Gene Regulatory Networks: extracting models from formal specifications

Adriano di Lauro

October 13, 2010

Summary of the Master Thesis
Master in Mathematics,
Major in Mathematical Logics and Theoretical Computer Science
Università degli Studi Roma Tre
Faculty of Mathematical, Physical and Natural Sciences
Departement of Mathematics
Supervisor: Marco Pedicini

Abstract

A *Gene Regulatory Network* represents a collection of proteins, enzimes and DNA fragments inside a cell, interacting with each other and with other substances in the cell. The study of GRNs became lately a matter of computer science simulation, because of the deterministic nature of protein interactions: any protein has a certain number of sites in which it is possible to bind other specific proteins or substances, and once a site is bound the protein reacts in a deterministic way, for example making new sites visible, creating or destroying links with other proteins, or changing its shape in order to become active in new interactions. Using data bases extracted by experiments and biological studies, one should resume all the known properties of each protein in a logical model in order to make a simulation and see how the system evolves. Final results of such a simulation can be biologically very important to understand new aspects of the evolution of a cell.

The aim of this thesis is to link two mathematical models used in the study of molecular biology: at a lower level *k-calculus*, a process algebra (developed in [1]) which defines processes called *reactions* considering the details of protein's structure, and at a higher level *polynomial dynamical systems*, used to model the global behavior of a GRN in a dynamical system with discrete time steps, by describing at every time the quantity of each gene in the cell. These two models approach the same problem, with different points of view: we want to describe them and show how they can 'communicate'. Our aim is to show how it is possible to link rigorously the microscopic description of k-calculus to the macroscopic behavior of a polynomial dynamical system (through mathematical proves and mathematically founded algorithms, both deterministic and stochastic), and how this link allows to extract better information about GNRs, without loosing the effect of any of the microscopic properties we modelled.

In Figure 1 it is possible to see a schematic description of the thesis: we evidentiated the roles of k-calculus and polynomial dynamical systems, showing with paths how they interact.

The first chapter: formal definition of k-calculus and its properties

k-Calculus, introduced in [1], is a computational model created to describe in detail how proteins and biological molecules interact. In k-calculus it is easy to include microscopic biological information in a model; moreover, *reactions* are defined only on the relevant part of the molecules, it is not necessary to consider the

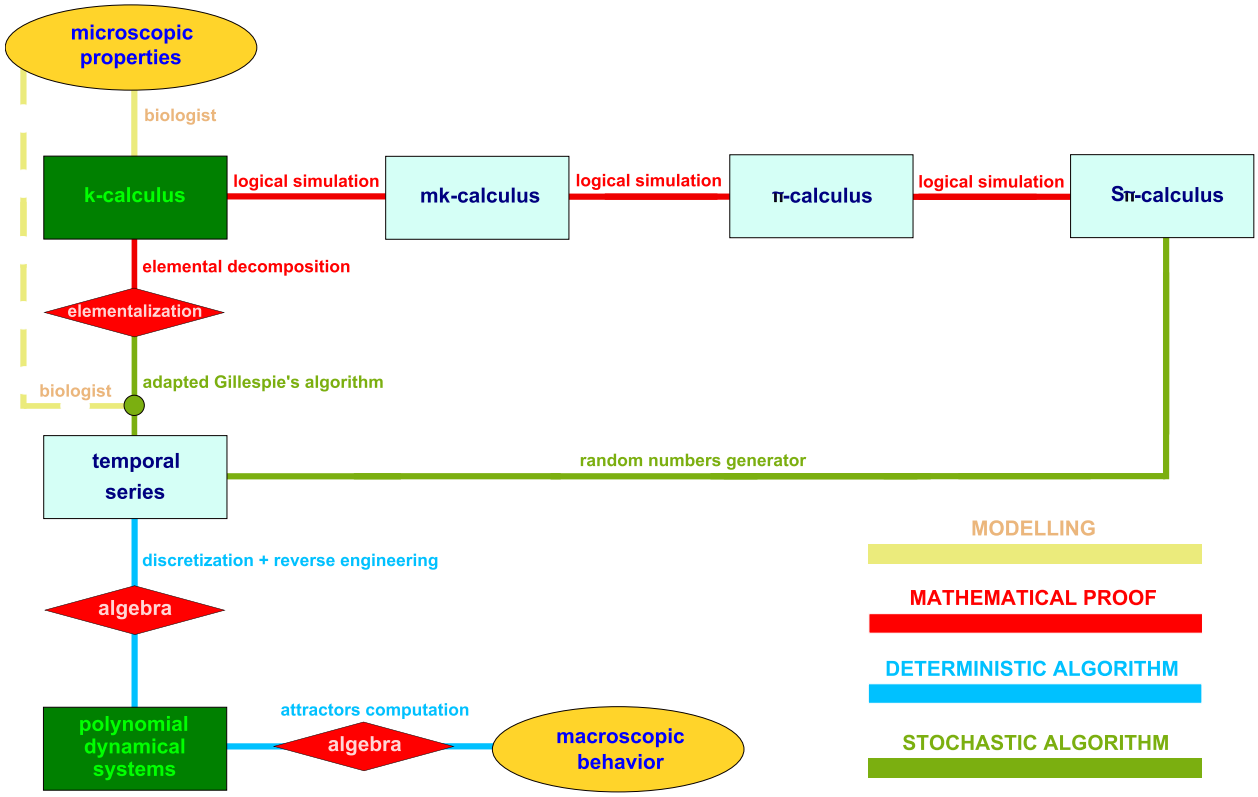

Figure 1: Structure of the thesis

whole protein while modelling a reaction. It is proved (always in [1]) that *k*-calculus is simulated in *mk-calculus*, a finer-grained version of it implementing only binary interactions between single molecules: the meaning of this simulation is that *k*-calculus is a model in which it is easy to encode biochemical reactions, and at the same time it represents a low-level description of protein interactions.

