MASTER IN COMPUTER SCIENCE

Design of a support system for modelling gene regulatory networks

Wattiez, Morgan

Award date:
2015

Awarding institution:
University of Namur

University of Namur
Faculty of Computer Science
Academic Year 2014–2015

Design of a support system for modelling gene regulatory networks

Morgan WATTIEZ

Supervisor: _____ (Signed for Release Approval
Jean-Marie JACQUET Study Rules art. 40)

Thesis submitted in partial fulfillment of the requirements
for the degree of Master in Computer Science at the University of Namur

Abstract

The understanding of gene regulatory networks depends upon the solving of questions related to the interactions in those networks. This study shows how a promising programming paradigm, constraint logic programming, can be used to design a support tool for modelling biological networks, with the emphasis on gene regulatory networks. We describe the most important elements of those complex regulatory networks, as well as popular formal methods and tools designed for their modelling.

Then we introduce to the constraint logic programming paradigm and its advantages for solving real world as biological problems. Our approach is to use this paradigm to design a simple support tool with the aiming to help in modelling gene regulatory networks. For instance we set the emphasis on the understanding of the nature of interactions in those networks. To illustrate our approach, we designed BioNet, and present its interesting capabilities. Finally, we illustrate some of its features with the modelling of the regulation in the *lac* operon.

Keywords

Gene regulatory networks, biological networks, network inference, Prolog, constraint logic programming.

Acknowledgements

I wish to express my gratitude to my supervisor, Professor Jean-Marie Jacquet, for his continuous encouragement as well as for providing me with all the necessary support for the research, his time, criticism, advices, and correction to this master thesis from the beginning.

I take this opportunity to express gratitude to all the faculty members for their help and support during the three last years of study at this university.

I would like to thank my friends and colleagues at work for their cheerfulness, unceasing support and encouragement.

I am also grateful to Frédérique who supported me through this venture, for her great patience and attention during the last three years devoted to courses and this master thesis.

In addition, thanks to one and all, who directly or indirectly, have let their hand in this unforgettable venture ...

Contents

Acronyms

AA amino acid.
ACM Association for Computing Machinery.
API Application programming interface.

BDD Binary Decision Diagram.
BioPAX Biological Pathway Exchange.
BN Bayesian Network.

cAMP cyclic adenosine monophosphate, or cyclic AMP.
CAP catabolite activator protein.
CLP constraint logic programming.
CLP(FD) constraint logic programming over finite domains.
CP constraint programming.
CSP constraint satisfaction problem solver.
CSP constraint satisfaction problem.
CTL Computational Tree Logic. *Glossary:* CTL.
CTRL Computational Tree Regular Logic. *Glossary:* CTRL.

DEG differentially expressed gene.
DNA deoxyribonucleic acid.

GN gene network.
GRN gene regulatory network.
GTM Graph theoretical model.
GXL Graph eXchange Language.

HGP Human Genome Project.

JDK Java SE Development Kit.
JPL Java-calls-Prolog API. *Glossary:* JPL.
JSON JavaScript Object Notation.
JVM Java virtual machine (JVM).

KEGG KEGG Pathway database.
KGML KEGG Markup Language.

LP logic programming.
LTL Linear Temporal Logic.

mRNA messenger RNA.

ODE ordinary and partial differential equation.

PADE Piecewise affine differential equation.
PBN probabilistic boolean network.
PPI protein−protein interaction network.

REST Representational State Transfer.
RNA ribonucleic acid.
rRNA ribosomal RNA.

SAT Boolean Satisfiability solvers.
SBGN Systems Biology Graphical Notation.
SBML Systems Biology Markup Language.
SIF Simple Interaction Format.
SSM state-space model.
STG state transition graph.

TF transcription factor.
tRNA transfer RNA.

XML Extensible Markup Language.

Glossary

activator a transcription factor that *increases* the rate of transcription and consequently has a positive impact on the rate of protein synthesis.
allele an alternate form of a gene.
atom atoms are the most fundamental unit of matter and join together into clusters called molecules.

chromosome a large structure present within living cells. It is made of a long stretch (segment) of DNA that represents many genes and proteins which associate to DNA. The chromosomes are responsible for storage, duplication, expression and evolution of DNA.
CTL a formal specification language used in model-checkers.
CTRL an extension of CTL (*Computational Tree Logic*) with the support of regular expressions.
Cytoscape an open source bioinformatics software platform for visualizing complex networks and integrating these with any type of attribute data.

DNA a nucleic acid which, in addition to proteins and fats, is one of the macromolecules present in the organism.

enhancer a particular DNA sequence that can be bound to activators to increase the level of transcription of a gene or multiple genes by activating the promoter region. It functions as a "turn on" switch of gene expression.
enzyme a macromolecular biological catalyst. The enzymes accelerate, or catalyze, chemical reactions. The molecules at the beginning of the process are called substrates. The enzyme converts these into different molecules, called products.
eukaryote a multicellular organism having a true nucleus as well as organelles within membranes. Eukaryotic cells are larger than prokaryotic cells. Cell division in eukaryotes can be either sexual or asexual, and involve processes of meiosis or mitosis. In meiosis, the produced cell is the result of the recombination of two parental chromosomes. In mitosis, a cell divides to produce two identical cells. In comparison, with prokaryotic cells, eukaryotes are much larger and complex. Examples are animals, plants and fungi.

gene a functional unit, structurally an individual segment of DNA, which contain the instruction to produce a functional product (RNA or protein).
genetics the study of biological information with the emphasis set on heredity, genes and genes variation in all living organisms.
genome it comprises all chromosomes and DNA sequences present in an organism. It is the provider of the complete hereditary information present in an organism and its individual cells. Functionally the genome is divided into genes while structurally it is divided into different DNA molecules or chromosomes.
GINML an extension of GXL which is used for the definition of logical regulatory graphs.
GXL Graph eXchange Language (GXL) is designed to be a standard exchange format for graphs.

inhibitor a sort of biochemical signal which prevents the expression of a particular gene even in the presence of an appropriate activator.
insulator a segment of DNA that blocks the interaction between an enhancer and a promoter.

JPL a bidirectionnal interface between Java and Prolog. It is provided as a part of SWI-Prolog and requires both Java SDK and SWI-Prolog.

macromolecule larger molecules made of long chains of individual molecules linked together.
metabolite a product of metabolism which as various functions such as fuel, structure, signaling, stimulatory and inhibitory effects on enzymes.
model checking a verification technique that consists in verifying that a system model meets or not a specification.
molecule the simplest unit of chemical compound, made of a group of two or more atoms linked together.

nucleotide the building blocks of DNA.

operator a segment of DNA to which a transcription factor binds to regulate gene expression. Usually, this transcription factor is a repressor that binds to the operator to prevent transcription.
operon an operon, from the French *opérer* (to operate), is a functional unit of DNA composed of two or more genes.
organelle a specialized subunit having a specific function in a cell. It consists of a tiny structure made of biological molecules. Organelles are to cells what an organ is to the body.

polypeptide a chain of amino acids (or polypeptide).

prokaryote a single-cell organism characterized by the lack of a nucleus and membrane-bound organelles. Prokaryotic cells are small and reproduce asexually, by binary fission or budding. They are present in nearly all environments, even under hard conditions such as extreme temperatures. Examples are bacteria and archaea.
promoter a region of DNA that initiates the transcription of a particular gene.
protein a large biological molecule, or macromolecule, that consists in one or more chains of amino acids. Each protein has its own unique amino acid sequence that depends on the sequence of the gene that encoded this protein. This particular sequence gives to the protein the three-dimensional structure that determines its function.

repressor a transcription factor that *inhibits* the rate of transcription by binding to a promoter and consequently has a negative impact on the rate of protein synthesis.
ribosome The ribosomes are the cellular "machines" driving protein synthesis by taking a chain of nucleotides (RNA) and translating it into a chain of amino acids (or polypeptide) which later folds into a protein.

silencer a particular DNA sequence that functions as a "turn off" switch that decreases the level of transcription.

trajectory a serie of state transitions.

XGMML the Cytoscape standard file format for saving graph layouts.

Chapter 1

Introduction

The GRNs (Gene regulatory networks) are at the heart of numerous processes vital for all the living organisms. Their study is an essential step to understand the mechanisms involved in the expression of particular genes and the production of specific gene products in reaction to environmental stimuli. Those GRNs networks play an important role in resistance to disease as well as the evolution of organisms, and their study has become a field of active research.

To achieve the study of those networks, numerous formal methods and modelling tools have been proposed in the last decades, with the emphasis on different dynamics depending on the nature and complexity of the studied phenomenons. However, the accurate modelling of such networks is difficult due to lacking data, and the many unknowns in the understanding of the interactions, especially for large networks. Some interesting qualitative modelling approaches have been developed to face the lack of data. Examples are the Boolean networks developed by Kauffman [1, 2] as well as Thomas' formalism [3–5]. They enable to reason on the dynamics of those networks with simple modelling approaches, in addition those formalisms have been largely described and improved until the present day, in the form of numerous model extensions with their pros and cons. However, the understanding of the nature of interactions is still a long and difficult process, and some questions are quite difficult to answer. In addition, the modelling of a particular phenomenon and the interpreting of gene expression data often requires intuition from the biologist researcher to identify the key interactions in GRNs.

In this thesis, we explore a promising technique to solve problems with many unknowns, with the aiming of using this technique to help the biologist researcher at his work on regulatory networks. For that purpose, we explore the constraint logic programming paradigm which is well suited to solve highly constrained problems. Interest is focused on their ability to find answers to complex questions involving unknowns

and uncertainty. Constraint solving techniques are studied, as well as their advantages and the perspective of using them to answer recurrent questions in the context of gene regulatory networks.

The first chapter is an introduction to genetics and its history, with the emphasis on gene regulatory networks and its mechanisms. It illustrates the key components involved in biological systems, from atoms to complex organisms, and explains notions such as DNAs, RNAs, genes and proteins as well as the interactions between them. The key processes of gene expression, gene regulation and gene regulatory networks are explained. We explain some techniques used to measure the gene expression, then we illustrate some gene regulatory networks and we give an example with the gene regulation in the *lac* operon. Then we describe some databases used by biologist researchers to exchange information on biological networks.

The second chapter focuses on the existing formal methods and tools used to achieve a more precise description of the dynamics of GRNs, with the emphasis on the most relevant formalisms covered in the literature from the 1960s. We give a general introduction to the gene network modelling approaches, the main issues in this particular context, such as the inherent noise in data, or the natural complexity of biological networks and their interactions. We present the modelling techniques used to achieve a precise description of GRNs, as well as their features. We propose a classification of the formalisms, regarding to their features, and we take the opportunity to propose a comparison of those formalisms with their extensions included. Numerous tools have been developed, based on these formalisms, to work on the modelling of biological networks, we present the most relevant ones, particularly the simulation and model-checking tools used for the verification of models. At the end, we describe some relevant data exchange formats used in those tools.

The third chapter introduces the constraint programming (CP) and constraint logic programming (CLP) paradigm, and the techniques used for the solving of constraint satisfaction problems (CSPs) in various fields, such as civil or mechanical engineering, air traffic control and finance. We show how it can be used to bring satisfiable answers to real world problems, as well as biological problems. We present its advantages compared to other programming paradigms, especially for the modelling of particular classes of problems. The main notions of CLP and the main techniques used by constraint solvers are detailed. Additionally, those notions are illustrated with the solving of CSPs. Then we present the Prolog programming language and some of its implementations dedicated to constraint solving, for instance the SWI-Prolog and its CLP(FD) library, and we take the opportunity to illustrate an example of CSP written in Prolog with the CLP(FD) library. Finally, we explain the strengths of constraint logic programming languages

in the context of the modelling of GRN, where their capabilities is a real advantage, especially for solving problems with uncertain or unknown parameters, for instance the study of the regulation phenomenons and interactions.

The last chapter presents BioNet, a support tool designed with the purpose of solving issues related to the study and modelling of GRNs by using constraint solving techniques. We describe the user requirements of a system with the aiming of helping the biologist researcher at his work on GRNs. We present the model and user interface used in BioNet, designed on the basis of the described user requirements. Its interesting features as well as the possibilities offered by its parametrization are detailed. Similarly, the use of BioNet is illustrated with the case of modelling the gene regulation in the *lac* operon, where we take the opportunity to illustrate a particular scenario. As well, we criticize the results obtained with this particular scenario. Then, the architecture of BioNet is described, including the tools and technologies used for its building. Finally, we present the limits of the tool and study some perspectives for the further improvements of BioNet.

We refer the interested reader to some additional reviews on the existing formalisms, such as [6, 7]. We also invite him to learn more about constraint logic programming, a paradigm described by Jaffar and Lassez in [8, 9]. For instance, Kimbal Marriott gives in [10] a good introduction to CP techniques used for the modelling of CSP. The most popular logic programming language used to illustrate those notions is Prolog, which has been used to design BioNet, via the SWI-Prolog implementation. In addition to SWI-Prolog, there exist numerous implementations of the Prolog language, not covered in this thesis, consequently we invite the interested reader to learn more about this interesting language, and particularly the extensions with capabilities to solve CSPs.

Finally, the Prolog source code of BioNet is completely available in appendix A, while additional parts of the implementation of server and user interface are shown to illustrate some technologies used in the BioNet architecture.

Chapter 2

Genetics and Gene regulatory networks

2.1 Introduction

Genetics is at its core the study of biological information with the emphasis set on heredity, genes and genes variation in all living organisms, from bacteria to multicellular animals and plants, which use large quantities of information in order to develop themselves, reproduce themselves and survive in their environments. Understanding how the information of genes is inherited from a parent organism to his descendants, what are the function and behaviour of genes as well as understanding how the organisms use this information of genes during their lifetime has become a field of active research. This chapter is an introduction to what is part of gene regulation networks, starting with elucidation of basic elements of living organisms, describing then structure and function of DNA, RNA and finally introducing mechanisms of gene expression and composition of gene regulatory networks.

2.2 From atoms to complex organisms

As illustrated in Figure 2.1 page 7, life is organized from simple atoms to complex organisms, where atoms are the most fundamental unit of matter which join together into clusters called molecules. Larger molecules are called macromolecules or supermolecules, and biology refers to macromolecules as the four types of molecules comprising any living thing, for instance these four types are lipids, proteins, nucleic acids and carbohydrates. Macromolecules are made of long chains of individual molecules linked together. In

the case of nucleic acids, these chained molecules are sort of building blocks of DNA, commonly called nucleotides. In the case of proteins, these molecules are amino acids (AAs for short).

Biological molecules in turn assemble into tiny structures called organelles, which are specialized subunits having a specific function in a cell. The cell is the basic unit of structure and function of living things, and some organisms are composed of a single cell, like bacteria. Many cells play only a specific role in an organism. Examples are blood cells or nerve cells. The living beings in turn are divided in two great families, revealing the existence of two levels of cellular organization.

Prokaryotes *prokaryotes* are single-cell organisms characterized by the lack of a nucleus and membrane-bound organelles. They are small and reproduce asexually, by binary fission or budding, and are present in nearly all environments, even under hard conditions such as extreme temperatures. Examples of prokaryotes are bacteria and archaea.

Eukaryotes *eukaryotes* are usually multicellular organisms, with a true nucleus and organelles within membranes. They are larger than prokaryotic cells. Cell division in eukaryotes can be either sexual or asexual, and involve processes of meiosis or mitosis. In meiosis, the produced cell is the result of the recombination of two parental chromosomes. In mitosis, a cell divides to produce two identical cells. In comparison, with prokaryotic cells who where the first to arise in the process of biological evolution, eukaryotes are much larger and complex. Examples of eukaryotes are animals, plants and fungi.

Cells in complex multicellular organisms are in turn organized in three levels : the most basic level is that of tissues which are groups of similar cells that act as functional units. Examples are muscle or root hair tissue. Tissues in turn join together into organs, body structures composed of different tissues joined in a structural and functional unit that plays a particular role. Examples are brain, skin, stomach and heart. The last level is when two or more organs function together to form an organ system, alternatively called biological system or body system. Examples are the digestive, immune, nervous and reproductive systems.

Finally, organ systems assemble into organisms, which are continuous living systems. An organism can also be comprised of only a single cell such as bacteria, and may be either a prokaryote or an eukaryote. Examples of organisms are human, bacteria and plant.

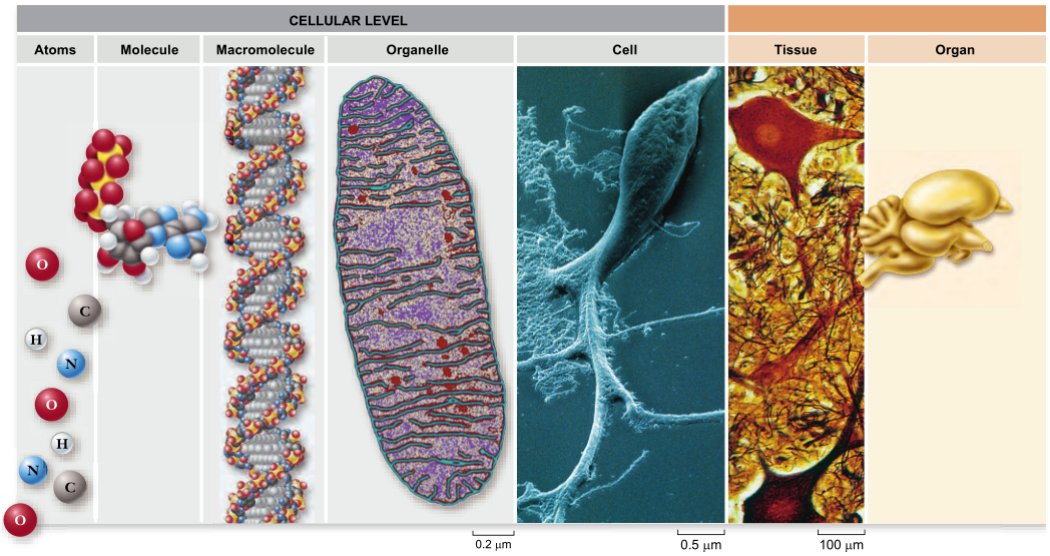

FIGURE 2.1: Life organization from atoms to organisms - From Raven et al. [11][p. 2].

