Jan; 30(1):1-21.

[85] A three-dimensional model of myxobacterial aggregation by contact-mediated interactions. Sozinova O, Jiang Y, Kaiser D, Alber M Proc Natl Acad Sci U S A. 2005 Aug 9; 102(32):11308-12.

[86] Dynamic cellular automata: an alternative approach to cellular simulation. Wishart DS, Yang R, Arndt D, Tang P, Cruz J In Silico Biol. 2005; 5(2):139-61

[87] Dynamic modeling of the central carbon metabolism of Escherichia coli. Chassagnole C, Noisommit-Rizzi N, Schmid JW, Mauch K, Reuss M Biotechnol Bioeng. 2002 Jul 5; 79(1):53-73.

[88] Validation of qualitative models of genetic regulatory networks by model checking: analysis of the nutritional stress response in Escherichia coli. Batt G, Ropers D, de Jong H, Geiselmann J, Mateescu R, Page M, Schneider D Bioinformatics. 2005 Jun; 21 Suppl 1():i19-28.

[89] Hybrid dynamic modeling of Escherichia coli central metabolic network combining Michaelis-Menten and approximate kinetic equations. Costa RS, Machado D, Rocha I, Ferreira EC Biosystems. 2010 May; 100(2):150-7.

[90] Dynamic simulation of the human red blood cell metabolic network. Jamshidi N, Edwards JS, Fahland T, Church GM, Palsson BO Bioinformatics. 2001 Mar; 17(3):286-7.

[91] Dynamic simulation and metabolic re-design of a branched pathway using linlog kinetics. Visser D, Heijnen JJ Metab Eng. 2003 Jul; 5(3):164-76

[92] Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell. Tyson JJ, Chen KC, Novak B Curr Opin Cell Biol. 2003 Apr; 15(2):221-31.

[93] In vivo analysis of metabolic dynamics in Saccharomyces cerevisiae: II. Mathematical model. Rizzi M, Baltes M, Theobald U, Reuss M Biotechnol Bioeng. 1997 Aug 20; 55(4):592-608.

[94] Horn F, Jackson R. General mass action kinetics. Arch Ration Mech An. 1972;47(2):81–116.

[95] Savageau M, Voit E. Recasting nonlinear differential equations as S-systems: a canonical nonlinear form. Math Biosci. 1987;87(1):83–115.

[96] Identification of genetic networks from a small number of gene expression patterns under the Boolean network model. Akutsu T, Miyano S, Kuhara S Pac Symp Biocomput. 1999; ():17-28.

[97] Genetic network inference: from co-expression clustering to reverse engineering. D'haeseleer P, Liang S, Somogyi R Bioinformatics. 2000 Aug; 16(8):707-26.

[98] Inferring qualitative relations in genetic networks and metabolic pathways. Akutsu T, Miyano S, Kuhara S Bioinformatics. 2000 Aug; 16(8):727-34.

[99] Probabilistic Boolean Networks: a rule-based uncertainty model for gene regulatory networks. Shmulevich I, Dougherty ER, Kim S, Zhang W Bioinformatics. 2002 Feb; 18(2):261-74.

[100] Qualitative simulation of genetic regulatory networks using piecewise-linear models. De Jong H, Gouzé JL, Hernandez C, Page M, Sari T, Geiselmann J Bull Math Biol. 2004 Mar; 66(2):301-40.

[101] The yeast cell-cycle network is robustly designed. Li F, Long T, Lu Y, Ouyang Q, Tang C Proc Natl Acad Sci U S A. 2004 Apr 6; 101(14):4781-6.

[102] Approximative kinetic formats used in metabolic network modeling. Heijnen JJ Biotechnol Bioeng. 2005 Sep 5; 91(5):534-45.

[103] Bringing metabolic networks to life: convenience rate law and thermodynamic constraints. Liebermeister W, Klipp E Theor Biol Med Model. 2006 Dec 15; 3():41.

[104] Boolean network analysis of a neurotransmitter signaling pathway. Gupta S, Bisht SS, Kukreti R, Jain S, Brahmachari SK J Theor Biol. 2007 Feb 7; 244(3):463-9.

[105] A structured approach for the engineering of biochemical network models, illustrated for signalling pathways. Breitling R, Gilbert D, Heiner M, Orton R Brief Bioinform. 2008 Sep; 9(5):404-21.

[106] A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. Blinov ML, Faeder JR, Goldstein B, Hlavacek WS Biosystems. 2006 Feb-Mar; 83(2-3):136-51

[107] Cellulat: an agent-based intracellular signalling model. González PP, Cárdenas M, Camacho D, Franyuti A, Rosas O, Lagúnez-Otero J Biosystems. 2003 Feb-Mar; 68(2-3):171-85.

[108] Identifying control mechanisms of granuloma formation during M. tuberculosis infection using an agent-based model. Segovia-Juarez JL, Ganguli S, Kirschner D J Theor Biol. 2004 Dec 7; 231(3):357-76.

[109] Formal agent-based modelling of intracellular chemical interactions. Pogson M, Smallwood R, Qwarnstrom E, Holcombe M Biosystems. 2006 Jul; 85(1):37-45.

[110] Development of a three-dimensional multiscale agent-based tumor model: simulating gene-protein interaction profiles, cell phenotypes and multicellular patterns in brain cancer. Zhang L, Athale CA, Deisboeck TS J Theor Biol. 2007 Jan 7; 244(1):96-107.

[111] Introducing spatial information into predictive NF-kappaB modelling–an agent-based approach. Pogson M, Holcombe M, Smallwood R, Qwarnstrom E PLoS One. 2008 Jun 4; 3(6):e2367.