We introduce *k*-calculus formally, as a process algebra. A *process algebra* is a computational model working on *processes*. Each process is abstractly represented by a name extracted by a countable set (as for every computational model). The difference between process algebras and other computational models are some particular operations that are generally allowed: these operations give a meaning to the concept of process. Some common operations are:

- parallel composition of two processes (there are two processes which are updated in parallel);
- nondeterministic binary choice between two processes (one of two processes is updated, while the other one is abandoned);
- sequential composition (when there is an operation which has the priority to be the first available on a particular process);
- communication between processes (a data is sent from a process to another one, through a communication channel).

Besides these operations, characterizing a process algebra, there are notions common to many other computational models, such as bound and free variables, or transition labels.

About the structure of *k*-calculus, a single element of its syntax is called *protein*, and it contains a number of *sites*, which are meant to represent an analogous of sites in real proteins. A site can be either *hidden*,

visible or *bounded*: respectively, a site which cannot be used, a site which can be used but it is free, and a site which is linked by an edge to an other site (for the formal syntax see Definition 1). *Solutions* are defined by induction, as sets of proteins totally defined on their sites.

Definition 1 (*k*-syntax) *k-Calculus' syntax consists of:*

- *a countable set of protein names $\mathcal{P}$; for proteins we use capital letters A, B, etc;*
- *a countable set of edge names $\mathcal{E}$; we use normal letters for edges, x, y, etc;*
- *a map from $\mathcal{P}$ to the set of natural numbers $\mathbb{N}$.*

The interpretation of the map $s : \mathcal{P} \to \mathbb{N}$ is given by $s(A)$ = the number of sites of A. An interface is a partial map ρ, from $\mathbb{N}$ to $\mathcal{E} + \{h, v\}$. If $\rho(\iota) = h$ for a site ι it means that the site is hidden; if $\rho(\iota) = v$ the site is visible; if $\rho(\iota) = x$, $x \in \mathcal{E}$ the site is bounded by x.

The relation defining what is a correct way to let proteins evolve is the *growth relation*: it represents a single step of growth of a protein (creation of a new edge, switching of a site between visible and hidden), and it depends on an edge name set $\{x_1, \cdots, x_m\}$ (shortly $\tilde{x}$), that is the set in which one can choose the edge names to use (see Figure 2 for a graphical representation).

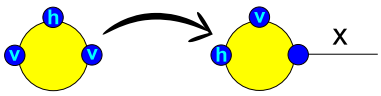

Figure 2: Graphic representation of an occurrence of growth relation: the letters h and v represent respectively *hidden* and *visible* sites, whereas x represents a free name associated to the new edge

Starting from the growth relation defined on single proteins, are defined *k-reactions*, the point of the syntax of *k*-calculus in which it is possible to encode biochemical reactions. Important properties of *k*-reactions are:

- a reaction can be either *monotonic* or *antimonotonic*, i.e., it can only create new edges, or only delete existing ones;
- a reaction is usually defined only on the sites which are biologically relevant for it: formally, this is allowed by defining reactions partially on the set of sites;
- it is also possible to *synthetize* a new protein.

Given a set of *k*-reactions and an initial solution, it is possible to define a transition system, in which to each step corresponds the application of one of the reactions. Elements of such a transition system are *complexes*, sets of proteins connected together through their sites: a set of complexes inside the current solution gives rise to a transition step if it *matches* a reaction. Depending on the shape of given *k*-reactions, the transition system associated to them may be nondeterministic.

Successively to the introduction of *k*-calculus, we show (again from [1]) how it is possible to simulate it in *mk*-calculus (a model describing biochemical reactions at a level of binary interactions between proteins) and π-calculus (the most common and general of process algebras). Besides the interest about simulating a high level language such as *k*-calculus in a finer-graned one such as *mk*, this first simulation is necessary to include *k*-calculus in π-calculus: in fact, π-calculus is a process algebra based on *communication* between processes, and *mk*-calculus naturally introduces communication procedures to coordinate at a binary level bigger *k*-reactions.

The simulation of *k*-calculus in π-calculus is important for logical reasons, because it puts in relation the new language with a well known computational model. Moreover, it is possible to implement on any process of π-calculus a structure of continuous time Markov chain (Definition 2), stochastically simulable

through a simple random number generator: this procedure is described in [2], the new language (called *$S\pi$-calculus*) consists in associating the rate of an exponential random variable to each process, according to its different possible nondeterministic evolutions. Given a process in the syntax of $S\pi$-calculus, together with its starting rates, it is extracted a random number according to each rate, and it is executed the π-transition step associated to the extracted number; then, a set of transition rules describes how to extract new rates associated to the evolutions of the process we obtained (to define transition rules are used properties of exponential variables).