2.3 DNAs and RNAs

2.3.1 Role

Deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) are nucleic acids made of long chains of individual molecules linked together, which is also the case for other macromolecules present in the organism. According to what Francis Crick suggested in 1956, the function of DNA is to *store* the *genetic information* (or *genetic material*) in its sequence of nucleotide bases. Then the DNA pass it (during the transcription process) to RNA which has the function of reading, decoding and use this information. This is done via a translation process, in which the information from the sequence of nucleotides is translated into a sequence of amino acids (AAs). Then these AAs are joined together to make a full protein chain. This is also called *protein synthesis* and is commonly referred to as the *central dogma* of molecular biology, as illustrated in Figure 2.2 page 8. In addition to its previously described role, DNA is also involved in the replication process. In this process, a copy of DNA is made during the cell-division cycle, a vital process enabling cells of the organism, like hair, skin and blood cells, to be renewed. Concretely, cells are renewed by division into new daughter cells, as previously described as meiosis or mitosis in section 2.2 page 5. RNA plays an important role in protein synthesis. More specifically, three different types of RNA are involved in cooperation to ribosomes. All of them being described on page 10, we first detail the structure of RNA and DNA.

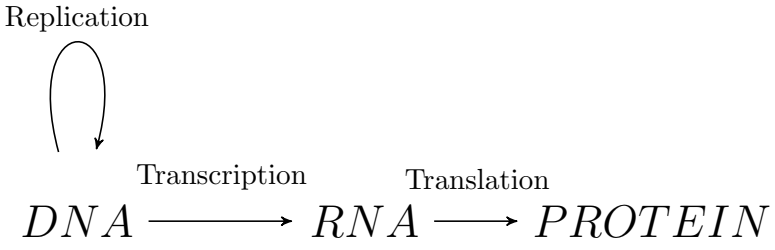

FIGURE 2.2: The central dogma in molecular biology

2.3.2 Structure

DNA and RNA are similar from the structural point of view. A DNA sequence is made up of two strands which are complementary (see Figure 2.3 page 8) to one another and assembled in a double helical structure, as stated by James D. Watson and Francis Crick by 1953. At the opposite, a RNA sequence is single-stranded. Each single strand of DNA is made of a linear sequence of – possibly hundreds of millions of – nucleotides. Four different types of nucleotides are found in DNA. They are represented by simple letters : adenine (A), thymine (T), guanine (G) and cytosine (C). RNA is also made of four types of nucleotides where the thymine (T) is replaced by uracil (U). These symbols (nucleotides and their letters) belong to a sort of alphabet which is called the genetic code. The complementary property of the two nucleotide strands means that each A, C, G, T in one strand is paired with a T, G, C or A respectively in the other strand, as illustrated in Figure 2.4 page 8. From another point of view, if we know the sequence of a strand, we can deduce the sequence of the other one. This complementary property is crucial in the process of DNA synthesis. As a result, the pairing aspect introduces a redundancy property and allows the cell to rebuild the entire genome, on the basis of a single strand.

A strand of DNA : `...T-A-T-G-C-A-G ...`
Its complement : `...A-T-A-C-G-T-C ...`

FIGURE 2.3: Example of two complementary DNA strands

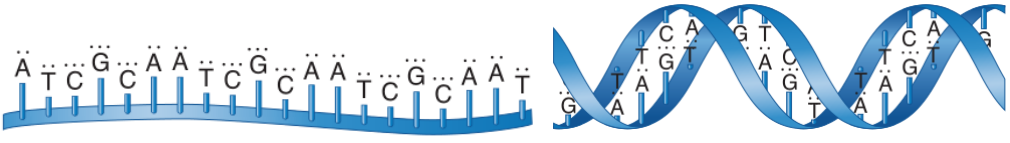

FIGURE 2.4: Single-stranded (left) and double stranded (right) sequences - From [12][p. 231].

The length of a DNA sequence (i.e. sequence of nucleotides) is measured in terms of *bases* (b) in single stranded DNA and, in the case of double stranded DNAs, in terms of pairs of nucleotides or base pairs (bp). 1,000 base pairs (1,000 bp or 1 Kb) equal

a *kilobase* and 1,000 Kb compose a *megabase* (1,000,000 bp or 1 Mb). The sizes of genomes of different organisms range from 580 Kb for the smallest bacterial genome to megabases, and for instance the human genome is composed of about 3,000 Mb. The large size of some genomes is not generally due to a great number of genes but especially to a great amount of repetitive elements in the DNA. One estimates that only 5% of the human genome is functional (codes for protein), while at least 50% is formed by repetitive elements.

2.4 Proteins

Proteins can serve various roles related to the development and function of an organism : they allow building the structures and to perform the metabolic reactions necessary for life. In addition, they participate in the regulation as *transcription factors* (see section 2.11.1 page20). As described in subsection 2.3.1 page 7, proteins are macromolecules made by chaining together simpler molecules known as amino acids or AAs. A typical protein contains 200-300 AAs but there exist proteins with less than 30-40 AAs as well as proteins having up to ten thousands AAs.

The function of the protein into a living cell or organism is determined by the exact types and order of chained amino acids which compose it, and this order determines also the three dimensional structure of the protein [13, p. 3]. Small errors in a amino acid (AA) chain can result in different shape and alter protein performance while major errors can annihilate its function. Only 20 distinct amino acids types are involved in the composition of proteins, as shown in Table 2.1 page 10. As a consequence if a protein is formed of 400 chained amino acids, 20^{400} different proteins can be made. This is actually more than the number of existing proteins on the earth. Once an amino acid sequence is known, it can be stored in shared sequence databases to help to learn more information on the protein function or predict the protein 3D structure from the amino acid sequence.

For example, a complete amino acid sequence shown in Figure 2.5 page 10 is that of iron/manganese *Superoxide dismutase* from the organism *Haemophilus influenzae*. It is a powerful antioxidant produced by the cells in the living organisms. Its function is to destroy superoxide anion radicals which are created within the cells and are toxic for the system. There exist 4 variants depending on the metal contained in the molecule : iron, manganese, copper or zinc.

As described sooner in section 2.4 page 9, the amino acid sequence determines the 3D structure of the protein, and this structure plays an important role in the function of

A Alanine	L Leucine
R Arginine	K Lysine
N Asparagine	M Methionine
D Aspartic acid	F Phenylalanine
C Cysteine	P Proline
Q Glutamine	S Serine
E Glutamic acid	T Threonine
G Glycine	W Tryptophan
H Histidine	Y Tyrosine
I Isoleucine	V Valine

TABLE 2.1: List of the 20 amino acids with their symbols

```
MSYTLPELGYAYNALEPHFDAQTMEIHHSKHHQAYVNNANAALEGLPAEL
VEMYPGHLISNLDKIPAEKRGALRNNAGGHTNHSLFWKSLKKGTTLQGAL
KDAIERDFGSVDAFKAEFEKAAATRFGSGWAWLVLTAEGKLAVVSTANQD
NPLMGKEVAGCEGFPLLGLDVWEHAYYLKFQNRRPDYIKEFWNVVNWDFV
AERFEQKTAHSNCAK
```

FIGURE 2.5: Example of sequence of 215 amino acids is that of iron/manganese *Superoxide dismutase*. From `http://www.uniprot.org/uniprot/P43725`.

the protein. In Figure 2.6 page 10 is shown the 3D structure of the superoxide dismutase whose the amino acid sequence has been given.

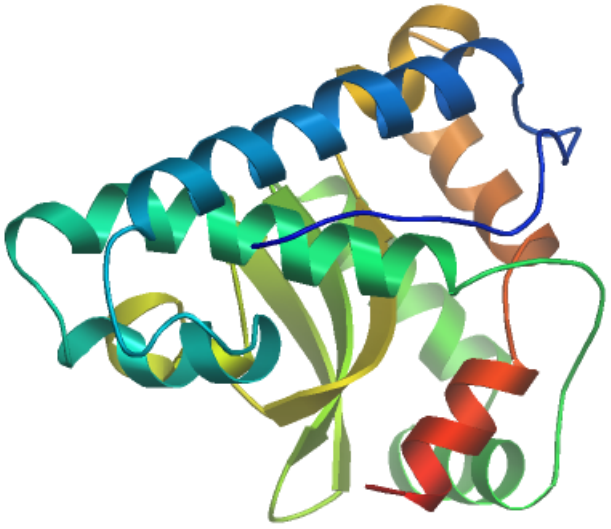

FIGURE 2.6: Superoxide dismutase 3D Structure - From `http://swissmodel.expasy.org/repository/smr.php?sptr_ac=P43725&csm=80FAC9F1C0D59D25`

2.5 Ribosomes and the genetic code

The ribosomes are the cellular "machines" driving protein synthesis by taking a chain of nucleotides (RNA) and translating it into a chain of amino acids (or polypeptide)

which later folds into a protein. Since the ribosome cannot directly work from the double stranded DNA sequence (found in chromosomes), the cell needs to create a copy of the DNA with which the ribosome can work. This is called RNA. Different types of RNA are involved : first the DNA encoding genes are transcribed by the cell into messenger RNA (mRNA), before any protein is made. In turn the mRNA carries the genetic information from DNA to the cell's ribosomes which receives from the mRNA molecule the instructions for protein synthesis. The mRNA sequence is a copy of the DNA sequence, with one-to-one matching, except that RNA uses the nucleotide uracil (U) in place of thymine (T) as seen in subsection 2.3.2 page 8. Last RNA types are the transfer RNA (tRNA) that carry appropriate amino acids into the ribosome for inclusion in the new protein, and ribosomal RNA (rRNA) which are the main molecules composing the ribosomes. Figure 2.7 page 11 illustrates the relation of ribosome and RNA from transcription to translation.

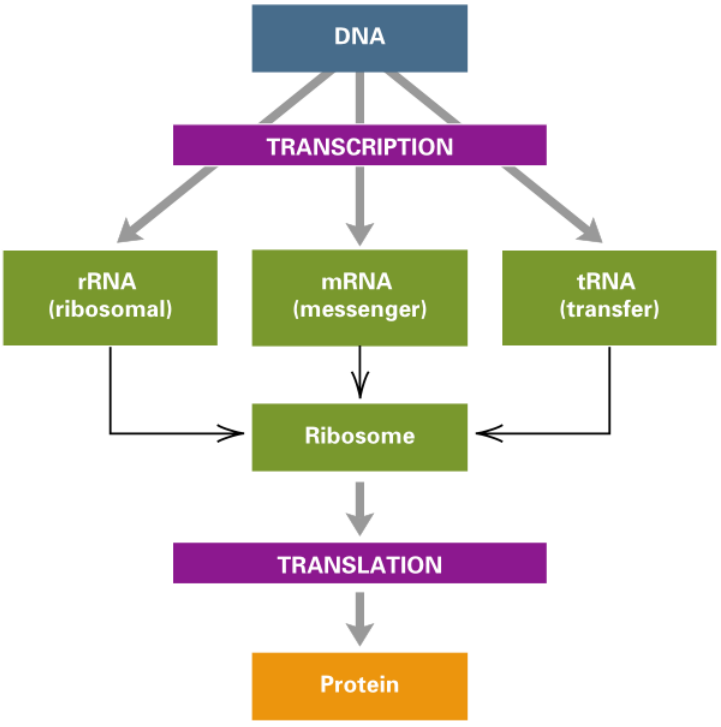

FIGURE 2.7: Roles of Ribosome and RNA as suggested by the central dogma - From [14][chap. 1][p. 17].

This translation depends on a dictionary known as the *genetic code* which was "cracked" in the 1960s by combined efforts of Marshall W. Nirenberg, Philip Leder and Har Gobind Khorana. It is a conversion table which is essentially the same for all living organisms, which describes how to translate a nucleotide sequence to AAs. However, it is not possible to build a one-to-one mapping between nucleotide and AA since RNA has only four distinct letter symbols (A C G U), while 20 different sorts of AAs are used in protein synthesis. The genetic code as illustrated in Figure 2.8 page 12

consists in a conversion table from codons (triplets of nucleotides) to the AAs within a polypeptide sequence.

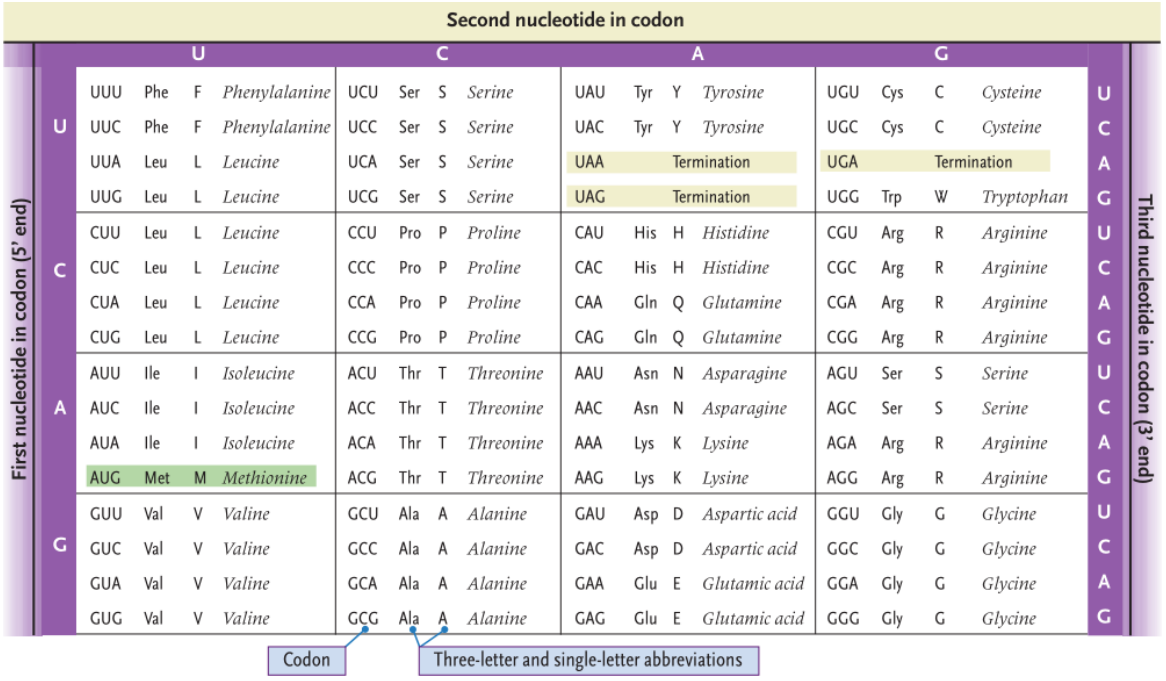

FIGURE 2.8: Genetic code - From [14]

Using the four different nucleotides, sets of 2 nucleotides could code $4^2 = 16$ amino acids, which is not sufficient to code all the AAs. Sets of 3 nucleotides could code $4^3 = 64$ amino acids, which is more than enough. Since there are only 20 AAs, the genetic code is redundant and as a consequence two distinct sets of three nucleotides can code the same AA. For example, both AAG and AAA code the amino acid Lysine. Finally, three of the codons are stop codons and mark the end of any particular amino acid chain. They are, for instance, UAA, UAG and UGA as shown in Figure 2.8 page 12. Additionally, Figure 2.9 page 13 states how sets of three nucleotides bases (also called triplets) are translated to AAs.

2.6 Genome, chromosomes and genes

The *genome* of an organism comprises all chromosomes and DNA sequences present in this organism. It is the provider of the complete hereditary information present in an organism and its individual cells. Functionally the genome is divided into genes while structurally it is divided into different DNA molecules or *chromosomes*. Chromosomes in turn are large structures within living cells, made of a long stretch (segment) of DNA that represents many genes which associate to DNA. They are responsible for storage, duplication, expression and evolution of DNA. Different organisms have different number

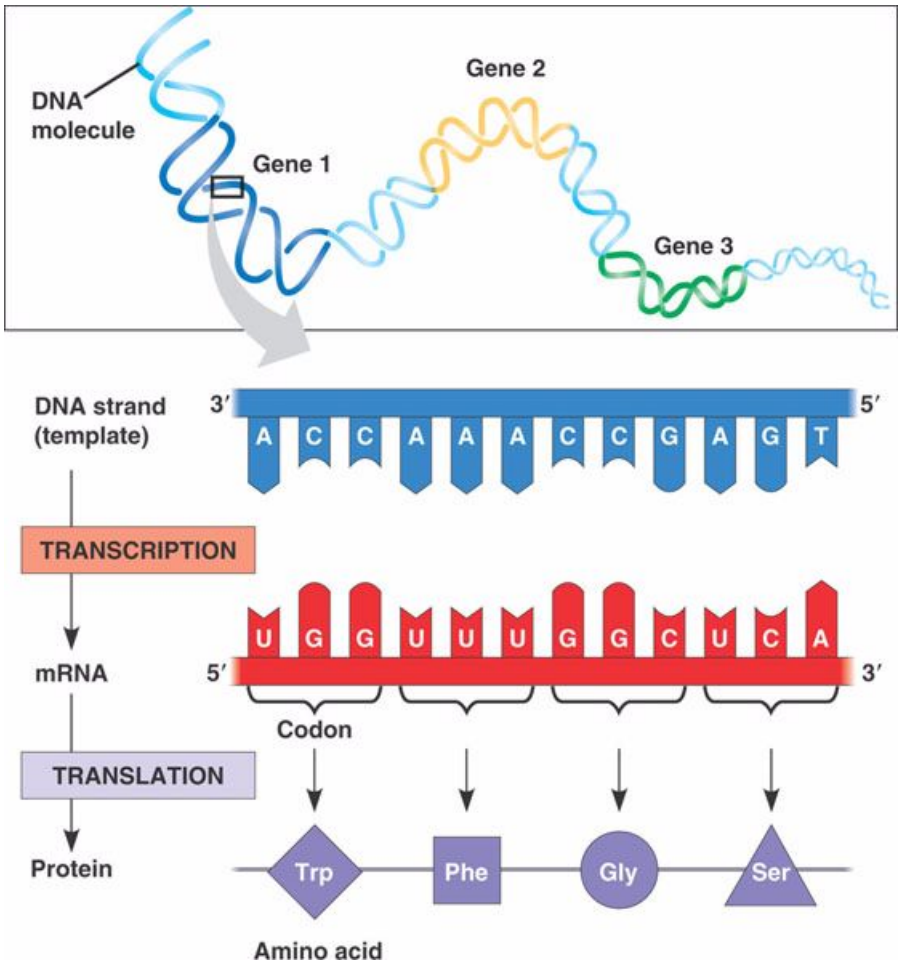

FIGURE 2.9: From nucleotides bases to amino acids [15]

of chromosomes and genes. For example it was discovered by 1956 that the human cells contain 23 pairs of chromosomes, for a total of 46. Usually (there are exceptions), the complexity of an organism is directly related to the number of chromosomes which compose it. Each chromosome found in the genome may contain numerous genes, but the number differs between eukaryotes and prokaryotes. Eukaryotes have usually more genes spread across many chromosomes while prokaryotes have fewer genes, located on a single chromosome. For example the genome of a type of bacteria such as the mycoplasma contains about 500 genes while the human genome may contain up to 20,000 to 25,000 genes [1] and the genome for rice may contain as many as 50,000 to 60,000. Gene and chromosome are illustrated in Figure 2.10 page 14.