[112] Agent-based simulation of reactions in the crowded and structured intracellular environment: Influence of mobility and location of the reactants. Klann MT, Lapin A, Reuss M BMC Syst Biol. 2011 May 14; 5():71.

[113] Modeling gene expression with differential equations. Chen T, He HL, Church GM Pac Symp Biocomput. 1999; ():29-40.

[114] John M, Ewald R, Uhrmacher A. A spatial extension to the π-Calculus. Electron Notes Theor Comput Sci. 2008;194(3):133–148.

[115] A model of TLR4 signaling and tolerance using a qualitative, particle-event-based method: introduction of spatially configured stochastic reaction chambers (SCSRC). An G Math Biosci. 2009 Jan; 217(1):43-52.

[116] Rule-based spatial modeling with diffusing, geometrically constrained molecules. Gruenert G, Ibrahim B, Lenser T, Lohel M, Hinze T, Dittrich P BMC Bioinformatics. 2010 Jun 7; 11():307.

[117] Stochastic approaches for modelling in vivo reactions. Turner TE, Schnell S, Burrage K Comput Biol Chem. 2004 Jul; 28(3):165-78.

[118] Bio-calculus: Its Concept and Molecular Interaction. Nagasaki M, Onami S, Miyano S, Kitano H Genome Inform Ser Workshop Genome Inform. 1999; 10():133-143.

[119] Pathway analysis in metabolic databases via differential metabolic display (DMD). Küffner R, Zimmer R, Lengauer T Bioinformatics. 2000 Sep; 16(9):825-36.

[120] BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Blinov ML, Faeder JR, Goldstein B, Hlavacek WS Bioinformatics. 2004 Nov 22; 20(17):3289-91.

[121] Qualitative modelling of regulated metabolic pathways: application to the tryptophan biosynthesis in E.coli. Simão E, Remy E, Thieffry D, Chaouiya C Bioinformatics. 2005 Sep 1; 21 Suppl 2():ii190-6.

[122] BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. Calzone L, Fages F, Soliman S Bioinformatics. 2006 Jul 15; 22(14):1805-7.

[123] A logical model provides insights into T cell receptor signaling. Saez-Rodriguez J, Simeoni L, Lindquist JA, Hemenway R, Bommhardt U, Arndt B, Haus UU, Weismantel R, Gilles ED, Klamt S, Schraven B PLoS Comput Biol. 2007 Aug; 3(8):e163.

[124] Structure-based kinetic models of modular signaling protein function: focus on Shp2. Barua D, Faeder JR, Haugh JM Biophys J. 2007 Apr 1; 92(7):2290-300.

[125] Computational models of tandem SRC homology 2 domain interactions and application to phosphoinositide 3-kinase. Barua D, Faeder JR, Haugh JM J Biol Chem. 2008 Mar 21; 283(12):7338-45.

[126] Essential operating principles for tumor spheroid growth. Engelberg JA, Ropella GE, Hunt CA BMC Syst Biol. 2008 Dec 23; 2():110.

[127] A bipolar clamp mechanism for activation of Jak-family protein tyrosine kinases. Barua D, Faeder JR, Haugh JM PLoS Comput Biol. 2009 Apr; 5(4):e1000364.

[128] Internal coarse-graining of molecular systems. Feret J, Danos V, Krivine J, Harmer R, Fontana W Proc Natl Acad Sci U S A. 2009 Apr 21; 106(16):6453-8.

[129] Towards a genome-scale kinetic model of cellular metabolism. Smallbone K, Simeonidis E, Swainston N, Mendes P BMC Syst Biol. 2010 Jan 28; 4():6.

[130] Grinnell, Jon and Packer, Craig and Pusey, Anne. *Cooperation in male lions: kinship, reciprocity or mutualism.* [*Elsevier*]. 95–105. (1995)

[131] Mosser, Anna and Packer, Craig. *Group territoriality and the benefits of sociality in the African lion, Panthera leo.* [*Elsevier*]. 359–370. (2009)

[132] HEINSOHN, ROBERT. *Group territoriality in two populations of African lions.* [*Elsevier*]. 1143–1147. (1997)

[133] Funston, PJ and Mills, MGL and Biggs, HC and Richardson, PRK. *Hunting by male lions ecological influences and socioecological implictions.* [*Elsevier*]. 1333–1345. (1998)

[134] Woodroffe, Rosie and Frank, Laurence G. *Lethal control of African lions (Panthera leo): local and regional population impacts.* [*Wiley Online Library*]. 91–985. (2005)

[135] Bauer, H and De Iongh, HH and Di Silvestre, I. *lion (panthera leo) social behaviour in the west and central african savannah belt.* [*Elsevier*]. 239–243. (2005)

[136] VanderWaal, Kimberly L and Mosser, Anna and Packer, Craig. *Optimal group size, dispersal decisions and postdispersal relationships in female African lions.* [*Elsevierr*]. 949–954. (2009)

[137] Grinnell, Jon and McComb, Karen. *Roaring and social communication in African lions, the limitations imposed by listeners.* [*Elsevierr*]. 93–98. (2001)

[138] Bygott, J David and Bertram, Brian CR and Hanby, Jeannette P. *Male lions in large coalitions gain reproductive advantages.* [*Nature Publishing Group*]. (1979)

[139] Brendel, Jason. *Geography and Climate.* (2007)

[140] Saundry, Peter *Protected areas.* [*Encyclopedia of earth*]. (2009)

[141] Sinclair, Anthony Ronald Entrican and Arcese, Peter. *Serengeti II: dynamics, management, and conservation of an ecosystem.* [*University of Chicago Press*]. (1995)

[142] Sinclair, Anthony Ronald Entrican. *Serengeti past and present.* [*University of Chicago Press*]. (1995)