The logical simulation of k-calculus in *mk* and π-calculus gives us the chance to implement in a single $S\pi$-process an initial solution and a set of k-reactions, and simulate its evolution through a random number generators: this simulation algorithm is exact and mathematically rigorous.

In Figure 1 the upper part of the scheme corresponds to this first chapter: microscopic properties of biochemical reactions are modelled in k-calculus, which through a series of logical simulations can be rigorously encoded in $S\pi$-calculus and simulated. The essential properties of k-calculus coming out from this sequence of process algebras are:

- k-calculus allows to model smartly biochemical reactions;
- it describes proteins and molecules in detail;
- since it is simulated in π-calculus, it is logically founded.

The second chapter: exact simulation of k-calculus

In this second part of the thesis we describe how it is possible to simulate exactly a transition system induced by k-calculus. This algorithm takes an important part in linking the microscopic model of k-calculus to macroscopic polynomial dynamical systems (the second model on which we focus, see Definition 5), since it returns as output a simulation which considers all the microscopic details modelled in k-calculus. The shape of output simulations is as *time series*, a succession of vectors $S(t) = (X_1(t), \cdots, X_N(t))$ representing at time t the number of molecules of each kind (the *state* of the system at time t): time series are the input data of a deterministic algorithm (*reverse engineering algorithm*, described in the third chapter) which models them into a polynomial dynamical system.

A simulation of k-calculus through $S\pi$-calculus introduced in the first chapter is not optimal, because:

- we can encode a biological transition system only passing through two previous simulations;
- in a π-process simulating a k-transition system there is a large amount of nondeterministic steps, because we have to put in the same process all the potential evolutions of each sample of protein according to each k-reaction;
- a simulation passing through π-calculus does not reflect at all the actual way proteins move and interact inside a cell.

Hence, we use a different philosophy respect to stochastic π-simulation: we take as a model a classical algorithm, introduced by Gillespie in [3], simulating stochastically a system of general coupled chemical reactions. The main tool necessary for a stochastic algorithm are Markov chains (Definition 2, we mentioned them also in the previous section).

Definition 2 (Markov chain) *A* Markov chain *is a succession of random variables homogeneous in time, which can take values in the same space X. 'Homogeneous' means that at each step of time there exists a fixed probability, depending only on the current state i, to make the variable pass in the new state j during the next step. That is*

$$P\{X_{n+1} = j \mid X_n = i, X_{n-1} = i_{n-1}, \cdots, X_1 = i_1, X_0 = i_0\} = P_{ij}.$$

The chain is said to be continuous time *(shortly called CTMC) if time steps are ranged on a continuous variable in* $\mathbb{R}^+$.

Gillespie's classical algorithm takes as input binary reactions in the shape of rules mapping two reactants (even of the same kind) into a linear combination of molecules (as an example, if we consider molecular species S_1 and S_2, a possible reaction can be $S_1 + S_1 \to S_1 + S_2$). Under hypothesis of a distribution of spheric molecules in a volume in thermal equilibrium, in [3] it is deduced, with calculations of physical kinetics, that for each binary reaction (indexed by μ), there exists a constant c_μ such that

$$c_\mu dt = \text{average probability that a particular combination of } R_\mu \text{ reactant molecules will react accordingly in the next infinitesimal time interval } dt. \quad (1)$$

The expression in (1) is called *fundamental hypothesis of the stochastic formulation of chemical kinetics*. The constant c_μ is deduced from a context of kinetic collisions of particles in a volume: the probability in (1) represents a collision between two occurrences of the molecular species necessary to activate the reaction μ.

The algorithm creates a Markov chain representing the process of collisions between all molecules in the solution. Using the hypothesis in (1), it is built a *joint probability distribution*, $P(\tau, \mu)$, which if multiplied for the infinitesimal time $d\tau$ represents the probability that, given a state $(X_1(t), \cdots, X_N(t))$ at time t, the next reaction will occur in the interval $(t + \tau, t + \tau + d\tau)$ and it will be of the kind μ. At each step, a particular couple $(\overline{\tau}, \overline{\mu})$ is extracted using probabilistic results, starting from a couple of random numbers extracted according to the uniform distribution: then, the current molecular population and the current expression for $P(\tau, \mu)$ are updated, and the process starts again. The algorithm prints the populations $S(t)$ after a given interval, and it stops when it is reached the length requested for the time series.

The important properties of this algorithm are:

- it is *exact*, in terms of probability;
- given the hypothesis in (1), which is deduced by thermodynamics, all the procedure is mathematically justified by theorems of probability.

Now, the principal task we had in this part of the thesis was to adapt the general algorithm exposed in [3] to work on k-reactions and k-solutions. There are two evident problems in doing this:

1. the algorithm in [3] takes as input only binary reactions, while k-reactions can be of any order;
2. a k-reaction is defined only on *part* of k-complexes, this means that, considering k-complexes as molecules inside a volume, there are tipically more ways to apply a k-reaction on the system: reactions do not depend only on collisions as in Gillespie's algorithm, at least not in the shape they have been defined.

To solve the first problem, we considered an extension of the algorithm in [3], made to take as input as well *unimolecular* reactions: in [4], there is a different physical justification to assume also in this case the hypothesis (1). A reaction which is either binary or unimolecular is called *elemental*.