[1] (20,000 - 30.000 according to [13, p. 3])

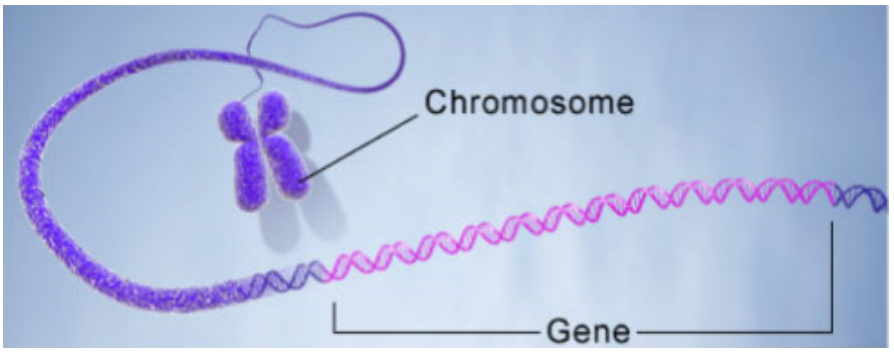

FIGURE 2.10: Gene and chromosome - From http://smart-therapeutics.com/Technology/RNAs-Role-in-Protein-Synthesi

2.7 Genes

In the 1860s Gregor Mendel, father of modern genetics, studied how *traits* are inherited between generations, and experimented from his studies on pea plants that organisms inherit traits by a "factor" which later would be referred to as a *gene*. His studies revealed the basic principles of inheritance, and have been later rediscovered in the 1900s and integrated in other works, as those of Thomas Hunt Morgan on the role of chromosomes in the inheritance mechanism.

A gene may be defined as a functional unit, structurally an individual segment of DNA, which contains the instructions to produce a functional product. In most genes, the functional product is a protein with a specific function, i.e., a *polypeptide* which is a linear sequence of amino acids (AAs) that folds into units that constitute proteins.

Similarities exist between the genes of different living organisms. For instance, [13, p. 5] reports that a gene with a specific role in an organism that would be placed in the genome of another organism would function as normally as in the original organism, even if these organisms were from different species.

Random changes that happen in genes are also known as "mutations" and may result in the creation of new *alleles* and new traits. Traits are the distinct characteristics of an organism which may have been inherited or determined by the environment. Examples of physical traits in the human being are eye colour, hair, texture, height as well as blood types or resistance to diseases and drugs. The information which may vary from an organism to another resides in particular genes and different copies of a gene do not necessarily give the same instructions (for coding proteins). Consequently, a gene may have multiple/alternative forms. Each one is called an allele (or allel), a name shortening the term *allelomorph* ("other form").

The study of genes and protein synthesis is at the heart of the study of gene regulation networks. The purpose of their study is to understand and elucidate the processes involved in the production of specific gene products in reaction to environmental stimuli, since those processes are known to be regulated and play a major role in resistance to disease as well as the evolution of organisms. Their understanding requires first a deep understanding of the interactions between genes and proteins. The following section 2.8 page 15, introduces to the DNA sequencing, which is required to the study of genes and proteins. Then, the focus will be set in the next sections on the gene expression (section 2.9 page 16) and gene regulation processes (section 2.11 page 18), which are at the heart of gene regulation networks.

2.8 DNA Sequencing

The discovery of the double helical structure of DNA in 1953 has paved the way of the development of DNA sequencing techniques. Initial DNA sequencing methods were published and developed in the 1970s. The first full DNA genome to be sequenced is that of the bacteriophage ϕX174 in 1977. DNA sequencing has many uses, such as determining the sequence of individual genes, operons, full chromosomes and entire genomes. In addition, it can be used for other tasks such as :

- identifying new genes and associations with diseases and potential drug targets.
- studying what and how proteins are made.
- understanding how different organisms evolve and how they are related.
- identifying species present in a body, water, dirt, debris filtered from the air or organisms.
- detecting the presence of known genes for medical purposes and parental/heredity testing.

The frequent progress in the development of new DNA sequencing techniques as well as the quick evolution of computers have had a huge impact on the understanding of genetics. They have allowed scientists to exchange large amounts of information and studies on genetics, which in turn has constantly increased the progress in this field. In addition, the exchange of sequenced data has made possible synthesizing an exact copy of all or a portion of a DNA molecule from another place in the world [chap. 1.2 13, p. 2]. An historical example is that of Human Genome Project (HGP), an international scientific research project, publicly funded, which proceeded to completely sequence the human genome and which is still known as the largest collaborative biological project in the world. It started in the 1990s to achieve his quest by 2003 with the final sequencing mapping of the human genome. It contributed in addition to the sequencing

of bacterium *Escherichia coli (shortened E.coli)* as well as different plants and animals. It has been found that the human genome contains 20,000-30,000 genes. In comparison, other organisms contain about to 500 genes for bacteria Mycoplasma genitalium and 14,000-19,000 for nematodes (roundworms) and fruit flies. [chap. 6.6 16, p. 122]. Examples of benefits due to the HGP include a better understanding of diseases in particular different forms of cancer and a better design of medications, in addition to a deeper understanding of their effects. [13, p. 8] [17][p. 6] [18].

2.9 Gene expression

Remember the central dogma (see Figure 2.2 page 8). It states how the genetic information flows from DNA via RNA in order to synthesize proteins with precise role and structure. However, the protein synthesis does not happen regularly in cells but occurs only when genes are activated by some stimuli. It belongs to a more general multi-step process called *gene expression* that proceeds from transcription to translation. It plays a role in driving the activity of proteins, by controlling the level of genes transcription or the number of mRNA transcripts available for translation. Gene expression may result in a product (also called a *gene product*), either RNA or a protein. This product affects the characteristics of cells and organisms in addition to expression of other genes [19]. Gene expression levels can be altered by cells in response to different conditions, as the environmental stresses/stimuli. This is part of the *gene regulation* process (covered in section 2.11 page 18). Finally, details of gene expression mechanisms may vary between prokaryotes and eukaryotes. For example eukaryotic genes can be divided in nucleotide sequences called *exons* and *introns* (sometimes also referred to as *intervening sequences*) at the opposite of prokaryotes. Exons are parts that are expressed through transcription and translation, as RNA products. They are *coding regions*, i.e., they specify (or "code") for proteins. At the opposite introns are not expressed but rather removed by *RNA splicing* before the final RNA product is made, and they are usually not considered as non-coding regions since there be not evidence they code for proteins or enzymes. Finally, exons are joined together as parts of the mature mRNA product whereas introns are absent, as illustrated in Figure 2.11 page 17. The role of introns is to separate coding regions (extrons) of the gene and allow different combinations of extrons to be joined together. As a consequence, a gene can code more than one protein. This principle is also known as *alternative splicing*. At the opposite prokaryotic genes splice rarely and mainly non coding RNAs. Another difference is that of transcription and translation processes which occur in the same cellular compartments in eukaryotes, where transcription is often coupled with translation. At the opposite of prokaryotes where transcription and translation occur respectively in the nucleus and the cytoplasm,

as a consequence the transcription and translation in prokaryotes are separated. Other differences exist but are not covered in this thesis. The interested reader can learn more about them from [13][p. 272].

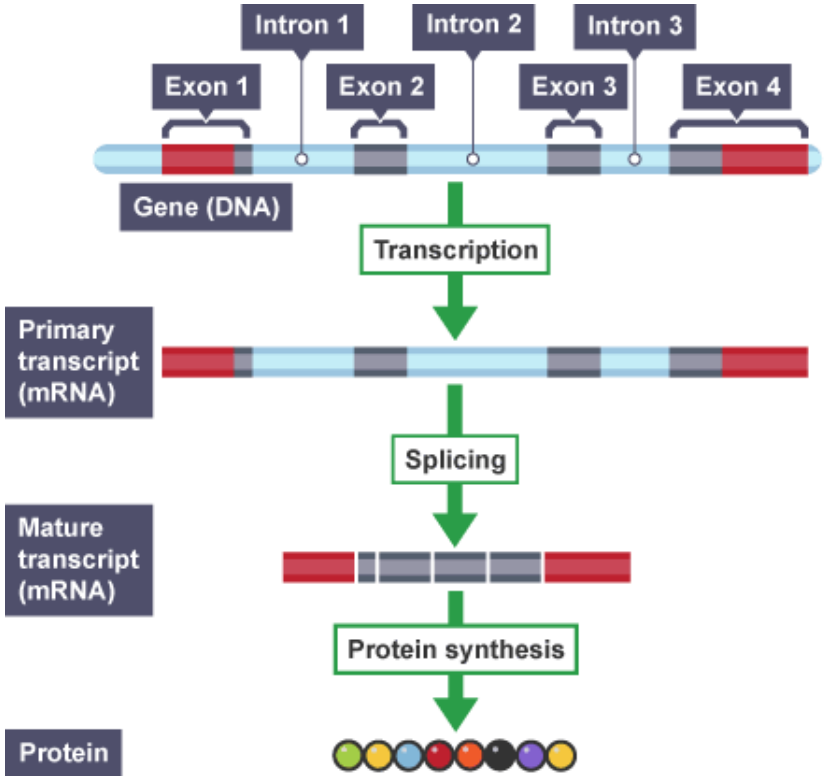

FIGURE 2.11: The process of protein synthesis - From http://www.bbc.co.uk/education/guides/zgrccdm/revision/2

2.10 Data and analysis of Gene expression

A major step in measuring gene expression has been the introduction of DNA microarrays, also called DNA chips. They are formed of microscopic DNA spots and are used to measure expression levels of mRNA transcripts, helping to determine which genes are activated or repressed in different cells under specific conditions and at different times. An example of microchip is illustrated in Figure 2.12 page 18.

The DNA microarrays are capable of measuring the difference in expression of large number of genes, even the complete genome of an organism, at the same time. As a consequence it is possible to achieve a better understanding with the comparison of multiple experiments, and by comparing differences between observations and predictions. Analysing the expression levels help scientists to identify the relations between genes at a functional level. For example genes having similar expression profiles often share common functions. Gene expression data are usually expressed by a matrix of continuous

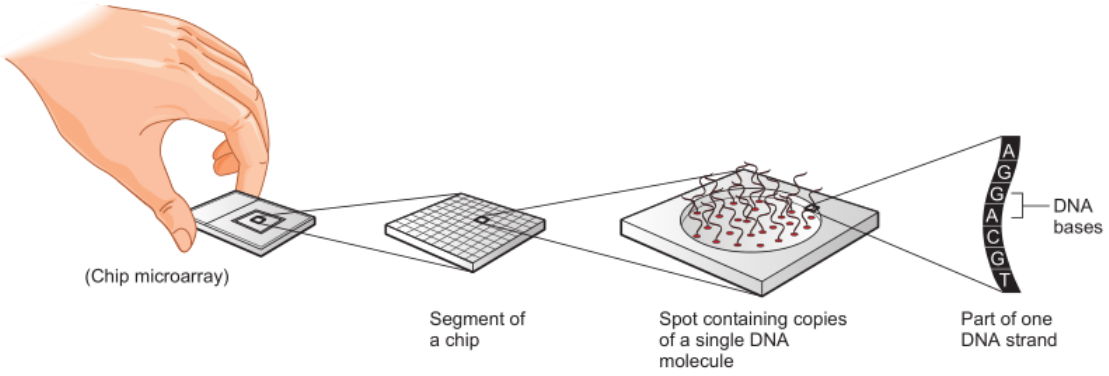

FIGURE 2.12: Schematic example of microchip - From Hartwell et al. [13][p. 9].

expression levels where rows are genes and columns are the different experiments ordered chronologically starting from left. An example of gene expression matrix is illustrated in Figure 2.2 page 18.

	Exp_1	Exp_2	Exp_3	...	Exp_m
$Gene_1$	0.2	0.3	0.42	...	0.6
$Gene_2$	1.4	1.2	0.98	...	
$Gene_3$	-0.1	0.4	-1.0	...	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	
$Gene_n$					

TABLE 2.2: Example of gene expression matrix (Inspired by [7])

Gene expression data are used in several methods related to the identification of gene networks [7], some examples are given in the Chapter 3, section 3.3 page 36.

2.11 Gene regulation

The gene regulation, also referred to as regulation of gene expression, is "the phenomenon in which levels of gene expression can vary under certain conditions" [12, p. G-7]. For instance the environment changes with regard to temperature and available nutrients, an example being the control of insulin expression that regulates the levels of glucose in blood. The phenomenon of gene regulation refers to control of some processes at the cellular level. It is essential for viruses, eukaryotes and prokaryotes and ensures the adaptability of an organism to its environment, as well as the cellular differentiation, by which a cell can turn in a more specialized type and differentiate from the other similar or parental cells. Examples of such processes are cell division, and production of proteins in response to certain environmental stimuli/stresses. It also includes several mechanisms used by cells to increase or decrease the production of gene products.

The main part of gene regulation occurs during the DNA-RNA transcription steps referred as *transcriptional* regulation. Gene expression is also regulated in other stages as *post-transcriptional*, *translational* and *post-translational* regulation. Among those most well known and understood are transcriptional and translational regulation. They are relatively slow processes that could take minutes or even hours to proceed, whereas post-translational regulation takes only a few seconds to take effect. Regarding their function, transcriptional regulation refers to the control of the rate of transcription. It controls how many copies of RNA are transcribed and executes a temporal control over the genes transcription so that it occurs only at proper times and in proper amounts. Translational regulation refers to the control of the levels of protein synthesized from its mRNA. Finally, post-translational regulation refers to the functional control of proteins that are already present in the cell. The transcriptional regulation process is described further in subsection 2.11.1 page 20.

Most of the cells in a multicellular organism contain the same genetic material but may look different because of gene regulation. The genes expressed in one nerve cell for instance are not expressed in the muscle cells, and vice versa. Genes can be classed in different groups depending on whether they are regulated or not. Some examples are :

Constitutive genes Most of the genes are usually regulated but exceptions exist like unregulated genes which are part of the class of *constitutive* genes – they encode proteins which are permanently needed for the survival of the bacterium, and consequently does not need regulation.

Facultative genes At the opposite of constitutive genes are *facultative* genes. They are transcribed only when needed, which means they are regulated in response to environmental stimuli/stresses so that the cells can synthesize only required proteins, at proper times and in proper amounts. This is a key benefit of gene regulation, since it avoids the waste of a valuable energy.

Housekeeping genes *Housekeeping* genes are constitutive genes that code for proteins constantly required by the cell. As a consequence, they are essential and transcribed at a constant level under any conditions. Example is *Glyceraldehyde-3-phosphate dehydrogenase* (GAPDH).

Inducible genes *Inducible* genes are genes whose expression can be either dependent on environmental stresses or time-dependent when it does not occur any time in the complete cell cycle.

As with gene expression, differences can be observed between gene regulation in eukaryotes and prokaryotes even if in both cases cells need to adapt to changes in their

environment, an example being the ability of humans to develop a tan in order to protect the cells of the skin against damages caused by UV rays. Finally, some genes are expressed during the development stage while others are expressed when the organism is ageing to adult [7][chap. 27.2].

The following sections detail first the process of transcriptional regulation which is one of the most well known and understood regulation process, and then the function of gene regulatory network (GRN) to study the machinery joining the elements involved in gene regulation.

The following subsection 2.11.2 (page 20) and subsection 2.11.1 (page 21) introduce the machinery joining the elements involved in gene regulation. This machinery is commonly referred to as gene regulatory networks or transcriptional regulation networks. Finally, the subsection 2.11.3 (page 24) gives an example of regulation for the case of the *lac* operon.

2.11.1 Transcriptional regulation

The transcription of genes depends on transcription factor (TFs) called either *activators* or *repressors*. They are regulatory proteins present in a gene, working alone or in cooperation with other proteins, that exert on transcription process either positively or negatively depending on the type of TF. Those positive or negative controls are also referred to as respectively *up-regulation* and *down-regulation*. Down-regulation is the process by which a cell decreases the amount of a cellular component, such as RNAs or protein, in response to environmental stimuli. At the opposite up-regulation is the process by which this quantity is increased by the cell. In order to exert their positive or negative control, TFs bind to specific regions of DNA adjacent to the genes they regulate. Examples of such regions are promoters or enhancers.

Promoter A promoter is a particular region of DNA that initiates transcription of a particular gene.

Enhancer An enhancer is a particular DNA sequence that can be bound to activators to increase the level of transcription of a gene or multiple genes by activating the promoter region. It functions as a “turn on” switch of gene expression.

By binding to *promoter*, a repressor *inhibits* transcription while an activator *increases* the rate of transcription, thereby having respectively a positive or negative impact on the rate of protein synthesis. Only some of all genes act as activators or inhibitors. Their identification is an important and complex task. Inhibitors are sort of biochemical signals

which prevent the expression of a particular gene even in the presence of an appropriate activator. The functions of regulatory proteins inhibitors and activators are controlled by small *effector modules* that either cause the transcription to increase or decrease. Examples are the *inducer* that increases transcription, *inhibitor* that prevents activator from binding to DNA and *corepressor* that binds to a repressor protein. Both inhibitor and corepressor decrease the rate of transcription. Other proteins which also participate in gene regulation without binding to DNA are excluded from the class of transcription factors. Example is the *coactivator* which is a protein that plays a role in the activation of transcription.

Other notions are involved in gene regulation. Among those are the following :

Operon The *operon* (from the French *opérer*, which means to operate) is a functional unit of DNA composed of two or more genes (also referred to as a cluster of genes) and is under transcriptional control of a single promoter. All genes which form this operon are either transcribed together or not at all. In subsection 2.11.3 page 24, an example of operon is described.

Silencer At the opposite of the enhancer, the *silencer* functions as a "turn off" switch that decreases the level of transcription.

Insulator The *insulator* is a segment of DNA that blocks the interaction between an enhancer and a promoter, and consequently determines the set of genes on which an enhancer has effect.

Operator A segment of DNA to which a transcription factor binds to regulate gene expression. Usually, this transcription factor is a repressor that binds to the operator to prevent transcription. An example of operator is described further in subsection 2.11.3 page 24 with the example of regulation in the *lac* operon.

2.11.2 Gene regulatory networks

A gene regulatory network (GRN) is the machinery joining the elements involved in gene regulation. It is made of the DNA segments which interact with each other and control the gene expression levels of mRNA and proteins. The structure of GRN has an effect on how an organism, even as simple as a bacterium, can adapt to a frequently changing environment (varying temperature, nutrients, and other factors) anywhere on the earth. The ability to control the response to environmental changes is the main function of regulatory networks in a bacterium. But all living organisms have, as a key property, the ability to grow by using sources of energy optimally. With the ability to

influence how genes behave, GRNs enable complex and rapid changes in the process of organisms evolution. An example of some change illustrated by Figure 2.13 page 22 is the way the two-winged fly evolved from a four-winged ancestor, which was caused the rewiring of the GRN.

FIGURE 2.13: Pictures of two-winged fly (left) and four-winged fly (right) - From [13][p. 7].

In appearance a GRN can be very complex and interactions can occur at different levels. It includes that metabolites interactions, at the lowest level, the proteins interactions at a higher level and finally genes interactions at highest level. In addition, complexity is added with possible interactions between the different components of those levels. The set of possible interactions between components is illustrated in Figure 2.14 page 22.

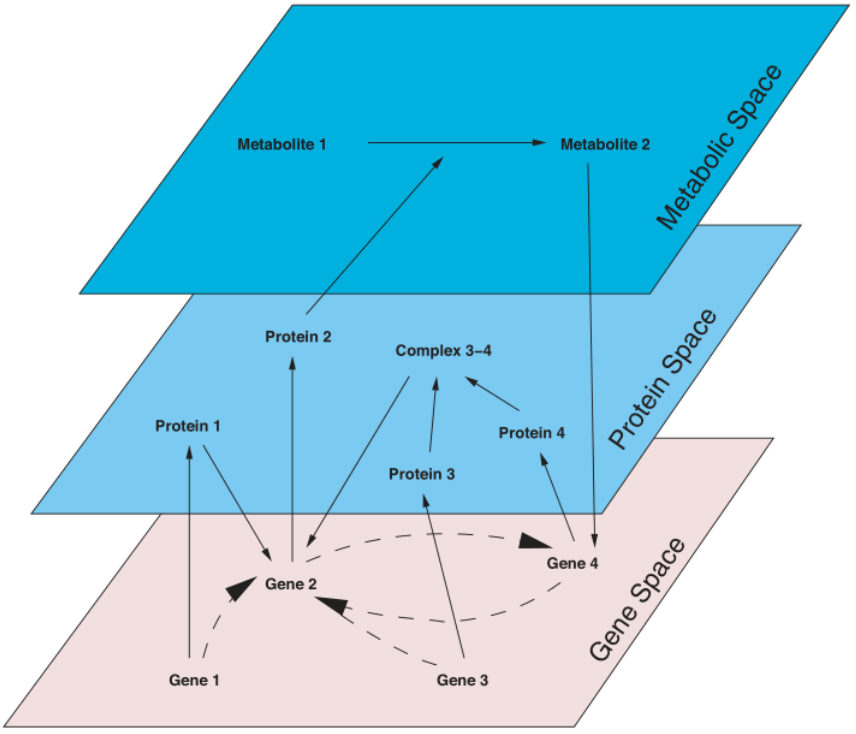

FIGURE 2.14: Examples of GRNs : Overall of interactions in gene regulatory networks - From [7]

A concrete example is illustrated in Figure 2.15 page 23 where protein-protein interactions in yeast are shown.

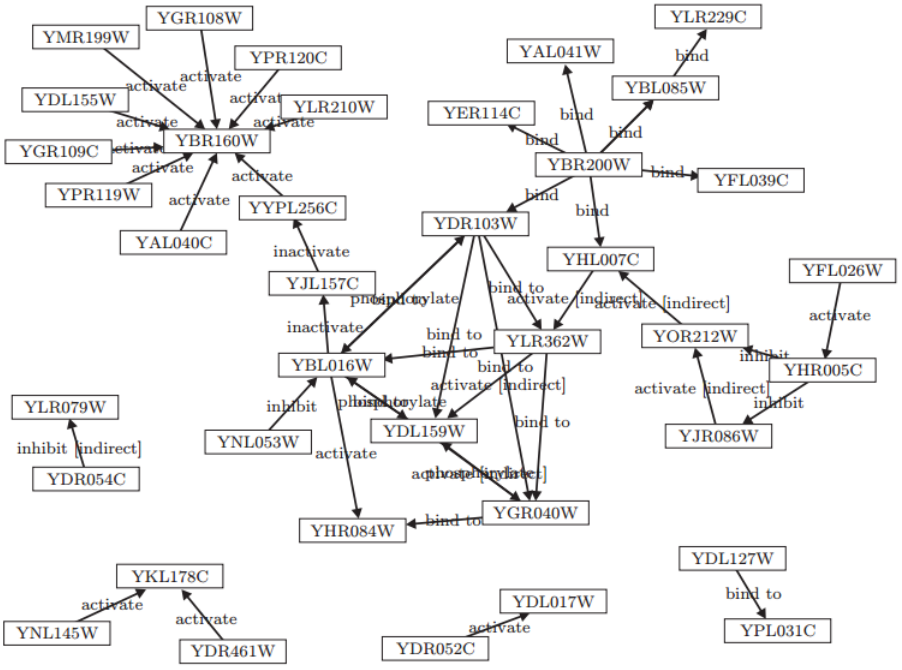

FIGURE 2.15: Interactions in yeast (especially *Saccharomyces cerevisiae*) - From [20]

Finally, Figure 2.16 page 23 illustrates how relatively complex a complete network can be, with the example of the interactions network of genes related to invasivity. There exist even more complex networks.

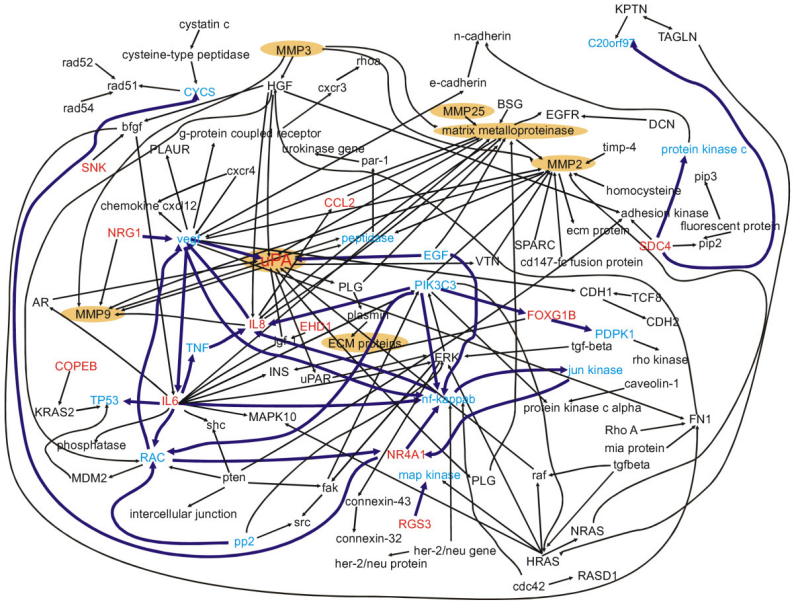

FIGURE 2.16: Examples of GRNs : The nVasion-network [21]

In addition to gene regulatory networks, biologists also actively study other biological gene networks. They include the followings :

Protein−protein interaction networks a protein−protein interaction network (PPI) comprises the physical interactions between two or more proteins as a result of biochemical events and/or electrostatic forces. The PPIs are regulated by multiple stimuli. Examples are the presence and/or concentration of other proteins and nucleic acids.

Metabolic networks a metabolic network comprises the biochemical reactions in a cell. They consist of metabolic pathways, and the regulation processes that control these reactions. Metabolic pathways include the chemical reactions that keep the living system in *homeostasis*, i.e., in a stable state. The components of a metabolic networks are the enzymes that catalyse (accelerate) the chemical reactions and substrates, while the relations between those components are metabolic interactions.

Cell signalling networks a cell signalling network is a relatively complex system governing the basic functions of the system, such as the development, immunity and repair activities. Those networks consist in biochemical reactions in a cell, where particular proteins known as receptors react to external stimuli, such as presence of particular hormones in the organism. Practically, the cells increase or decrease their sensitivity to particular hormones by increasing or decreasing the number of receptors at the cell's surface. At the opposite, hormones themselves can cause the cell to down-regulate or up-regulate. The components of these networks are the proteins.

2.11.3 Gene regulation in the *lac* operon

In the 1960s, François Jacob and Jacques Monod discovered the *lac* operon (lactose operon) in *E. coli* bacterium. This bacterium inhabits the intestinal tracts of many animals, including mammals. The diet of E. coli depends upon what the host animal is eating. Even if its preferred food is glucose, other foods can be used, but glucose will be preferred.

The *lac* operon is designed in E. coli for the digestion of lactose. Structurally, the regulatory system of *lac* operon consists of a promoter, an operator (which has the role to turn on or off the operon) in the regulatory region, a repressor lacI, and last three different and adjacent structural genes, namely *lacZ*, *lacY* and *lacA*, which code for the proteins involved in the digestion of disaccharide lactose. For instance, the

lacZ encodes the β-galactosidase (LacZ), and the *lacY* gene encodes the β-galactoside permease (LacY) which enables the entry of lactose from the medium into the cell. Both are enzymes necessary for the uptake and utilization of lactose. Finally, the *lacA* gene encodes the thiogalactoside (LacA), but its role is still uncertain.

The structure of the *lac* operon is illustrated in Figure 2.17 page 25.

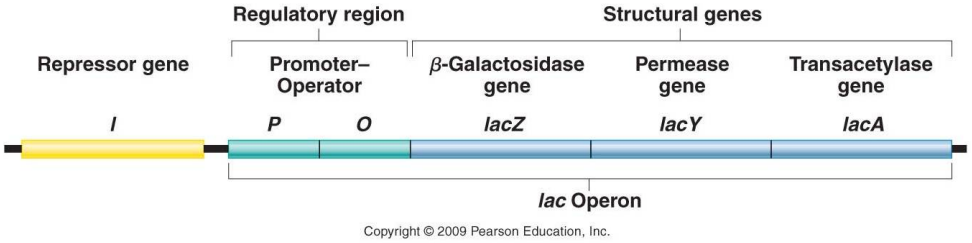

FIGURE 2.17: Structure of the *lac* operon

In the details, the three structural genes of *lac* operon are transcribed as a single mRNA in reaction to glucose/lactose ratio and this mRNA is translated into three proteins, which are required to digest of disaccharide lactose.

Gene regulation in the *lac* operon was the first GRN completely understood. It comprises numerous mechanisms, related to presence/absence of lactose and glucose. First, we describe the mechanisms related to the lactose :

1. The structural genes of the *lac* operon are regulated by a repressor, the protein LacI, when it is activated. The protein LacI is encoded by the regulatory gene *lacI*.

2. In the absence of lactose, a by-product of the lactose metabolism, the allolactose, binds to the *lac* repressor and activates it. In turn, the repressor inhibits the action of the operon. Practically, the operator is switched to off by the repressor which is physically fixed to it. Consequently, this binding stops the transcription of genes into mRNA. It is an advantage since the cell avoids to waste energy in the production of enzymes for the lactose metabolism.

3. At the opposite, in the presence of lactose, the repressor is inactivated, consequently the operator is unblocked and the genes of the operon are able to code again for enzymes.

This sort of regulation is referred to as inducible regulation, because the genes of operon are turned off by a regulatory factor until it interacts with an environmental stimulus that stops the repression.

The second mechanism involved is related to the presence/absence of glucose. It can be described as follows :

1. In the presence of glucose, even when both glucose and lactose are present, the bacterium will preferably use glucose as the carbon source to produce energy. Consequently, the presence of lactose is not sufficient for induction of the *lac* operon and the use of lactose is prevented until the concentration of glucose is very low.

2. In addition, the catabolite activator protein (CAP), which is regulated by cyclic adenosine monophosphate, or cyclic AMP (cAMP), plays also an important role. The concentration of cAMP decreases when glucose is used as carbon source. But when glucose is not available, the concentration of cAMP increases, and this change of concentration level serves as a signal for the bacterium that glucose is absent. Consequently, the bacterium switches to lactose metabolism. It is done by the action of binding of cAMP to CAP, which form a cAMP-CAP complex. Then, when this cAMP-CAP complex binds the promoter region, it initiates the transcription of the *lac* structural genes. Without the binding of the activator CAP, the transcription will perform at a low level.

3. Finally, when glucose is present and cAMP concentration level is low in the cell, it has the effect to turn off the *lac* operon.

The above description abstracts some details to simplify the understanding of regulation dynamics, but is sufficient to cover the global picture. A formal description of this GRN, based on the above description, is proposed in Chapter 3 page 42.

2.12 Databases of biological information

Sequencing projects as the HGP have contributed to collect important quantities of data and permitted to discover numerous genes and regulatory sites. The collected data are usually shared through databases. Example is the KEGG Pathway database (KEGG) database [22, 23] which contains information on structure and function of about 15,346,261 genes and about 2000 species [24, 25]. There exist plenty of specialized biological databases that help to share and collect the important quantities of data, from microarrays database, interactions data to metabolic pathways as well as description of genes and gene products. Examples are Stanford Microarray Database [26], EcoCyc [27] (metabolic pathways, transcriptional regulation of *E. coli* bacterium), SGD [28], YeastNet [29] (a probabilistic functional gene network for baker's yeast), BioGRID [30]

(protein-protein interactions), BIND [31] and DIP [32] (protein interactions), GO [33] for a consistent definition of gene products and ERGO [34]. Some tools aggregate the data of all these interaction databases, as for example ConsensusPathDB [35, 36] which aggregates the data of 32 public resources, including binary and complex protein-protein, genetic, metabolism, signaling, gene regulation and drug-target interaction in humans. All those biological networks can be described via graph-theorical approaches or similar, as those covered further in the Chapter 3 page 29. Finally, biologists benefit from many tools to help them visualize data from those databases, as biological networks. Examples of the well known tools Cytoscape [37], VisANT [38], Pathway Studio [39], GENeVis [40], Patika [41] and Proviz [42] (dedicated to the visualization of protein–protein interaction networks). We refer the reader to [43] or [44] for additional information.

2.13 Conclusion

The material in this chapter was an introduction to the field of genetics, with the emphasis set on GRN and the processes known as gene expression and gene regulation. However, it does not cover all the details and interesting concepts of genetics such as the structures and behaviours of genes, DNAs, RNAs, the ribosomes machines, and more generally all the interesting biochemical behaviours studied by scientists. A lot more could have been said on the details of gene expression and gene regulation mechanisms from the biologist point of view, but our interest in this thesis is focused on the general mechanisms. The reader could have been interested by the details and we refer it to [13, 16] for more information. In addition, the amount of information provided by the literature is too large, often complex, and as a consequence a lot of information and details have been omitted. For example, the introduction section for the DNAs and RNAs at page 7 did not mention the RNA World hypothesis. This hypothesis suggests that RNA was the first information processing molecule to appear, even before proteins and DNA, as reported in [13][p. 4], and this hypothesis is believed by many biologists. Also, more could be said about gene regulation which in details comprises numerous mechanisms and interactions at different levels, and which are only covered from the surface in this chapter. Nothing is said neither about the role of RNA Polymeraze, a key enzyme that serves as initiator of the transcription in prokaryotes. As described section 2.12 page 26, there exist many types of regulated biological networks, each one covered with consequent literature, but since the current thesis has set the emphasis to GRN, the details of other studied networks were not clarified. For the interested reader, a summarized view of the history of genetics has been compiled in Table 2.3 page 28.

TABLE 2.3: History of genetics

1868	Basic principles of inheritance are discovered by Gregor Mendel.
1910	Thomas Hunt Morgan shows that genes reside on chromosomes.
1923	Frederick Griffith discovered that DNA carries genes responsible for pathogenicity..
1933	Jean Brachet is able to show that DNA is found in chromosomes and that RNA is present in the cytoplasm of all cells.
1941	Edward Lawrie Tatum and George Wells Beadle show that genes code for proteins; see the original central dogma of genetics.
1943	The Luria-Delbrück experiment demonstrated that mutation in bacteria was random. .
1953	Discovery of the double helical structure of DNA by James D. Watson and Francis Crick. .
1956	Joe Hin Tjio established the correct chromosome number in humans to be 46.
1956	The central dogma of molecular biology is articulated for the first time by Francis Crick..
1961-1967	The genetic code is cracked by combined efforts of Marshall W. Nirenberg, Philip Leder and Har Gobind Khorana.
1972	The first sequence of a gene is determined, for instance the gene for bacteriophage MS2.
1977	DNA sequencing methods are developed independently by Frederick Sanger, Allan Maxam and Walter Gilbert. The first full DNA genome to be sequenced being that of the bacteriophage ϕX174.
1980s	Availability of complete genomes of small viruses, as the Epstein-Barr virus in 1984.
1990	Starting of the Human Genome Project.
1998	First genome sequence for a multicellular eukaryote is released.
1995	Datum of first complete genome of a free living organism.
2003	Human Genome Project is completed : the human genome is sequenced to a 99% accuracy.

Chapter 3

Formal methods for studying gene regulatory networks

"All Models are wrong, but some are useful". – George E. P. Box

3.1 Introduction

The study of a biological system is a complex task that requires a good understanding of the system dynamics and the regulation processes. Biological networks usually contain many components and interactions, and their descriptions evolve along with the time. The existing knowledge on biological systems as well as the exchange of information via biological databases, can help to build better descriptions of those systems. However, it is crucial to have the ability to valid this complex knowledge by using more precise descriptions, such as formal models. This chapter is an introduction to modelling techniques and describes some existing formal models used in the study of biological systems.

3.2 Gene networks modelling

> The ultimate goal of the genomic revolution is the understanding the genetic causes behind phenotypic characteristics of organisms. [...] The availability of genome-wide gene expression technologies has made at least a part of this goal closer, that of identifying the interactions between genes in a living system, or *gene networks* [chap. 27 7, pp. 27-1]

Interactions between genes, proteins and metabolites are usually described visually with a graph of nodes connected by edges. These graphs carry various names as gene networks (GNs) or gene regulatory networks (GRNs) in the literature.

These GNs are usually constructed by observing how genes, proteins and metabolites interact with each other during particular experiments. The observed behaviours are analysed and collected using Microarray technology to measure the values of gene expressions, also named gene expression levels. From these measures, a network can be inferred following a multi-step process depending on the model in which these microarray data are used. GNs are helpful for various uses in modelling the different regulated biological systems. From understanding which genes are involved in regulatory interactions, how they are involved, how they cooperate and behave when facing to changes in external environment. Their study is a challenge since large number of components are involved in regulation and these components as well as the interactions between them can be very complex. But it is crucial in the elucidation of the nature and organization of interactions involved in biological organisms.