Knowing about the adaptation for elemental reactions of Gillespie's algorithm, we propose a formal definition of an *elemental decomposition* for k-reactions. Starting from a transition system $(S, \mathfrak{A})$, where S is the initial solution and $\mathfrak{A}$ is the set of reactions, we define a procedure to extract an equivalent transition system $([\![S]\!]_{e,\mathfrak{A}}, \mathfrak{E})$, in which $\mathfrak{E}$ is composed only by elemental reactions.

The charachteristics of our elemental decomposition are:

- it is formally correct according to k-calculus syntax;
- to decompose a reaction which is not elemental we consider every combinatorial possibility the complex has to be built, not to exclude any of the ways in which a random simulation may occur;
- to prove correctness of the decomposition (i.e. to prove that executing random steps in $([\![S]\!]_{e,\mathfrak{A}}\mathfrak{E})$ we do not obtain anything that would not be reachable from $(S, \mathfrak{A})$) we introduce conditions
    1. not to let interact two subreactions of different reactions on the same proteins before the first reaction is completed

2. to distinguish more occurrences of the same protein in the same reaction.

We proved completeness and correctness of the elemental decomposition (Proposition 1, Proposition 2); in Proposition 2 the notation T^{ec} represents a *cleanup*, a solution in which all the reactions which have not been completed are cancelled.

Proposition 1 (completeness of the elemental decomposition) *For each set $\mathfrak{A}$ of k-reactions, and for any solutions S, T, if we denote with $\mathfrak{E}$ the elemental decomposition of $\mathfrak{A}$, we have*

$$(S \xrightarrow{\mathfrak{A}}_* T) \Longrightarrow (S \xrightarrow{\mathfrak{E}}_* T).$$

Proposition 2 (correctness of the elemental decomposition) *For each k-transition system $(S, \mathfrak{A})$ (which elemental decomposition is $([\![S]\!]_{e,\mathfrak{A}}, \mathfrak{E})$) we have:*

1. $([\![S]\!]_{e,\mathfrak{A}} \xrightarrow{\mathfrak{E}}_* T) \Longrightarrow (S \xrightarrow{\mathfrak{A}}_* T^{ec})$;
2. *if we call $\mathfrak{A}_m$ the subset of $\mathfrak{A}$ composed only by its monotonic reactions,*

$$[\![S]\!]_{e,\mathfrak{A}} \xrightarrow{\mathfrak{E}}_* T$$

implies that, for each $T' : T^{ec} \xrightarrow{\mathfrak{A}_m}_* T'$, it is not possible to have

$$T^{ec} < T' \leq [\![T]\!]^{-1}_{e,\mathfrak{A}}$$

where $\leq$ represents the subcomplex relation extended to solutions.

Having proposed a way to translate correctly any k-reaction into elemental ones, the remaining problem to use Gillespie's algorithm on k-calculus is the multiple possibility of applying a k-reaction. The solution we propose is to use the classical algorithm only to model *collisions* (we mean also unproperly unimolecular collisions): given the current molecular population $S(t)$, we extract a list of all the k-complexes matching at least a reaction, and through Gillespie we simulate the collision of one of them; then, knowing which collision occurred, we select one of the reactions available using weights associated to each of them. Mainly, we decompose a step of the algorithm in [3] in two steps, one to simulate collisions, one to select the reaction.

The advantage of our method is that we distinguish between the *collision rates*, used to model collisions exactly as in [3], and the *reaction weights*, which represent only the likelihood that each reaction has to be activated if in competition with others. In particular, reaction weights may be directly decided by the biologist according to additional information available.

In Figure 1, the work of this part of the thesis is represented by the path connecting k-calculus to time series: a mathematical proof for the elemental decomposition of k-transition systems, and the stochastic algorithm of Gillespie to simulate them (in which collisions are separated by choice of reactions). As a resume of the properties of this simulation, we say:

- it is done through a stochastic algorithm which is exact;
- we reproduce the thermodynamical conditions under which molecular collisions occur;
- none of the microscopic properties of k-calculus is lost.

The third chapter: using commutative algebra to construct polynomial dynamical systems

In this chapter we describe *polynomial dynamical systems*, the model in which we want to include information derived by the exact simulation of k-calculus. A polynomial dynamical system is a set of n nodes valued

over a finite field $\mathbb{F}$, provided with a *transition function* expressed as a vector of polynomials $F = (f_1, \cdots, f_n)$ such that $f_i \in \mathbb{F}[X_1, \cdots, X_n]$ for each $i = 1, \cdots, n$. In a polynomial dynamical system representing a Gene Regulatory Network each node corresponds to a biochemical molecule (a gene, a protein, etc), the values over $\mathbb{F}$ represent the distribution of the corresponding molecule in the cell, and the transition function a deterministic way to let each node evolve, in function of the previous state of the system.

We progressively define a polynomial dynamical system, to evidentiate its properties.