A path to achieve this elucidation is by experimenting and using formal methods and by creating new models of the studied biological systems as well as improving the existing ones. One usually starts by measuring the states (gene expression levels) through Microarray technology. Once the data are collected, one can observe or at least assume the presence of genes interactions by using various analysis techniques applied to the data. In addition to these interactions, the *trajectory* can be determined from the data. The trajectory is the evolution of gene expression through a succession of states transitions, i.e., the succession of observable states for a gene at different times of an experiment. The last step is to infer a gene network model from these expression data, a process also referred to as biological network inference. It consists in making predictions about the network. An example is the inference of the network topology, with the goal to predict the relations between the network components. This model inference is used to verify some assumptions and check if the global picture of the GRN can be understood and described accurately, which means using precise models based for example on mathematical formalisms. The achievement of description of biological networks through formal methods is essentially complemented by the support of computer tools and simulation techniques to study and answer various biological questions. All these techniques are not yet perfect but the building of an accurate model can be achieved by starting with an initial technique/model, complemented by repeated revising, and simulation of the behaviour of biological systems. These simulations reveal the adequacy of the model and help to guide to research in the right direction until a more accurate modelling of GRNs is achieved [6, pp. 67, 70] [45][p. xiii,xiv].

3.2.1 Techniques and limits

Modelling and simulation tools and techniques can be applied on inferred gene network models to acquire better knowledge on the behaviour of biological systems. By simply comparing differences between assumptions (predictions) and analysis results (observations), it is possible to make simple predictions and infer some rules. In turn methods from the field of statistics can help to acquire a deeper understanding on the experiments results and a better knowledge of the biochemical processes including the impact of drugs or diseases on human beings as diabetes and cancers. Identifying the potential causes of those diseases can be started by analysing the variation in gene expression based on data from different cells, tissues, organisms of distinct patients. A deeper understanding of those biochemical processes indirectly allows to address real medical problems of people. But the production of disease-specific cures and health care solutions requires also sufficient knowledge on regulatory networks involved in these biochemical processes to develop efficient solutions [7, p. 27-5]. At the current time a number of known gene regulatory networks GRNs have been detailed in miscellaneous databases publicly available [7]. Examples are KEGG [22, 23] and EcoCyc[46–48]. Thanks to the scientific contributions and publications at the international level, new gene expression data of many experiments and organisms are made publicly available regularly, for reuse to the research community. For instance, at this time, the KEGG database contains information on the structure and function of about 15,346,261 genes and about 2000 species [24, 25] whereas ArrayExpress [49][1] contains almost 28 TB of data for more than 56,000 genomics experiments from various laboratories.

However, despite the benefits of having a large amount of data made available, it is a challenge to choose a good model in which these data can fit since it usually depends on the needs. In addition, all models have pros and cons, and may require different parametrizations. An additional problem occurs when comparing the data of multiple existing databases between them. Practically there are good chances that all these data do not coincide at all between the different databases. This is an issue to have so many differences in existing referential since it complicates the analysis of existing data and building of accurate GRNs. But it can be explained, at least partially, by the defects of techniques and tools used to collect and analyse all these data. A classical example is the inherent problem of noise that microarray techniques are facing. The complex mechanisms used to collect and process the data imply some perturbations at several levels which may lead to erroneous conclusions for studies based on these data, and these perturbations result in variations between data. Variations in microarrays can be caused by several factors, alone or together :

[1] http://www.ebi.ac.uk/arrayexpress/

Washing The microarray washing which is made between two experiments and can be imperfect or at least not regular among the washed surface.

Hybridization The hybridization is the core process of microarrays. It is relatively complex and not error prone.

Normalization The normalization processes that occurs after the measurement of expression level, to clean the inherent noise in data. This process may be performed differently from a case to another and may consequently be destructive.

Outside the problem with microarray and measurements techniques, other differences may appear. For instance, there may be differences related to the experiments. The experiment conditions may vary at the time of analysis (compared to another similar experiment and analysis) and multiple stimuli can interfere during a particular experiment. In addition, differences may appear related to the nature of the experiment itself, the studied behaviours and biochemical processes. However, the large amount of data can balance this problem and enables to compare notable differences. For example, differences between a damaged cell and a normal cell, as well as difference between a tumour cell and a healthy cell. The analysis of differences among multiple patients is useful in the case of studying how a specific disease impacts the system in different ways, and how in turn is the GRN affected by this disease. In addition, these data permit observing the common behaviours between different experiments to improve and infer the various models of GRNs, as well as the knowledge on metabolic networks and cell signalling pathways. Still, redundant or incomplete information in pathways databases as KEGG can lead to false conclusions on a gene network topology. In [50], Ye and Doak proposed MinPath as a potential solution to this problem. It is a pathway reconstruction approach that keeps only the required pathways in gene networks and minimizes redundancy in information to reduce the gap between different pathway databases. An example is shown in Figure 3.1 page 33 where the gap is shown between MinPath and KEGG pathways or other naive pathway reconstruction methods. GRNs are large and complex and are studied for a long time. Despite the great number of mathematical formalisms developed since the 1960s, actual models are not sufficient to answer all questions neither to ensure the identification of all true interactions between genes. As reported in [6], even if some components involved in gene expression can be identified like proteins or molecular mechanisms, less is known and understood about the interactions between genes and other components in GRNs.

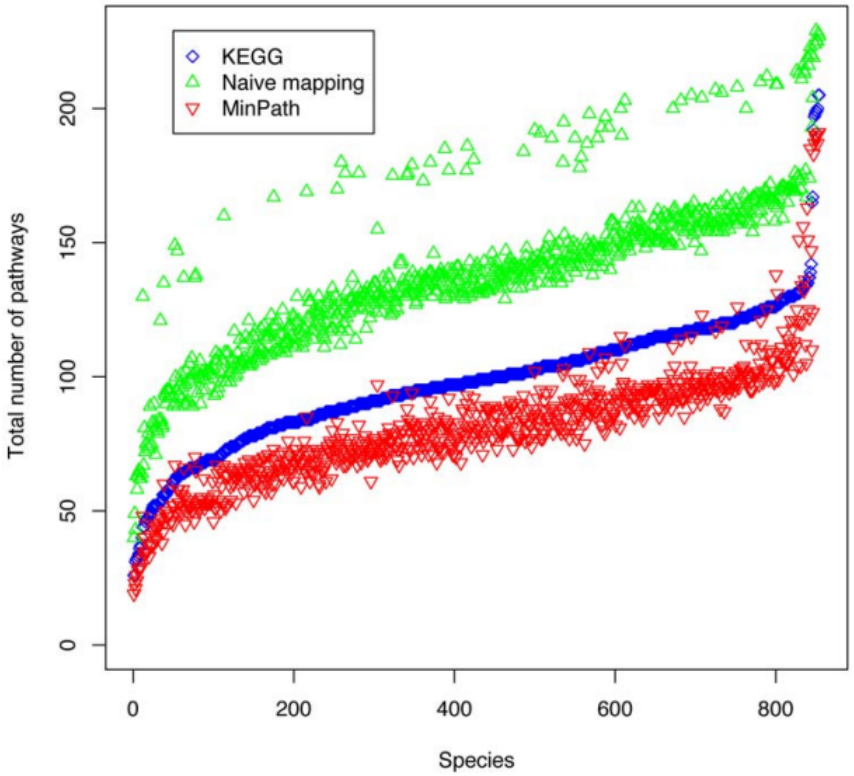

FIGURE 3.1: Number of pathways between naive approaches and MinPath reconstruction method [50].

3.2.2 Exploring organisational aspects of gene networks

With more databases and knowledge becoming regularly available, modelling techniques emerged and gained popularity. They helped to reach to a deeper understanding of the behaviour of these networks [6, pp. 69] and a better identification of some of their fundamental (biological) properties, increasing the efficiency of their modelling. A key to understand the behaviour of a gene network is to study how its behaviour is affected by external environment stimuli. Through experiments, it is possible to simulate the evolution of a large GRN in time in response to various perturbations and environment signals. For example it can be made at the level of the network topology by adding/removing nodes of the network (genes) or sections (modules, or clusters of genes that share similar functions) of the network. Changes in the expression levels can also reflect the changes in different contexts. For instance one can change the concentrations of proteins and other gene products in order to understand how it affects the regulation of overall interactions for a specific module of a complete network. It can be made by forcing either activation or inhibition of some gene, or by forcing the production of some gene products, such as RNA or proteins. Finally it can also be performed by altering the level (strength or weight) of the interactions that exist between the nodes

of the network, for instance the interactions between genes and proteins. Many properties have been discovered and the number of those properties is still growing. More recently it has been observed that the topology of GRNs is sparse, i.e., there are a few edges per nodes, and they also tend to be scale-free [7, 51]. This property plays a key role in the robustness of these networks against random topology changes. In addition, these studies indicate that particular topologies choices can guarantee this robustness. A clue is, in the evolution process, the natural convergence of these network structures to scale-free design. GNs exhibit the behaviour of having highly central nodes (hubs) responsible of the most of the overall regulation. Those hubs are directly connected to other nodes, in the worst case they are connected via very short paths. As reported in [7], the term of small-world network is also used to describe groups of nodes which are essentially connected through the same hub. GNs tend to be organized into modules, which is related to the capacity of organisms to evolve and adapt to their environment. For instance the evolutionary process tend to favour less costly connections between the nodes in networks. Connections across long paths are costly to maintain and tend to disappear. As a consequence modules emerge under the form of highly connected nodes formed of genes that share the same functions or are connected through very short paths. For instance this property has been recently experimented via an evolutionary algorithm where the evolution of network across many generations was simulated, as reported in [52]. Also, it has been observed that some nodes are responsible for regulation of genes, $2-4$ in bacteria [53] and $5-10$ in eukaryotes [54].

3.2.3 From data to modelling

Several models could describe the qualities and behaviour of gene regulation, and a considerable number of qualitative or quantitative models have emerged, with some of them described in section 3.3 page 36. Once the emphasis is set on a modelling approach, the next step of the modelling calls to data, either experimental or even faked (generated), that serve to prove the correctness of the model. The experimental data are usually collected by using Microarray technologies. The Microarray technologies have, among the existing techniques, the advantage to be fast and cheap. Then all the data, collected at different times or from different experiments, has to be organized into clusters of genes with similar expression profiles. Patterns of expression profiles may appear in a network. By comparing the large amounts of gene expression data taken at different times in from experiments, collected from different patients and databases, it is possible to identify common patterns. In turn these recurring patterns in gene expression profiles lead to identify common rules and properties of a specific GRN. As reported in [55], when differences are found in levels at which these genes are expressed

for identical cells and environment, the gene expression levels can be observed to oscillate at least around an average value. In the literature, the term of *differentially expressed genes* (DEGs) is used to describe these differences in gene expression levels.

Measurements methods from the field of statistics, such as the Spearman and Pearson correlation coefficients, may help to analyse these differentially expressed genes and find similarities in various gene expression profiles. As reported in as reported in [56, 57], those correlation coefficients are used to measure the extent to which different changes are correlated, as for example changes in values of variables. The Pearson correlation evaluates the linear relationship between two continuous variables, whereas the Spearman correlation evaluates the monolithic relationship between those variables, which means that those variables can change together but possibly at different rates. Examples of such methods are those developed by Rice et al. [58] or Zuo and Tadesse [59]. Some probabilistic (or stochastic) models could be better suited to explain the variation of gene expression values, but simple models could give a better global view of a network behaviour and require less precise data.

There are various ways to model and simulate a GRN, usually by using the current knowledge of existing biological systems. Often these models focus on a particular studied mechanism inspired by some initial assumptions. However, the choice of a model is constrained by the study to achieve, and is related to the complexity of the studied phenomenon. Even the simplest gene network is a complex system involving a consequent number of parameters. Those parameters can require a great number of experiments to fit them to the data in a process which is also referred to as *network inference* or *reverse engineering* [7][p. 27-1], [60][p. 1]. The process of network inference consists in identifying the interactions between the components of the network, in our case genes and proteins, which can be observed from gene expression data (experimental data). Identifying those interactions lead to identify possible gene networks consistent with gene expression data, to achieve the modelling of the biological system. A recurrent problem with network inference is when the amount of available data is not sufficient for accurate identification of the model and its parameters. In addition, a wrong normalization of those data as well as their misunderstanding can lead the observer to make erroneous conclusions. Finally, the misconceptions on the studied behaviours can lead to inconsistent models and unreliable visual representations of the studied system or phenomenon. Whereas these modelling techniques are not perfect, they have at least already proven their utility and are useful to illustrate and improve the knowledge on aspects of the biological systems.

3.3 Models overview

"What a model cannot do is often more important than what it can do.". – [45]

Depending on the amount of available experimental data and the biological knowledge, GRNs can be modelled at different levels [7, 61]. Indeed, the emphasis can be set on different interactions depending on the chosen degree of abstraction as well as the nature of goal to achieve. Examples are metabolic interactions, gene−gene interactions, gene−protein and protein−protein interactions. All these interactions together play a role in the overall regulation mechanism and consequently they should be part of the same model. Practically, however, this is not always the case. Since that would require a complex and costly simulation of all phenomenons and all their parameters, as well as the inclusion and understanding of all the components in a same network. In addition, the graph of GRNs including all their components can be rather large and hard to interpret. As a consequence the scope is usually limited to capture the global picture, some phenomenons are abstracted in formal models that range from discrete logic to systems of equations, while the simulation or study is limited to parts of the network, also referred to as modules. The paper [6, pp. 69] has given a (non exhaustive) overview of already known and described mathematical formalisms, such as directed graphs, Bayesian networks (Friedman et al. [62], Murphy et al. [63], Hartemink et al. [64], Moler et al. [65]), Boolean networks (and their generalizations) and ordinary and partial differential equations (ODEs) Mestl et al. [66]. In the section 3.3.1 page 36, some of these formalisms are described. However the purpose is not to give an exhaustive set of all known formalisms but rather a description of the most known ones, in addition to some less known ones. The description also includes the goals these different formalisms address as well as their different weaknesses or strengths and applicable domains. Finally, a comparative overview of all these formalisms is given at the end of this section, including the main properties of these formalisms. It has been compiled in a comparative table based on several studies and reports such as [6, 7, 45, 60, 67–69]. See Table 3.2 page 49.

3.3.1 Classification

Continuous vs discrete models Continuous models are well suited for a more accurate description but require quantitative experimental data to first fit all the model parameters. They can serve as quantitative simulations of the biological systems. Discrete models set their interest on the relationships between entities and tend to abstract the details to give a general description of the biological

system phenomenons. Examples of continuous models are differential equations while examples of discrete models are boolean networks [61] (see subsection 3.3.3 page 40).

Dynamic vs static approaches Dynamic models are usually more complete than static ones. For example static models do not have time component in them, but at the opposite dynamic models involve more parameters to fit. Examples of dynamic models are Boolean networks and linearised differential equations, while examples of static models are Bayesian networks and graph theoretical models (see subsection 3.3.2 page 38).

Deterministic vs stochastic approaches In a deterministic model, the value of gene expression is the same at different times for identical contextual parameters, while in a stochastic model the value of gene expression depends on random variables and probability distributions. Stochastic models are useful to describe complex systems by handling random connectivity and functionality. In addition, they are powerful to infer a gene network from expression data since they can better handle the presence of noise in the data. Bayesian networks are an example of stochastic model. At the opposite deterministic approaches like boolean models are too deterministic to simulate the real behaviour of a biological system and this determinism can lead to erroneous conclusions. This determinism is however an advantage in tasks that does not require such level of accuracy or realism.

Qualitative vs quantitative approaches Qualitative approaches as the boolean formalism try to answer the questions on the global picture of a system. For example by discretizing the real values of gene expression levels or by ignoring low level behaviours of such systems. At the opposite quantitative approaches such as equation-based models try to consider the details of these behaviours such as the chemical reactions at low level. As a consequence they require more computational time for the simulations since more parameters are involved. Both have proved their utility in different cases and have their limitations : qualitative formalisms are usually easier to use but are less accurate, whereas quantitative approaches tend to require a large amount of data to infer their large number of parameters before being useful. See also [70] and [61][p. 66-67] for considerations of qualitative pros and cons over quantitative modelling of GRNs. Boolean networks as well as GTMs belong to the qualitative approaches.

Synchronous vs asynchronous approaches The phenomenons that occur in a biological system such as transcription from DNA to mRNA and translation from mRNA to can be modelled in different ways. For example phenomenons such as time delays between biological events as well as their duration are modelled

by using either synchronous or asynchronous events. In the synchronous model, the gene expression levels of all genes are updated simultaneously (in consecutive time points). This is computationally simpler for large networks however it is also less accurate. At the opposite the asynchronous model is closer to the reality of biological phenomenon, in which the transitions between gene expression levels are made at different time points [68]. In this model, the time component is usually discretized to the intervals between consecutive experiments time. As described in [7], if these time intervals are small enough, the expression level of a gene i through time can be approximated as follows :

$$\frac{x_i(t_{j+1}) - x_i(t_j)}{t_{j+1} - t_j} \approx f_i(x_{i_1}(t_j), x_{i_2}(t_j), ...)$$

Whose the different components are defined as follows :

- x_i on the left is the level of concentration of gene expression for gene i.
- t_j and $t_j + 1$ are the consecutive time-steps that link the expression of x at different times.
- x_{i_j}'s describe the levels of concentration of molecules/transcription factor that influence x_i's level of expression.
- $f_i(\cdot)$ is the function specifying how the inputs influence x_i.

In Garg et al. [68], a comparative study has been made between synchronous and asynchronous in terms of computational efficiency. It reveals that asynchronous approaches are computationally costly to perform, and as a consequence it is more difficult to get decent execution time. At the opposite synchronous approaches can handle modelling of the same networks, or even larger networks, in a few amounts of time (one speaks of minutes). In [68] a more efficient method is also suggested, combining the two formalisms, which has the advantage to reduce the gap in execution time between synchronous and asynchronous modelling approaches.

3.3.2 Graph (theoretical) models

According to the notations of the graph theoretical paradigm, a GRN can be modelled as a connected and directed interaction graph structure. It can be defined as $G(V, E)$ that describes a graph G composed of vertices V linked through edges E. With the same notation applied to biological networks, nodes/vertices are genes, proteins transcripts or metabolites defined as $V = \{1, 2, ..., n\}$. While edges between nodes represent transcription factors or functional linkages/interactions between nodes. These interactions can be labelled in the graph with the type of linkage (*activation*, *inhibition*). The edges are usually defined as $E = \{(i, j)|i, j \in V\}$ [5].

The edges can be defined as tuples such as ⟨a,b,s,...⟩ where s is either equal to + to represent activation or − to represent inhibition of b by a. Each edge $a \to b$ in G can be labelled with either the function or type of the relationship. For instance with a sign (S_{ab}) which is either positive when a is an activator, or negative when a is an inhibitor of b. An example is illustrated in Figure 3.2 page 39.

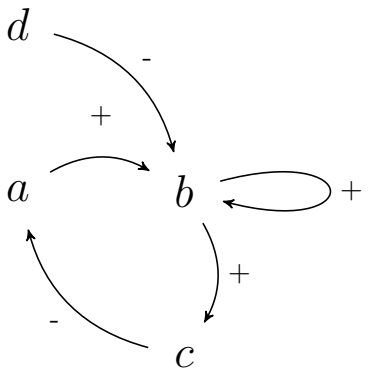

FIGURE 3.2: A simple regulatory graph with 4 nodes and 5 interaction edges

This modelling method can describe biological systems and uses the power of graph theoretical approach to make some computations and gain a better understanding on how biological systems behave. Using biological data in a graph paradigm allows applying ideas from the graph theoretical approach, including algorithms and techniques as well as ideas and engineering from statistics, graph theory and computer science fields. Despite these advantages a graph defined as G, in which the interactions between the components are described with an edge connected to a pair of nodes, is way too simple to match with real studied biological systems. It is needed to go further, for example by using a *hypergraph*. It is a generalization of a simple graph in which an *hyperedge* connects any number of vertices at once, as illustrated in Figure 3.3 page 40. Then it is possible to represent more complex relationships such as cooperative regulations in which multiple components intervene in the regulation of another one. In a hypergraph, edges are defined as tuples ⟨a,B,S⟩ where B is equal to a list of genes regulated by a, while S is the list of signs that indicate the type of regulation. In addition, edges can be associated with a weight value that indicates the strength of the regulation. This weight value is usually visible in the form of a labelled edge. As described in [7], edges can be also used to describe temporal or causal relationships which exist only under certain conditions [7, 27/9] [chap. 1 5, pp. 1] [6, pp. 70].

Once applied to graph models, inference techniques such as network inference can help to identify important elements of the network such as edges parameters, co-expressed nodes (gene or gene clusters), regulatory and causal relationships in which the network

components are involved. Co-expression nodes are the nodes which have very similar, or strongly correlated, expression profiles. As reported in [7, p. 27-9], co-expression networks are graphs that contain highly co-expressed nodes. Although these models allow describing the relationships between genes, they do not express the dynamics of these relationships, such as time component. As a consequence it is not possible to perform all sort of simulations. However, these models permit answering a lot of biology-related questions related to gene regulation and networks by using algorithms and methods on the network graph. It permits accomplishing various tasks as identification of interacting genes, comparing different gene networks to search for similarities between pathways, finding hubs (central nodes highly connected to other nodes), detecting cycles and attractors, and simply achieving a better understanding the network topology [6, 7, 61, 72]. To sump up, the main advantage of these models is to resolve the topology of the studied networks but they suffer from a lack of features required in simulation experiments.

3.3.3 Boolean networks

Modelling techniques of gene regulatory networks based on boolean logic first emerged in the 1970s. They have been introduced by Kauffman [1, 2], to face the frequent lack of quantitative data. In these models a gene is considered as expressed (ON/up-regulated) or not expressed (OFF/down-regulated), having its expression value equal to 1 (expressed) or 0 (not expressed). This expression obeys to simple logical rules such as follows :

- If the expression of a gene depends on the effect of other genes, the rule can be defined as

$$A \leftarrow (B \textbf{ AND } C).$$

  Where the value of A is conditioned by the value of B and C.
- if just B or C is sufficient to express A, rule could be written as

$$A \Leftarrow (B \textbf{ OR } C)$$

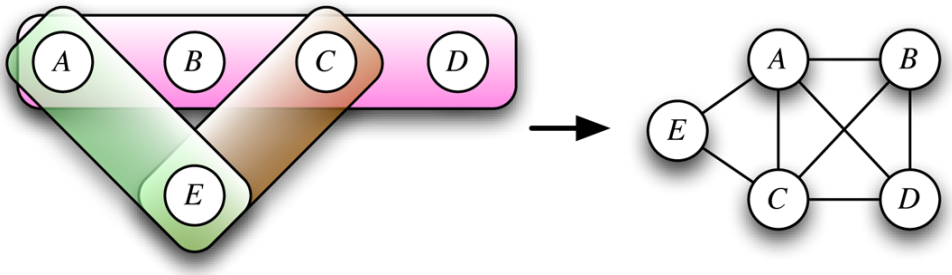

FIGURE 3.3: Comparison between a hypergraph (left) and a simple interaction graph (right) - From [71]

where the value of A is conditioned by either B or C.

- If a gene A is expressed under the condition a gene B is OFF and a gene C is ON, rule could be written as :

$$A \Leftarrow (\neg B \textbf{ AND } C)$$

where the value of A is conditioned by the value of B or C.

In order to fit the boolean parameters of such model, the continuous values of the gene expression levels (from $-\infty$ to $+\infty$) have to be converted into boolean ones (1 or 0). It can be done by using simple translation rules such as

- $IF(Expr(Gene) < 0.5$, $Gene$ is not expressed and is equal to 0 or $false$.
- Otherwise, $Gene$ is expressed and is equal to 1 or $true$.

To a Boolean network are associated different states, each composed of the different expression values of all genes at a specific time. The transition of a state to another is sometimes called an *experimental unit*, or consecutive observations of genes expression (states). A state transition pair is composed of first the gene expression values as measured before a perturbation (the input) and second the gene expression values after the perturbation (the output). The perturbation is the result of interactions and phenomenons that occur between genes, i.e., the effects of the activators and inhibitors on other genes. There exist several possible visual representations of these gene interactions and state transitions. The Figure 3.4 page 41 illustrates a simple interaction graph where genes are represented as nodes and relationships between them are represented by edges that have a different shape depending on the type of relationship (activation, inhibition).

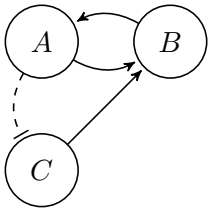

FIGURE 3.4: A simple interaction graph

The corresponding truth table is shown in Table 3.1 page 42. It maps an initial state (input) at T to the next state (output) at $T + 1$.

The state transition graph (STG) as illustrated in Figure 3.5 page 42 codes the dynamics of the networks. These dynamics show the evolution of a state to another, where the output of a state is the input of another state, and the value of a gene A at time $T + 1$ is determined by a regulation function at time T.

State	Input (T)	Result ($T+1$)
	A B C	A B C
1	0 0 0	0 0 1
2	0 0 1	0 0 1
3	0 1 0	1 0 1
4	0 1 1	1 0 1
5	1 0 0	0 0 0
6	1 0 1	0 1 0
7	1 1 0	1 0 0
8	1 1 1	1 1 0

TABLE 3.1: An example of truth table

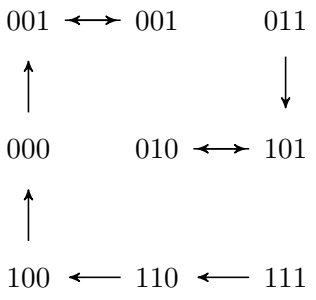

FIGURE 3.5: A simple state transition graph (STG)

In the case of network inference, boolean networks have often been used for their simplicity. They allow to easily capture the qualitative properties and the dynamics of a biological system and in addition allow to test new approaches on large networks. The simple inference algorithm for boolean networks consists in trying all possible boolean functions on inputs on all combinations of genes and inputs [7]. However, Akutsu et al. reports in Akutsu et al. [73] that a network can be generally determined with a few expression patterns, for instance with $\emptyset(\log n)$ patterns in the case of boolean networks.

Finally the wiring diagram shows the connections between the different nodes. This sort of diagram is usually used to check if some connections are missing. Here, it is useful to check if all possible state transitions are taken into account. An example is given in Figure 3.6 page 43 where the nodes A, B, C represent the values at time T and nodes A', B', C' the values at time $T+1$.

In [74], the Boolean formalism is used to describe the regulation of a previously studied gene regulatory system, for instance the regulation in *lac* operon, detailed in Chapter 2 section 2.11.3 page 24. To describe this GRN, they used to represent the most important components involved in the regulation of the *lac* operon. Then they illustrated those states through the four essential scenarios. Indeed, since the states of the network are described by *on* and *off* in the Boolean model, four states are possible to

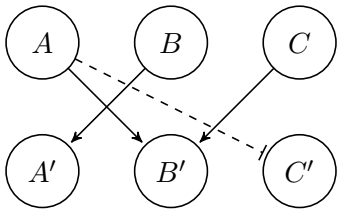

FIGURE 3.6: A simple wiring graph defined as $A' = B$, $B' = A$ and C, C' = not A

represent the different combinations of glucose/lactose. Via those four scenarios, they simulated the states of the different nodes, and as a result they determined a set of steady states, as illustrated in the Figure 3.7 page 43.

binary model	**+Glc / +Lac**	**-Glc / -Lac**	**+Glc / -Lac**	**-Glc / +Lac**
cAMP	0	1	0	1
allolactose	0	0	0	#
LacI-bound	1	1	1	#
lacZYA mRNA	0	0	0	#
LacZ	0	0	0	#
LacY	0	0	0	#
lactose_int	0	0	0	#
Observed				
lac operon	off	off	off	on

FIGURE 3.7: Steady states in the *lac* operon - From [74]

Note that the presence (resp. absence) of the glucose is noted $Glc+$ (resp. $Glc-$) while the presence (resp. absence) of the lactose is noted $Lac+$ (resp. $Lac-$). The rows in the Figure define the distinct elements involved in the regulation of the *lac* operon. In this Figure, the LacI-bound represents the LacI repressor bound to the *lac* promoter, and te lacZYA defines the three structural genes of the operon. For additional details, we re refer the interested to [74].

3.3.4 General logical method

In 1973, René Thomas proposed in [3] a method that extends the Boolean network with multivalued variables and allow asynchronous transitions between states. This method focuses on the logic of genetic interactions and gives the ability to reason on the dynamics of gene networks via a simple framework [5, 6, 61]. Thomas' work has been largely reviewed and various extensions of his method have been proposed to describe number of biological systems. It can be used to infer information about the dynamic properties of such system from the corresponding interaction graph. In addition, it

allows to reason on the system dynamic properties even in the absence of information on the value of network parameters. Finally, it enables the automatic determination of the appropriate logical parameters of the network. The gene networks are usually described by R. Thomas in terms of graphs similar to those of section 3.3.2 page 38.

Multiple notions are introduced relatively to states of the networks. For example, since the number of states of a boolean network is finite (2^N possible states) and since a boolean network has a deterministic nature, a trajectory will soon or later visit a previously observed state. This phenomenon is referred to as a *cycle* or *attractor*. In [75], R. Thomas speaks of stable states to identify the cyclic behaviours in a system. An example can be visualized in the case of state 2 in Table 3.1 page 42 where the state 001 leads to state 001. The same cyclic behaviour can be observed in Figure 3.5 page 42 since it is another view of the same network. In [5] Thomas' method has been complemented by an approach using *temporal logic* and *model checking techniques*. This method permits to constraint the logical parameters according to hypothetical or observed properties, then it can generate the parametrizations compatible with those properties and test their consistency. When no valid parametrization can be found, it means that the initial observations are inconsistent. It is useful to test the validity of observations or assumptions on a dynamical model. However, such a method also has its limits and requires enumerating all parameters of a network, which can be a complex task for large networks. Allowing the capture of the qualitative properties of biological networks, and even if they proved their utility to answer realistic questions on biological systems, Thomas' formalism has limitations. For example, it ignores that genes can have different levels of expression at different times and as a consequence the interactions are viewed as synchronous while in real biological networks those interactions are usually asynchronous. In addition, Thomas' method considers the levels of expression values as boolean values (1 or 0) while in reality the levels of expression are continuous values. However, the general logical method can serve as a qualitative representation of a system to capture its general picture, i.e., easily understand and analyse the global dynamics until more experimental data become available for the concerned system, which would then allow to use of a more accurate approach. Those models are not sufficiently accurate to answer all the biological questions on GRNs. This simplicity is also a strength especially at the modelling of very large networks. Indeed, quantitative formalisms often lack required experimental data, whereas qualitative formalisms are less demanding. Thomas' model has been extended to other approaches like probabilistic boolean networks (see section 3.3.5 page 45) which introduces stochasticity, asynchronous boolean networks, kinetic logic models with multivalued variables/functions (see section 3.3.6 page 46), multiplexes to simplify description of interactions by reducing the number of

network parameters [76] or piecewise linear differential equation systems [67][5, pp. 1], [45][p. 21-22], [19, p. 2].

3.3.5 Probabilistic Boolean networks

The probabilistic boolean network (PBN) formalism was introduced in 2002 by Shmulevich et al. in [77] as an extension of the boolean network model where the regulation functions obey to a probabilities-based logic and several predictors functions are associated to each target gene. A PBNs can thus be visualized with a graph representation, where stochastic edges describe probabilistic dependencies between nodes. This probabilistic feature affects also the state transition table or graph of such networks, since the existence of each transition is conditioned by a probability. This property is illustrated in the STG in Figure 3.8[2] page 46.

PBNs were developed to express the randomness inherent to biological systems, to be as close as possible from the real behaviour. They consist in modelling stochastic extensions of boolean networks in which genes are either repressed or activated at different times, depending on the values of gene expression levels. PBNs are good at describing randomness in biological networks at an abstract level. But despite this strength they do not handle the context specific mechanisms involved in gene regulation. For example some genes interact most often in particular context than in others, such as contexts related to a specific disease or patient type [78]. This is why this model is not sufficient and has in turn been extended. Examples of extensions consist in context-sensitive boolean models such as those described in [79] and [80]. Those models are sort of constrained PBNs where the constraints are set on the usage conditions of probability. Finally, the PBN formalism is not the first attempt to introduce randomness in boolean networks formalism. For instance random boolean networks were already introduced in 1969 by Kauffman in [1, 2]. There exist some tools in which stochastic behaviours can be simulated. An example is that of SGNSim, a Stochastic Gene Networks Simulator which is able to model GRNs. The interested reader can find more details on this tool in [81, 82]. In addition, stochastic mechanisms of gene regulation and their modelling are also studied in [55] where a semi-stochastic model is proposed to simulate the steady states of biological and chemical systems. In this model, based on Monte-Carlo approximations and Gillespie algorithms, biological and chemical systems, in which regulation is ruled by stochastic mechanisms, are simulated from master chemical and stochastic equations. Additional details on PBNs are available for the interested reader in [53, 73, 83][p. 250] and [69][p. 2].

[2] http://adam.plantsimlab.org/userGuide.pl

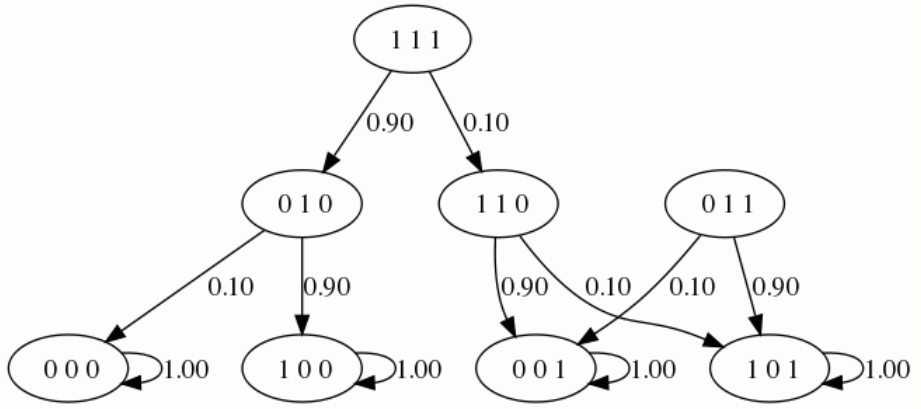

FIGURE 3.8: A probabilistic boolean network with predictor functions - from ADAM website[84]

3.3.6 Kinetic logic models

The Kinetic logic model was introduced by Thomas in 1979 ([4]) and further refined in the 1990s. It introduces asynchronous description and in addition increases the accuracy of gene expression levels. For instance, to the state of each gene is associated a discrete value, such as "not expressed", "expressed at low level", "expressed at a medium level" or "fully expressed". By generalizing the boolean networks, the kinetic logic models allow handling the transitions between states with more granularity and as a consequence allow describing the values and rates of change with more accuracy. As described in [67][p. 12-13, 20, 28] this model introduces variables, associated to gene products, to code the absence or presence of particular gene products. Once the level of concentration of a gene product has reached a certain level the gene product is marked as present; otherwise the gene product is marked as absent. Other notions are introduced to describe the various states and dynamics. Functions are introduced to represent the gene expression levels. Time delays are introduced to code the delays between biological processes as well as their duration. This is useful to take into account the delays/durations of biological processes such as transcription, translation, as well as the delays before activation/inhibition of a gene once a certain quantity of gene product is accumulated. Functions are used to represent the rate of synthesis, i.e., the evolution of the concentration levels of gene products and proteins through time. In [67], the notion of feedback loop has been introduced to describe circular chains of interaction. Depending on the number of inhibitory interactions in the loop, the loop is either positive (even number of interactions) or negative (odd number of interactions). Those feedback loops were introduced to describe systems having chains of cause-and-effect related to the presence of circuits or loops, as described in [45][p. 22, 32-33] [67][p. 10]. The introduction of the temporal dimension in kinetic logic model as well as positive and negative feedback loops described in Thomas and D'Ari [67, 67], Remy et al. [85] enabled a more precise and

also more complex description of biological systems. As a consequence this model has a higher predictive value in comparison to boolean networks, and kinetic logic models are obviously best suited for systems involving multiple feedback loops phenomenons.

3.3.7 Bayesian networks

Bayesian Networks (BNs) are a generalization of the boolean model. They have become well suited for representing stochastic models in the uncertainty of artificial intelligence. In addition, they have been largely used for modelling GRNs. The base of this formalism is the Bayes Theorem, which establish a relationship between the current probability of an event with prior probability. In its simple form, this theorem is stated by the following equation :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

- A and B are events, $P(A)$ and $P(B)$ are the independent probabilities of respectively A and B.
- $P(A|B)$ is the probability of A given B.
- $P(B|A)$ is the probability of B given A.