Definition 3 (time-discrete dynamical system) *A* time-discrete dynamical system *consists of a set of nodes (or vertices) V, each of them being a variable which can have different values on a predetermined discrete space S. To each node v_i is associated a function $f_i : S^N \to S$ to determine its evolution.*

Definition 4 (finite dynamical system) *A* finite dynamical system *is a time-discrete dynamical system in which the time transitions of every node are effectued synchronously.*

Definition 5 (polynomial dynamical system) *A* polynomial dynamical system *is a finite dynamical system, with choice of values in a generic finite field $S = \mathbb{F}$. Functions f_i are expressed in the shape of polynomials in $\mathbb{F}[X_1, \cdots, X_N]$.*

Synchronicity, introduced in Definition 4, allows us to speak about global *states* of the system. As an example of how synchronicity works, let's consider a simple net with three vertices, ranged over $\mathbb{F}_3$, and functions $f_1 = Y^2$, $f_2 = 2X + 1$ and $f_3 = X + 2Z$. If the whole network is in the state $(X, Y, Z) = (2, 0, 2)$, to compute the next state we 'store' the values of the state $(2, 0, 2)$ and we calculate independently each function f_i: in this case the next state of the system is $(0, 2, 0)$; see Figure 3 for the transition space of the states of this system.

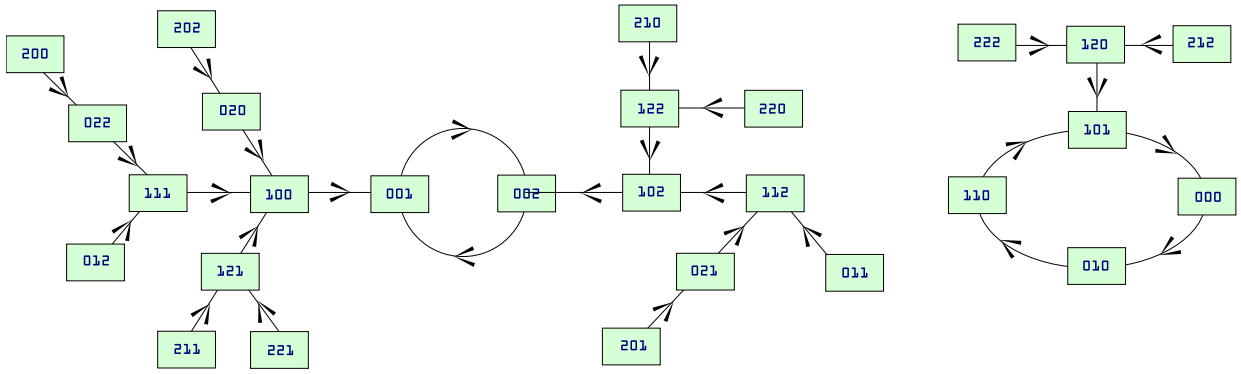

Figure 3: Transition space of the $\mathbb{F}_3$-polynomial dynamical system described by a transition $F = (Y^2, 2X + 1, X + 2Z)$

The characteristics making polynomial dynamical systems good to model the macroscopic behavior of a Gene Regulatory Network are:

- they are synchronous and discrete in time, which makes their dynamics easy to study;
- they support a mathematical structure such as finite field algebra: it is quite easy to range protein quantity on a finite field, because every finite set S can be seen as a field, provided that its cardinality is a prime or a power of a prime, $p^n : n \in \mathbb{N}^\star$, p prime.

We expose from [8] an algorithm to construct a polynomial dynamical system, representing time series produced by a stochastic simulation of the kind of Gillespie (this problem is generally called *reverse engineering*). The algorithm takes as input a series of m states, $\bar{s}_1, \cdots, \bar{s}_m$, such that $\bar{s}_j = (s_{j,1}, \cdots, s_{j,n})$. The states

must be vectors in a finite field $\mathbb{F}$: to have them in this shape it can be applied a *discretization* algorithm to our molecular population of the kind of Gillespie (a vector of integers, where each element represents the exact number of molecules of the corresponding specie). The discretization algorithm (see [9]) uses a *single-link clustering* method, together with a criterion based on entropy, to determine the smallest finite set (of cardinality D) over which it is possible to range the original integers maintaining their distribution: we take the smallest prime $p \geq D$, and riarrange the values over $\{0, \cdots, p-1\}$, which has the same cardinality of the finite field $\mathbb{F}_p$.

Starting from a time series of states in $\mathbb{F}_p^n$, the algorithm constructs a polynomial interpolating $m-1$ points, f_i^0, such that

$$f_i^0(s_{j,1}, \cdots, s_{j,n}) = s_{j+1,i}, \tag{2}$$

for each $j = 1, \cdots, m-1$: this is done by using the classical interpolation basis of Lagrange. Once a particular solution $(f_1^0, \cdots, f_n^0)$ has been constructed, the typical problem is how to choose the best sample of transition function interpolating the points in (2). In [8], it is suggested for each f_i^0 a condition of *minimality*, respect to the ideal I of all the functions vanishing over the set $\{\overline{s}_j \mid j = 1, \cdots, m-1\}$: each polynomial f_i^0 is reduced modulo I, to obtain the final transition function $(f_1, \cdots, f_n)$. The criterion of minimality for f_i chosen in [8] has the aim to delete from f_i^0 every part which would model a behavior not supported by the simulation.

So, the reverse engineering algorithm needs two subalgorithms to be completed:

1. an algorithm to reduce a polynomial $f \in k[X_1, \cdots, X_n]$ modulo an ideal I, i.e. to find a polynomial r such that $f = h \cdot g + r$;
2. an algorithm to compute the ideal vanishing on a given set of points.

In computational algebra of polynomial rings over a field k, problems such as *ideal description* and reduction of polynomials over an ideal are solved as a consequence of the definition of *Gröbner bases*. A Gröbner basis for an finite set of polynomials $\{f_1, \cdots, f_s\}$ is a basis for which the division algorithm of a polynomial f modulo $\{f_1, \cdots, f_s\}$ leads to a solution $f = a_1 f_1 + \cdots + a_s f_s + r$, where the remainder r is univoquely determined.

Definition 6 (monomial ordering) *A* monomial ordering *on* $k[X_1, \cdots, X_n]$ *is a relation* $>$ *on* $\mathbb{Z}_{\geq 0}^n$ *(which corresponds to a relation on the set of monomials* $\{X^\alpha \mid \alpha \in \mathbb{Z}_{\geq 0}^n\}$*) satisfying:*

1. $>$ *is total;*
2. *if* $\alpha > \beta$ *and* $\gamma \in \mathbb{Z}_{\geq 0}^n$ *then* $\alpha + \gamma > \beta + \gamma$*;*
3. $>$ *is a well-ordering on* $\mathbb{Z}_{\geq 0}^n$*, this means that every nonempty subset of* $\mathbb{Z}_{\geq 0}^n$ *has a smallest element respect to* $>$*;*

Definition 7 (multidegree) *The* multidegree *of a polynomial* $f_i(X_1, \cdots, X_n) \in k[X_1, \cdots, X_n]$ *is defined by*

$$\mathrm{mdeg}(f) := \max_{\mathbb{Z}_{\geq 0}^n}\{\alpha \mid a_\alpha \neq 0\}.$$

The leading coefficient, leading monomial *and* leading term *are defined by*

$$\begin{aligned} LC(f) &:= a_{\mathrm{mdeg}(f)} \in k, \\ LM(f) &:= X^{\mathrm{mdeg}(f)}, \\ LT(f) &:= LC(f)LM(f). \end{aligned}$$

The extension of the euclidean division with remainder algorithm to polynomial rings of more than one variable is affected to the lack of a unique acceptable polynomial degree: depending on the particular *monomial ordering* chosen (see Definition 6) and on the order in which the f_i are listed, some monomials of f may be divided in different ways giving rise to different remainders. About this problem we expose, from [5], classic studies in computational algebra; in sequence,

- we define monomial orderings and monomial ideals;
- we give examples of uniqueness problems deriving from the division algorithm;
- we prove the Hilbert basis theorem for polynomial rings (Theorem 1);
- we characterize Gröbner bases and expose the classical algorithm (Buchberger) to compute them from a finite set of polynomials.

For Hilbert basis theorem every ideal has a finite generator set, for which it is possible to compute an equivalent Gröbner basis; moreover, it's possible to constrain the Gröbner basis under certain conditions which make it unique (the unique *reduced* Gröbner basis).

Theorem 1 (Hilbert basis theorem for polynomial rings) *Every ideal $I \subset k[X_1, \cdots, X_n]$ has a generating set $\{g_1, \cdots, g_t\}$, with finite number of elements, such that $\langle LT(I)\rangle = \langle LT(g_1), \cdots, LT(g_t)\rangle$.*

Definition 8 (Gröbner basis) *Given an ideal $I \subset k[X_1, \cdots, X_n]$, a* Gröbner basis *for I is a finite set $\{g_1, \cdots, g_t\}$ generating I and such that $\langle LT(g_1), \cdots, LT(g_t)\rangle = \langle LT(I)\rangle$.*

Gröbner basis is a useful tool for any algorithm which handles polynomials and polynomial ideals, because it gives an univoque result to the division algorithm in more than one variable (this fact implies many regular properties, for example if $f \in I$ then the division of f modulo a Gröbner basis for I gives always remainder zero). About our particular problem, the reduction modulo I is achieved with the division modulo a Gröbner basis for I, so the only thing remaining to do is to compute the reduced Gröbner basis for the ideal vanishing on each point of the series. For this there are many algorithms, based on different ideas: the most efficient one is generally the algorithm exposed in [7], which uses a linear algebra method (*row-reduction of a matrix*). We chose to describe another algorithm, found in [6], which is not as straightforward as the one in [7] but is based on a theoretical proposition of commutative algebra (Proposition 3).

Proposition 3 *A set $\{g_1, \cdots, g_t\} \subset I$ is a Gröbner basis for $I = I(P_1, \cdots, P_m) = I(V)$ if and only if $|\mathcal{B}(g_1, \cdots, g_t)| = m$.*

The overall complexity of the reverse engineering algorithm, in its most efficient version, is exponential in the length of input time series and polynomial in the number of points. Resuming the essential properties of polynomial dynamical systems, we can say:

- they model biological events, in the shape of time series representing the evolution of the system, into an automaton reproducing the macroscopic effects of initial gene distributions in a Gene Regulatory Network;
- they are synchronous, so we are allowed to study their behavior through global states;
- their transition function is defined as a polynomial, hence we can work in a well-known mathematical framework such as commutative algebra and finite field algebra;

The fourth chapter: extracting information about macroscopic behavior

In this last part of the thesis we show how to deduce informations about Gene Regulatory networks from the dynamics of a polynomial dynamical system. A PDS can be seen as a discrete analogous of continuous dynamical systems generated by differential equations: applying iteratively the transition function F we move deterministically inside the transition space, with steady states and cycles.