Multiple reasons can explain the choice of the Bayes theorem to describe GRNs. First, the gene interactions are complex and often inferred from noisy or incomplete data, which favours models that can play with uncertainty. Second, genes regulate each other and are not expressed independently, which favours models where causality between phenomenons can be described.

Structurally, a BN is a specialized graph model that handles probabilities relationships between variables and allows observations of some hypothesis or data. As described in [69], the topology of a BN consists in an acyclic directed graph where nodes are random variables with discrete or continuous values. The choice between discrete or continuous values to represent genes expressions levels, while the edges describe dependencies between nodes. The choice between discrete or continuous representation depends only on the type of data to deal with. BNs can model causal networks where an edge from B to A states for B "causes" A, as described in [63]. There exist variants of BNs. Examples are Gaussian networks, module Networks, mixture Bayesian networks, state-space models (SSMs) and dynamic Bayesian networks which are able to infer interactions from time-series data [69, 83, p. 270]. Finally, an example of tool is given with Banjo[86][3] which help to learn the structure of static and dynamic Bayesian networks.

[3] http://www.cs.duke.edu/ amink/software/banjo/

3.3.8 Differential-equations-based Models

Differential equations modelling is a major formalism in the mathematical biology, favouring continuous, deterministic and more realistic approaches of discrete and stochastic mechanisms. They translate the biological systems into a set of differential equations, which describe chemical and physical constraints, and a set of variables which describe concentrations of gene products, for instance proteins or mRNAs. Modelling methods making use of differential equations allow a better understanding the mechanisms involved in GRNs since they allow a more detailed description of these mechanisms. As described in [87], a basic differential equation model to compute gene expression level can be defined as follows :

$$g_1(t+\Delta t) - g_1(t) = (w_{11}g_1(t) + ... + w_{1n}g_n(t))\Delta t$$
$$\dots$$
$$g_n(t+\Delta t) - g_n(t) = (w_{n1}g_1(t) + ... + w_{nn}g_n(t))\Delta t$$

- where $g_i(t+\Delta t)$' define the level of expression of gene i at time $t+\Delta t$.
- and w_{ij} defines the weight of influence of gene j on gene i where $(i, j = 1...n)$

In this system of equations, the expression level of genes at time $t+\Delta t$ depends on all levels as expressed at time t. Optionally, for each gene, additional extra terms can be added to indicate the influence of additional factors. Since the model uses a set of equations to model biological mechanisms, it is comparatively less abstract than boolean or Bayesian formalisms. But this quantitative approach also requires a large amount of prior data and biological knowledge before the behaviours can be modelled in a system of equations. They have been largely extended in the form of specialized frameworks to address different needs. An example is Langevin's approach which addresses the case of non-deterministic systems where identical states can lead to different possible outputs. Other forms and extensions of the equations-based models exist such as piecewise affine differential equations (PADEs)[61][p. 54-55], non-linear differential equations, piecewise-linear equations, qualitative linear equations, linear additive models, stochastic master equations, partial differential equations and other spatially distributed models, as reported in [7, 69]. The interested reader can also refer to [88] for more information on those models and their extensions.

3.3.9 Comparison

Since the number of identified models in the literature is quite huge, the comparison of models which is illustrated in Table 3.2 page 49 set the emphasis on the most models

Models	Properties									
	Qualitative	Quantitative	Discrete	Continuous	Deterministic	Stochastic/Probabilistic	Synchronous	Asynchronous	Static	Dynamic
Graph theoretical models	✓		✓	✓	✓	✓	✓	✓	✓	
Boolean networks	✓		✓		✓		✓	✓[89]		✓
Probabilistic Boolean networks	✓	✓	✓		✓	✓	✓	✓	✓	✓
Bayesian networks		✓	✓		✓	✓	✓	✓	✓	✓
Kinetic logic models	✓	✓	✓		✓	✓		✓		✓
Differential-equations systems		✓		✓	✓	✓	✓	✓	✓	✓

TABLE 3.2: Synthesis table of formal models used in gene networks modelling

cited in the current thesis. As well, some models are considered with their extensions included, since those extensions improve the initial model. This short-cut is made to simplify the reading of the table and the comparison of model features. As a consequence, the table is not exhaustive and should be interpreted with caution by the reader.

3.4 Existing modelling and simulation tools

3.4.1 Model checking tools

Determining if a model satisfies certain properties is a complex task since it requires a lot of analysing and validating steps. Practically, this can be done with the support of computer tools that automatize the verification of model properties by a systemic and exhaustive exploration of the model, under the condition that those properties can be clearly defined. Following this idea, several tools have been developed such as model-checking tools or simulators to emulate the behaviour of large biological systems with a great number of state transitions.

Model checking tools have emerged 30 years ago and help to achieve a better understanding of GRNs. Both model checking tools and simulators usually require data in a certain form to fit their parameters. For example, quantitative tools require numerical

values to fill their kinetic parameters, in addition the concentrations of molecular components need to be specified as well. Some tools such as DBSolveOptimum[4], Gepasi[90][5] and PLAS[91] (Power Law Analysis and Simulation) are used for simulations of biochemical reaction networks by using differential equations systems [92]. This section proposes a non exhaustive list of some existing model-checking and simulation tools commonly cited in the literature. The interest reader can also refer to [44, 61, 93, 94] for more details on model-checking tools. Finally, Chapter 4 page 68 introduces a programming language which is perfectly suited to implement model-checking techniques.

NuSMV [95, 96] is a open-source symbolic model-checker for analysing synchronous finite-state and infinite-state systems based on Binary Decision Diagram (BDD). It is used by other bioinformatics tools such as BIOCHAM (see page 50 in this section). NuSMV supports the verification of system properties against requirements written in temporal logics, for instance the Linear Temporal Logic (LTL) and CTL formal specification languages, respectively invented in 1977 and 1981, whose the syntax is given in the NuSMV user manual. NuSMV is well suited to determine if a biological model satisfies a set of biological properties, which is particularly useful in the case of GRNs which are systems with complex behaviours.

Biochemical Abstract machine (BIOCHAM)[97][6] is a modelling environment for systems biology. It supports static analysis and inference of unknown model parameters from temporal logic and can be used for both qualitative and quantitative models. The specifications are written in a simple rule based language and are then verified against the NuSMV model-checker. Users can directly encode biological properties using CTL formulas into BIOCHAM [94].

Selection of Models of Biological Network (SMBioNet)[98][7] is based on the multivalued logical formalism of René Thomas and CTL. It consists in a tool for modelling of GRNs to help the biologist in the task of verifying the coherence of biological models. This verification is performed against temporal properties extracted from logic formulas. The tool can generate the valid parametrizations for a network and the matching STG, starting from an input composed of a GRN and logic formulas that express assumptions and other knowledge. The generated parametrizations are consistent with the proposed input, since they are generated using the NuSMV model-checker [99][p. 111].

[4] http://insysbio.ru/en/software/db-solve-optimum
[5] http://www.gepasi.org/
[6] http://lifeware.inria.fr/biocham/
[7] http://www.i3s.unice.fr/ richard/smbionet/

GINsim[100][8] is a Java software for modelling and simulating qualitative models of GRNs which is based on a discrete logical formalism. It is able to analyse and perform simulations of the qualitative behaviour, from inputs given by the user under the form of asynchronous and multivalued logical functions. It uses logical regulatory graphs for GRNs in addition to STGs to represent the dynamic behaviour. The graphs can be exported in the GINML file format, which is a XML-based format that extends Graph eXchange Language (GXL) and originally designed for GINsim to support the two types of graphs.

Genetic Network Analyzer (GNA) developed by de Jong et al. [92] is a tool for the modelling and simulation of GRNs. It is based on a qualitative simulation method using piecewise-linear (PL) differential equation models. GNA parses a PL model given as input, proceeds to the simulation of the system and outputs as a result a transition graph with qualitative states and transitions between those states. It helps to assert the correctness of a model built on assumptions. Specifications of biological properties are encoded using a pattern system, or by using CTL/CTRL (*Computational Tree Regular Logic*) formulas and then verified using model checkers such as NuSMV and CADP [94][p. 3]. In [92], de Jong et al. demonstrated the utility of GNA on the regulation of initiation of sporulation in *Bacilus subtilis*. GNA and CTL are also used in [101] and [102] where Batt et al. proposes an approach based on combination of qualitative simulation and model checking techniques.

3.4.2 Data exchange formats

XML-based formats have become popular for data management tools as well as in existing simulation, modelling and visualization computer tools applied to the field of biology. Many XML-based format have emerged in the last years, such as those cited in the previous subsection 3.4.1 page 49. In addition, many other popular formats have emerged, for instance some popular pathway file formats such as Systems Biology Markup Language (SBML), CellML, BioPAX and SIF [103].

SBML The SBML format was first released by 2001. It is able to describe various models related to computational biology, and to describe many classes of biological phenomenons such as gene regulation, cell-signalling pathways and metabolic pathways. It has become the standard for the description of computational models in systems biology since it was first cited in publications and used in software in this field. But the SBML language can be used to describe a larger set of processes. In addition, SBML has the purpose of ensuring the survival of models regarding to

[8] http://ginsim.org/

the lifetime of the software initially used to create them and allow the researchers to work with different software environments without having to rewrite models. A review of the advantages of popular SBML format has been studied in [104] and [105].

CellML CellML is a popular format similar to SBML and neither limited to description of biochemistry models. For example, it can be used to store and exchange computer-based mathematical models. It consists in the description of components and the relationships between them. Each component can be abstract or it can have some concrete interpretation. The reader can refer to [106] for more information.

BioPAX The Biological Pathway Exchange (BioPAX) format is used to handle information on biological pathways and topologies of biochemical reaction networks. It can describe genetic interactions and GRNs. The reader can refer to [107] for more information.

SIF The Simple Interaction Format (SIF) format was created by Cytoscape for visualizing molecular interaction networks. SIF can be used to build a graph from a list of interactions and combine different sets of interaction into a larger network. At the opposite of some other known formats, SIF do not handle layout information.

There exist many other XML-based formats such as PSI-MI for description and exchange of protein-protein interactions or yet Systems Biology Graphical Notation (SBGN) that is defined by its authors as "an effort to standardize the graphical notation used in maps of biological processes". Pathway data in those formats can be downloaded from various online resources. Then pathway vizualisation tools such as Cytoscape[37] and others can import and work with those files in various formats. Sometimes tools cannot handle the input format, but there exist tools to convert a format to another. For instance, tools such as KEGGTranslator[108][9] have the ability to convert pathways from the KGML format into BioPAX, SBML, XGMML and SIF formats. KGML is a pathway file format, also XML-based and used by the popular [22, 23] database. The interested reader can refer to a review of standard tools and exchange formats is given in [44].

3.5 Conclusion

The current chapter introduced the study of biological processes and phenomenons from the computer scientist point of view, with the emphasis set on the modelling of

[9] http://www.ra.cs.uni-tuebingen.de/software/KEGGtranslator/

highly studied phenomenons known as GRNs. Their study include the use of various analysing, inference and simulation techniques usually supported by the use of computer tools and formal methods from the fields of mathematics and statistics. The combined use of various techniques can often help in increasing the knowledge of studied phenomenons and increase the accuracy of the existing models for a particular GN. However, those biological processes are often complex to study in the large and require a large amount of data depending on the goal and the complexity of the phenomenon. In addition, some GRN are known to be particularly large and their study is computationally costly, since the complexity grows exponentially with the number of components and interactions in those GNs. It was shown that a variety of models have been proposed in the last decades, with their pros and cons, to elucidate the role of genes and their interactions. Based on a set of formal methods and tools among the existing ones, it was shown that no particular tool or formal method is practically applicable to the modelling of all the known biochemical processes involved in a complete GRN. At the opposite, the research is often limited to the study of particular phenomenons at a particular level of abstraction, using specific models to model specific behaviours. Then the most common models have been covered and compared regarding to their main properties and a classification of those models has been proposed. Finally, it was shown that many data exchange file formats have been created to describe the biological processes and enabled the exchange of information related to this field. This chapter described a few tools related to the modelling and simulation of biochemical processes. In addition, it introduced the concept of model-checking which is used to check some properties of the model of a GRN in order to eliminate inconsistencies in models. On the overall, the study of those biological processes may require fitting complex model parameters with a large amount of data and the exchange of these data between researchers is important.

Chapter 4

Constraint logic programming for analyzing GRNs

4.1 Introduction

Many formal methods and models for describing the dynamics of GRNs have been proposed in the literature. They address the modelling of complete biological systems, or at least subsets. These formal methods have been improved and extended in many ways. Although there are numerous, their efficiency is not as easy to prove. At least, those models can be partially validated by using model-checkers. However, those model checkers require a good understanding of the dynamics of a system. There are questions related to the study of those system dynamics which involve issues such as an uncertain or incomplete understanding of some phenomenons and interactions. This chapter is an introduction to a promising technique for solving problems with uncertain or unknown parameters : constraint programming and, in particular, constraint logic programming. In addition, it will describe the advantages of using those techniques to solve real world problems. Finally, it will give some examples and clues related to an efficient usage of those techniques for solving problems related to the study of GRNs.

4.2 Problems solving approaches with constraint solvers

In Chapter 3 page 36, several formalisms were described, each one solving a particular class of problem in the context of modelling GRNs. The studied qualitative and discrete models allow reasoning on the regulation of biological systems from a theoretical point of view. They permit reducing the distance between the description of a biological

system and the reality, without the need to process a large quantity of data. Based on experimental data, it was shown how some models could be inferred and checked by the biologist researcher against experiment results. However, this process requires from the biologist a lot of time to compare and verify their assumptions in one of these models. The biologist research has usually to fit the model parameters with experimental data, elucidate and the gene network topology and verify step-by-step the correctness of the results through experiments. This process is repetitive, often costly in time since it requires a lot of verifications, a good understanding of the formal models and dynamics of the studied systems. A lot of time is usually wasted on choosing models, collect appropriate data and make them fitting the parameters the global complexity. The cost of those steps could be reduced with a tool that does the preliminary job of guiding the researcher in the right direction. A clue could be to let him reason of the assumptions about a biological system, rather than fitting this model with data and reasoning on its correctness. Such a tool could lead the researcher propose some rules and assumptions, and check their consistency visually fitting any model parameters with experimental data.

Some interesting tools have already been described, such as the model-checkers in Chapter 3 page 49. Model-checkers usually use formal specification languages such as CTL to describe the model rules, or are even built upon engines that use tools dedicated to logic reasoning and as a consequence perfectly suited to reason on model correctness. Using a model-checker to verify a model supposes that the dynamics of the system are well understood and can be described before being verified by the model checker. This is not always possible, especially for systems where a large amount of parameters are hidden or unknown. In such situations, it is more pertinent to start with studying a smaller part of the system, such as modules as seen in Chapter 3 page 36. This is equivalent to apply model-checking on specific regulation, at the opposite of the overall. Even in this context, the goal is still to apply verification and logic reasoning on a model. Model-checkers often use a specification language for specifying the rules and perform logical reasoning to validate them. Those specifications are logically analysed and checked against the model, which is a repetitive process. It requires from the model-checker user the study of a specification language and a good understanding of the model dynamics, before an existing model can be checked.

In addition to model-checkers which are very interesting for reasoning and verifying a model, there exist in the field of computer science other powerful tools and programming languages perfectly appropriate for logical reasoning, as well as reasoning on models defined with rules. Usually, traditional approaches to solving logical problems are to start an algorithm in a traditional programming language and start to translate the domain knowledge and the problem parameters in an equivalent form accepted by the

programming language. In addition, the algorithm for solving the problem, i.e., the resolution logic, has to be specifically programmed in a programming language starting from zero. It can be done in most programming languages with a certain amount of effort. Other approaches for solving a problem use existing tools, often very specific to a domain. However, there exist more generic approaches to problem solving. An example consists in reusing existing, generic, algorithms to solve problems. In this category, constraints solvers are quite popular. They exist in many forms to address different classes of problems and have already been implemented in various existing programming languages. Some examples of constraint solvers implemented in different programming languages are given in [109]. A *constraint solver* consists in a generic mechanism which can help to find the satisfiable answer(s) of a problem expressed as a system of constraints. These constraints, defined by the user, translate hypothesis about the system and are parsed by the constraint solver which finds a way to a solution. The user-defined constraints themselves guide the solver to eliminate unsatisfiable solutions by reducing their search-. The next section 4.3 introduces a programming paradigm dedicated to constraint solving.

4.3 Constraint logic programming

In the constraint programming (CP) paradigm, the relations between variables are expressed with constraints. At the opposite of classic programming languages, no step-by-step sequence to the right solution has to be expressed, but rather the properties and qualities of the solution to be found.

Constraint logic programming (CLP) in turn is a specialized form of the CP paradigm, first defined by Jaffar and Lassez in [8]. CLP merges the benefits of Horn clauses logic programming and constraint solving in a flexible and expressive paradigm which is in several cases more efficient than other existing ones. A logic program is composed of rules written in the form of *clauses*. A typical logical clause is composed of a *head* and a *body*, and expresses the notion that this head is true if its body is true. Otherwise, the formula expressed by the head is false. Logic programming allows to reason on atomic formulas and infer new satisfiable properties from an initial set of goals, i.e., conjunctions of atoms, whereas constraint solving set the emphasis on finding appropriate parameters' inference regarding to a set of clauses. An example of logical clause can be defined as follows:

```
man(X) :- human(X), age(X) >= 18.
```

Where the head is `man(X)` and the body is `human(X), age(X) >= 18`. It can be read as

```
man(X) if human(X) and age(X) >= 18.
```

Namely, any x who is human an is 18 years old is a man.

In addition, a logical clause defined with an empty body can be considered as a fact. As an example, this simpler logical clause

```
human(author).
```

expresses the fact that `author` is a human and could be a satisfiable valuation for `X` in `human(X)`. Those clauses, also known are Horn clauses, are quite simple and have their extensions in order to represent various concepts, such as negation or conjunction of logic formulas. Constraint logic programming, which is derived from declarative programming, includes constraint satisfaction. A constraint program consists in a description of the domain values and problem constraints which are properties to keep satisfied for these domain values. Formally, a constraint satisfaction problem (CSP) can be defined as a triple $\langle X, D, C \rangle$ whose different components are defined as follows :

- $X = \{X_1, ..., X_n\}$ describe a set of variables,
- $D = \{D_1, ..., D_n\}$ describe the domain of values,
- $C = \{C_1(S_1), ..., C_m(S_m)\}$ describe a set of constraints where each S_i is a set of variables.

With the values of a variable X_i depends is upon the values of the non empty domain D_i.