Definition 9 (descendants, ancestors) *The* descendants *of a state s are all the states reachable from it, whereas its* ancestors *are the states from which it is possible to reach it. Mathematically, s' is descendant of s (s is an ancestor of s') if there is a succession of intermediate states s_j such that*

$$F(s) = s_1, F(s_1) = s_2, \cdots, F(s_m) = s'.$$

Definition 10 (cycles) *A* cycle *is a subset C of the set of states maximal respect to the property that every state in C is both ancestor and descendant of every other state. An* attractor *is a subset A of the set of states such that every descentant of an element in A still belongs to A. The* basin of attraction B_A *of an attractor A is the set of all the ancestors of elements in A: mathematically,*

$$B_A = \{s \mid \exists s' \in A : s \text{ is ancestor of } s'\}.$$

According to Definition 9 and Definition 10, and considering the discrete nature of the system, we can have a precise idea of the global dynamics of a polynomial dynamical system. The initial state chosen for an execution of the model determines the particular basin of attraction containing all the following states (see Figure 3); after a finite number of steps, the dynamics enters into an attractor (that is, a cycle: in our case, working in a discrete set, the two definitions coincide). In the biological interpretation of polynomial dynamical systems, an attractor is a state reached by the cell under certain initial conditions (a certain amount of each of its proteins and genes): for example the process of a cell reaching an attractor can correspond biologically to *cell differentiation*, being it the way a general kind of cell differentiates into more specific ones. There are two kinds of cell differentiation which can be represented by polynomial dynamical systems: if the attractor is composed only by a single state (a cycle of length one), the cell reaches that state and it does not change anymore; if otherwise the attractor is a cycle with more than one element, the cell oscillates periodically between different gene expressions. In this interpretation, the length of an attractor (the number of states in the cycle) represents the evolution time of the cell. It is also possible to study the sensitivity of attractors to disturbances (changes of the network, changes of the state of a node, etc.), to have results about the corresponding stability of the cell to damage or mutations.

So, in order to understand the evolution of a Gene Regulatory Network, the main result we want is to compute attractors of a PDS and their length (in Figure 1, this part of the thesis is represented by the last path on the bottom). We present two subalgorithms to simplificate preliminarly a polynomial dynamical system:

- first, we remove redundancy from the system (Definition 12, see also [10]), this simplification is useful since it reduces the number of nodes: usually it is very common to find redundancy in a Gene Regulatory Network, because necessary to prevent consequences of genetic damages in a cell;
- second, we reduce our attentions on *connected components* of the associated graph (see Definition 11): attractors of the whole system can be combinatorially calculated by the ones of each component.

Definition 11 *In a polynomial dynamical system, we say that the node v_i* influences *another node v_j if the function f_j describing the evolution of v_j is not trivial on the variable X_i. It is possible to define an oriented* associated graph *$\langle E, V \rangle$, in which V is the set of nodes in the polynomial dynamical system, and an oriented edge is created between two nodes v_i and v_j if v_i influences v_j.*

Definition 12 (redundancy) *A node $v \in V$ of a polynomial dynamical system is* redundant *if the system obtained from G by removing v has the same number and length of attractors. If a node is not redundant, it is called* relevant.

Finally, we expose an algorithm to compute attractors and their lengths (see [12]). The main problem of computing attractors is the exponential size of the transition space of a PDS (which has cardinality p^n): we can find quickly the attractor corresponding to each single initial state (by computating iteratively the composition of F by itself, and reducing modulo p exponents of the intermediate polynomials), but it's not possible to know which state must be checked to find an attractor which hasn't been calculated yet. Moreover, to be sure that we computed *all* the attractors, we should check every possible initial state.

The algorithm in [12] looks for a path of increasing length k in the transition space: when such a path is found and it contains a double occurrence of the same state, it means that it contains an attractor, and we start looking for another path, always of increasing length, with the constraint that it does not contain the attractors we have already found. When it is not possible to find anymore a path of the current length, it means that we calculated all the attractors.

In [12] this procedure is implemented by using *multi-valued boolean formulas* to describe a sequence of k states in the transition space (and an additional condition to avoid ending up in an already found attractor), and by solving them with the Davis-Putnam algorithm for multi-valued formulas (MV-SAT solver, see [11]). The MV-SAT solver proceeds with an exhaustive search of all possible values trying to satisfy the formula, but avoiding to check useless variables by pruning subtrees in exaustive search: in the worst case, time of execution is exponential, but this exaustive search is performed in an efficient way and in practice, many heuristic optimizations are implemented in SAT-solvers.

We propose a different implementation of the algorithm in [11], using Gröbner basis. We want to represent the condition of a set of states $s_q, \cdots, s_r$ to be in sequence with the system of equations

$$\begin{cases} F(X_q[1], \cdots, X_q[n]) & = & (X_{q+1}[1], \cdots, X_{q+1}[n]) \\ & \vdots & \\ F(X_{r-1}[1], \cdots, X_{r-1}[n]) & = & (X_r[1], \cdots, X_r[n]). \end{cases} \tag{3}$$

The system in (3) can be represented with an ideal (that we call $I_{q\cdots r}$) of the polynomial ring

$$\mathbb{F}_p[X_q[1], \cdots, X_q[n], X_{q+1}[1], \cdots, X_{q+1}[n], \cdots, X_r[1], \cdots, X_r[n]]$$

which has dimension $n(r-q)$ over $\mathbb{F}_p$. The ideal is generated by the polynomials

$$f_i(X_j[1], \cdots, X_j[n]) - X_{j+1}[i], \qquad \text{for } j = q, \cdots, r-1 \text{ and } i = 1, \cdots, n.$$

To complete the analogy between the system in (3) and the Boolean formula used in [12], we add to (3) a polynomial assuming a value different by zero on each element of the already found attractors, and vanishing on each other state (we obtain it by interpolation on a finite field): this way, at each step of the algorithm we have a system of polynomial equations F_P, in nk variables (n copies for each element of the path of length k) whose solution (if there exists) is a sequence of k states which doesn't end in one of the attractors already found.

To find at each step a set of states satisfying F_P, we propose to reduce it preliminarly through the computation of its reduced Gröbner basis: this simplificates the system, thanks to the elimination theorem (Theorem 2).

Theorem 2 (elimination theorem) *Let $I \subset k[X_1, \cdots, X_n]$ be an ideal with Gröbner basis G, respect to lexicographic order with $X_1 > X_2 > \cdots, > X_n$. Then, for every $0 \leq k \leq n$, the set $G_k = G \cap k[X_{k+1}, \cdots, X_n]$ is a Gröbner basis of I_k.*

The way Theorem 2 can be applied is: starting from a set of equations $f_1, \cdots, f_s \in k[X_1, \cdots, X_n]$, the reduced Gröbner basis calculated according lex order eliminates, as far as it is possible, variables from the last equations. From algebraic geometry (see Proposition 4) we know that solving the system composed by the reduced Gröbner basis is equivalent to solve F_P: knowing this, we can try every possible value for the last equation, which depends on as less variables as possible, and substitute their solution in the previous ones, coming back to change the current values if we find a conflict. This method, analogously to SAT, is exponential in the worst case but it is usually quicker since it proceeds eliminating useless solutions.

Proposition 4 *$\mathbb{V}(I)$ is an affine variety: in particular, if $I = \langle f_1, \cdots, f_s \rangle$, then $\mathbb{V}(I) = \mathbb{V}(f_1, \cdots, f_s)$.*

As a resume of this last part of the thesis, we say:

- in a polynomial dynamical system representing a Gene Regulatory Network, the most important information about macroscopic behavior of the GNR derives from attractors and their lengths;
- through optimal algorithms it is commonly possible to compute attractors in a time less than exponential;
- following the thesis from k-calculus to extraction of attractors from a PDS, we find macroscopic behaviors which are consequence of all microscopic details of the description of proteins and molecules.

Original contributions of the thesis

In this thesis we present the following original contributions:

- **a formal decomposition of any k-reaction into elemental reactions** which still fit into the syntax of k-calculus: we prove this decomposition to be *complete* (Proposition 1) and *correct* (Proposition 2), moreover it considers all the combinatorial possibilities to assemble the original k-complex, hence it can be used in a simulation modelling completely the original set of reactions;
- an **adaptation of Gillespie's stochastic algorithm to work on k-reactions**, by modelling in the original reaction constants only physical parameters leading to molecular collision, and separately, into *balanced weights*, the likelihood each reaction has to be activated if in competition with other reactions;
- a **direct algebraic way** to implement the algorithm in [12], computating attractors in a polynomial dynamical system: instead of using Boolean formulas, we model a path of length k, which doesn't fall in any of the attractors already found, in a system of polynomial equations, and use Gröbner basis and the elimination theorem to reduce and solve the system.

References

[1] Vincent Danos, Cosimo Laneve *Formal molecular biology - TCS 325, 2004*.

[2] Corrado Priami *Stochastic π-calculus - The Computer Journal 38 (1995) 578-589*

[3] Daniel Gillespie *Exact stochastic simulation of coupled chemical reactions - The Journal of Physical chemistry, Vol. 81, No. 25, 1977, 2340-2361*

[4] Daniel T. Gillespie *Simulation methods in system biology - Lecture Notes in Computer Science (2008) Vol 5016/2008, 125-167*

[5] D. Cox, J. Little, D. O'Shea *Ideals, varieties, and algorithms - Springer (1997)*

[6] J. Farr, S. Gao *Computing Gröbner bases for vanishing ideals of finite sets of points - AAECC-16 (2004) 118-127*

[7] J. Abbott, A. Bigatti, M. Kreuzer, L. Robbiano *Computing ideals of points - Journal Symbolic Computation 30 (2000) 341-356*

[8] Reinhard Laubenbacher, Brandilyn Stigler *A computational algebra approach to the reverse engineering of gene regulatory networks - Journal of Theoretical Biology 229 (2004) 523-537*

[9] Elena S. Dimitrova, John J. McGee, Reinhard C. Laubenbacher *Discretization of time course data - http://polymath.vbi.vt.edu/discretization/DimitrovaMcGeeLaubenbacher.pdf*

[10] Elena Dubrova, Maxim Teslenko, Andres Martinelli *Kauffman networks: analysis and applications - ICCAD 2005: 479-484*

[11] Cong Liu, Andreas Kuehlmann, Mattew W. Moskewicz *CAMA: a multi-valued satisfiability solver - http://www.cadence.com/cadence/cadence_labs/Documents/kuehl_ICCAD_2003_Cama.pdf*

[12] Elena Dubrova, Maxim Teslenko, Liu Ming *Finding attractors in synchronous multiple-valued networks using SAT-based bounded model checking - ISMVL 2010: 144-149*