The solution to a CSP is a valuation of domain values to $S_1 ... S_m$ that satisfies all the constraints. A classical example of constraint satisfaction problem is that of the map-colouring problem, which is illustrated in Figure 4.1 page 59. In this example, the goal is to colour the map so that adjacent regions have different colours. A definition of this problem is as follows :

Variables WA, NT, Q, NSW, V, SA, T represent the 7 regions on the map.
Domains $D_i = \{red, green, blue\}$ represent the domain of possible values, for instance a set of three colours.
Constraints adjacent regions cannot have the same colour.

An example of a satisfiable valuation is
$\{WA = red, NT = green, QT = red, NSW = green, V = red, SA = blue, T = green\}$

FIGURE 4.1: Constraint satisfaction - an example with the map-colouring problem - From [110]

The same problem can be illustrated in the form of a constraint graph which makes their understanding easier by visualizing the constraints and variables, where nodes of the graph represent the variables and the edges represent the constraints. An example is illustrated in Figure 4.2 page 59. Note that being an island, T is not constrained.

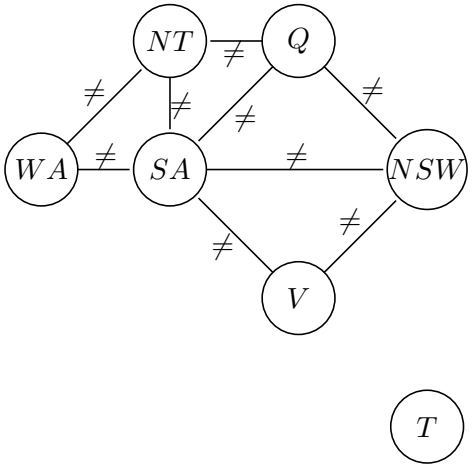

FIGURE 4.2: An example of constraint graph for the map colouring problem

At the opposite of other programming paradigms where step-by-step instructions have to be specified for finding a solution, the purpose of CLP is to avoid the writing of those instructions and let the user focus on the goals. Examples of goals could be to find the satisfiable valuations for unknown variables or find an optimal, or the best solution, to an optimisation problem. In this purpose, the user does not need to code the process of solving a problem procedurally (= "HOW"), but only to declare the properties of the final solution (= "WHAT"). Practically the problem is expressed in *Horn* clauses extended in the syntax of CLP. Problem solving is then delegated to a constraint solver that compiles it in its own internal representation. The constraint solver searches for

a state of the universe in which as many constraints as possible can be satisfied, by using solving strategies. Basically, the solver searches for values for all the variables. It evaluates the user-constraints or user-defined rules until is has explored all the possible paths or a satisfiable solution is found. A problem is said to be satisfiable if there exists a solution according to it. Otherwise it is unsatisfiable. A solution is reached when there exist a valuation of all variables that satisfies all constraints without exception. The Figure 4.3 page 60 illustrates the global flow from the initial problem to the solution.

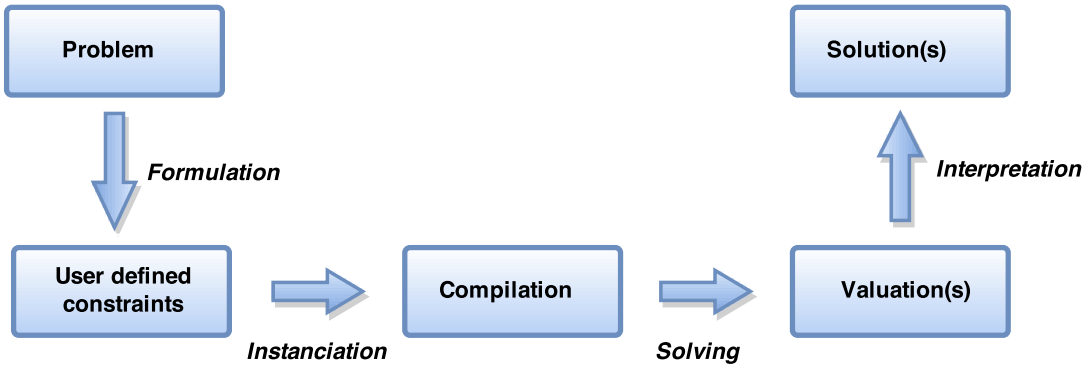

FIGURE 4.3: Constraint programming - flow from a problem to its solution

In addition to be well-suited for constraint satisfaction problems, constraint solvers are also perfectly suited to reason on problems of combinatorial complexity which are at the heart of many real applications, at the opposite of traditional programs for which such task is rather difficult. Additionally, a traditional program is usually composed of problem-specific instructions and algorithms and is by definition less easy to adapt when the problem evolves. Constraint solvers usually require only changes in the description of the solution properties, i.e., the user-defined constraints, not in the solving algorithms/strategies. Faced to a constraint solver, a programmer has only to code those specific domain constraints in a programming language supported by the constraint solver. Since there are yet generic solving algorithms implemented internally in the constraint solver himself, nothing more is usually required. As a consequence, constraint logic programming can be considered as one of the purest programming languages, combining mathematical facets, traditional computer programming and artificial intelligence, where only user-defined constraints need to be programmed to specify the behaviour of a system, and the knowledge of a constraint logic programming can be reused in the context of any other constraint programming language.

4.3.1 Solving of constraint satisfaction problems

Solving of a constraint satisfaction problem requires a particular form of search. Obviously, a satisfaction problem can be solved by generating all possible valuations for variables of the problem and then testing if a satisfiable solution exists. This is also called *generate and test*. It can be applied without exceeding the computation time when the domain values are very limited and the set of solutions is finite. However, some problems have an infinite number of solutions. For instance, there is an infinite set of satisfiable valuations for $X > Y$, where X and Y are numerical values. In addition, enumeration techniques cannot be applied for a continuous domain of values real number values since the enumeration would be infinite. Finally, this is not a very efficient way to take an advantage of CP. For instance, the constraints are only used passively to check the satisfiability of a solution after the valuation is performed. More efficient techniques flavour the validation of solution along the valuation of variables and as a consequence highly reduce the amount of performed valuations required to build a satisfiable valuation. The most used solving techniques include *backtracking*, *local search* and *constraint propagation*, described in the subsection 4.3.2.

4.3.2 Solving techniques

Backtracking The technique of backtracking ensures to find any existing solutions that are consistent with the constraints. This algorithm starts with all variables being unassigned. First a variable is chosen and the algorithm assigns all possible valuations to this variable. Then for each consistent valuation obtained that is consistent with the user defined constraints, a recursive call is made to the same algorithm, except the variables are initialized with the previous valuation obtained, consequently the previously assigned variables will not be reassigned again but at the opposite another variable is chosen. The algorithm proceeds step-by-step, trying all possible valuations until their exploration has been achieved, as illustrated in Figure 4.4 page 62. If a consistent valuation has been found for all variables, this solution is returned. In some cases the algorithm "backtracks". It happens when the valuation of a variable cannot be satisfied regarding to the previous performed valuations. In this case, the algorithm cancels the current exploration, until it is able to try another valuation not performed before. The algorithm stops when a solution is found, or when it can't go further, i.e., all possible consistent valuations have been tried.

Local search Local search is an incomplete method, it may eventually find a solution to a problem but it is not complete. At the opposite of backtracking, local search

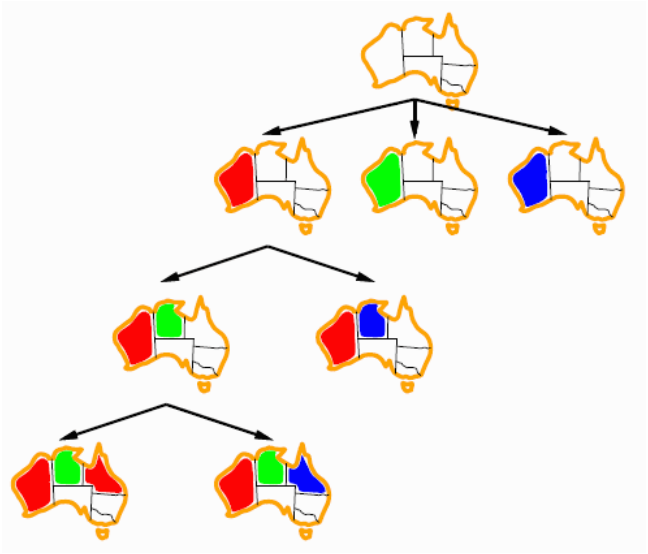

FIGURE 4.4: Constraint resolution techniques - An example with backtracking applied to the map-colouring problem - From [110]

may stop without finding any solution, even if a solution exists. This algorithm uses a state to represent the variables assignments. At each step, the algorithm modifies the value of a variable to increase the number of total satisfied constraints in the current state. This reassignment is done so that it violates the fewest possible constraints. Since the algorithm tries to minimize the amount of violated constraints, it uses a cost function that returns this amount. At each step, this cost is evaluated along with the assignments. This algorithm is well suited for particular classes of problems, such as the n-Queens problems.

Constraint propagation Constraint propagation is a form of inference, used to simplify a constraint satisfaction problem and make it easier to solve. This is done by transforming an existing constraint satisfaction problem formulation in a form which is simpler but equivalent. Both formulations will have the same set of solutions. The idea is to creating new variables, reinforcing existing constraints and reducing domain of variables to converge easier to the solution. The propagation of a constraint, i.e., the assignment of a value to a variable, reinforces the constraints locally. For instance, in the map colouring problem, the assignment of a colour to a region reduces the possible assignments for other regions, which can be illustrated by Figure 4.5 page 63, where the assignments $WA = red$ and $Q = green$ propagate constraints for assignment of other variables, for instance NT and SA (last line). This technique can also be used to prove the satisfiability or unsatisfiability of a problem. However, it is only applicable and efficient in certain cases and cannot provide early detection of all constraint propagation failures.

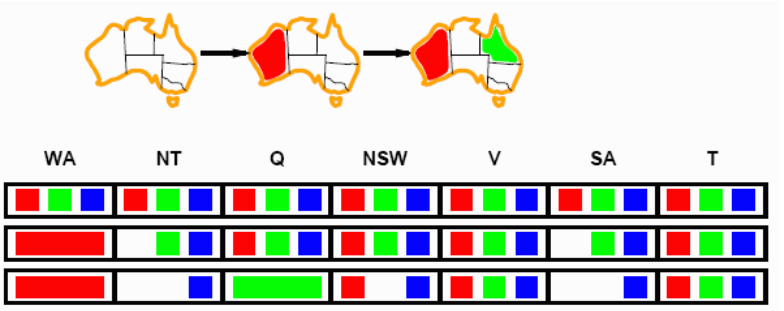

FIGURE 4.5: Constraint resolution techniques - An example with constraint propagation applied to the map-colouring problem - From [110]

These techniques are at the heart of constraint reasoning, and ones cannot speak of constraint reasoning if they did not exist. They have been extended in the form of variants that improve their behaviour for particular classes of problems. However, those variants are out of the scope of this thesis.

4.3.3 Node and arc consistency

Finally, there exist various consistency techniques to reduce the search space, by enforcing the consistency on variables and constraints. Some important properties of consistent CSPs are known as node consistency and arc consistency. Their names refer to their representations as nodes and arcs of a constraint graph. Such a graph is illustrated previously in Figure 4.2 page 59.

Node consistency A CSP is node consistent if each primitive constraint in it is node consistent. A primitive constraint c is node consistent with domain D if either $vars(c) = x$ or $|vars(c)| \neq 1|$. An algorithm enforcing consistency for a CSP can achieve it for a constraint c of the problem by removing all values from $D(x)$ that falsify c.

Arc consistency A CSP is arc consistent if each primitive constraint in it is arc consistent. A primitive constraint c between two variables x and y is arc consistent with domain D if, either $vars(c) = x, y$ or $|vars(c) \neq 2|$. An algorithm enforcing consistency can achieve it for a CSP by removing all values from $D(x)$ for which there is no value in $D(y)$ that satisfies x.

The following CSP is not node consistent (see Z):

Definition $X < Y \wedge Y < Z \wedge Z <= 2$

Domain values $D(X) = D(Y) = D(Z) = 1, 2, 3, 4$

While the following CSP is node consistent but not arc consistent:

Definition $X < Y \wedge Y < Z \wedge Z <= 2$

Domain values $D(X) = D(Y) = 1, 2, 3, 4, D(Z) = 1, 2$

Finally, the following equivalent CSP is arc consistent:

Definition $X < Y \wedge Y < Z \wedge Z <= 2$

Domain values $D(X) = D(Y) = D(Z) = \emptyset$

As well, the map colouring CSP is node consistent and arc consistent since the constraints and domain values do not allow any inconsistency.

4.3.4 Syntax of a constraint programming language

The constraints and goals of a problem can be deduced from the user's definition of the problem. Once they have been extracted from the definition, they can be expressed using a syntax supported by the constraint programming language. Basically, this language supports variables ($X, Y, Z, T_A, U_1, List, \dots$), function symbols ($+, -, \div, \times, \dots$) and relational symbols ($=, <, \leq, >, \geq, \neq, \wedge, \vee, \dots$). Those symbols are sufficient to define simple constraints. At the heart of a constraint satisfaction problem description, is to construct a good definition of a problem so that the problem is as simple as possible to be described in the form of constraints. The domain of the problem has certainly an influence on the syntax of user-defined rules. Other criteria such as the choice of variables, domain values, the rules' syntax as well as the arguments of these rules need to be defined by the user. However, the user can use primitive constraints, with a syntax similar to Horn clauses, to build his own user constraints. For example a constraint could contain a relational symbol such as $\neq$ or $+$ which requires two arguments of certain types (integer values, real values, ...). Other primitive constraints can be deduced from the available relational and function symbols previously described. Given a constraint domain D, a primitive constraint is the simplest form of constraint.

Examples of primitive constraints are :

- $X = 2Y$
- $X \geq Y + 3$

The most important operation in constraint reasoning is to find if a constraint is satisfiable. It is when there exist at least one possible valuation for which the constraint has

a solution, while an unsatisfiable constraint does not have any solution (or the solution is unknown).

Examples of satisfiable and unsatisfiable constraints :

- $X \leq 3 \wedge Y = X + 1$ is satisfiable
- $X \leq 3 \wedge Y = X + 1 \wedge Y \geq 6$ is unsatisfiable

A satisfiable constraint is either true or false depending of the valuations of its variables. For example, a constraint such as $X = Y + 2$ is either true or false depending on the valuations for X and Y. For instance it is true for ($X = 3$ and $Y = 1$). Finally, two different constraints can represent the same information : they are equivalent if they have the same set of solutions. For example, $X = 0 \wedge Y = 1$ is equivalent to $Y = 1 \wedge X = 0$.

By combining primitive constraints, complex constructions can be obtained such as $c_1 \wedge ... \wedge ... \wedge c_n$ Whose components are defined as follows:

- $\wedge$ is equivalent to the logical *and* (his counterpart is $\vee$ for *or*)
- $c_1 ... c_n$ are primitive constraints.

As for logical Horn clauses, the head of a rule is true if the body can be proven. Similarly, the above constraint $c_1, ... \wedge ..., c_n$ is true when all primitive constraints $c_1 ... c_n$ are true. In turn those more elaborated constraints can be assembled into complex conjunctions of constraints as $C_1 \wedge C_2$, which are head of rules having their body defined as follows:

Assume C_1 is defined as $c_1 \wedge ... \wedge ... \wedge c_n$
And C_2 is defined as $c'_i \wedge ... \wedge ... \wedge c'_m$
Then $C_1 \wedge C_2$ is defined as $c_1 \wedge ... \wedge ... \wedge c_n \wedge c'_i \wedge ... \wedge ... \wedge c'_m$.

4.3.5 Simplification and optimization

Those rules are used to describe and evaluate the user goals to solve. However, before evaluating the user goals, constraint solvers proceed to translate a problem in an equivalent and simplified form. Indeed, two equivalent constraints can represent the same information and can be written in many forms. An important kind of simplification consists in the elimination of redundant and useless information. For example, the simple constraint $0 \leq 3$ can be simplified to *true*. Constraint solvers use different techniques to translate the initial problem in a simpler form, until only primitive constraints are left.

Certain class of CSPs do not focus on finding a solution to the initial problem but focus on finding the best solution. Some of these problems belong to the class of

optimization problems, since they desire an optimal solution. Constraint implement algorithms such as the simplex to find optimal solutions. An optimization problem (C, f) consists in a constraint and an objective function f, where the search of an optimal solution for constraint C is guided by the goal to maximize of minimize the function f. Some optimization problems may have more than a single optimal solution. Similarly, an optimization problem may not have an optimal solution at all.

This class of problem is in turn divided into two categories depending on the type of the variables, for instance discrete or continuous. The basic goal in optimization problem is to maximize or minimize the value of particular functions, depending on the specific problem to solve. Implementations of CP languages offer generally some built-in functions such as *maximize* or *minimize* which are well-suited to address those generic optimization problems. Those functions take a set of arguments (G, E) where G is the goal and E defines the expression to maximize (respectively minimize depending on the need) and return the best solution for G. As well, many other generic functions are usually available in CPs implementations and those represent a great advantage in solving constraint problems since the user benefits from the existing generic constraint solving algorithms which are at the heart of constraint solvers, as well as the available built-in functions.

4.3.6 Usages and benefits

CP is recognized as one of the more exciting developments in the programming languages of the last decades and is becoming a method of choice for optimization problems [10]. It has been described by the ACM as a strategic direction in the field of computer search. Its utility has been proved in a large panel of real-world applications and problems. For instance, it is used in various fields such as civil or mechanical engineering, air traffic control, finance and others, with implementations dedicated to various different problems. Those classes of problems range from constraints satisfaction, planning/routing, computer-aided decision and verification tools, security analysis, consistency checking, diagnosis and configuration. Moreover, the role of CP has been illustrated in the field of biology for several use-cases such as network inconsistency detection, metabolic network analysis and analysis of the dynamics of biological system models, as reported in [111]. Reasons are directly related to the advantages of using CP. For example, developing prototypes is easy, and it is fast to build reduced versions of a final program. CP improves flexibility, maintainability and modifiability of a program with a reduced cost, due to the way constraints are encoded. Finally, the constraint solvers use efficient algorithms from mathematics, artificial intelligence and operations research, consequently the programs developed with CP have usually good performances.

As shown previously, the problems addressed by CP are in the class of CSPs (constraint satisfaction problems) and solving such class of problems often requires using a constraint solver, a program specialized in resolution of constraints. Known classes of solvers are Boolean Satisfiability solvers (SAT) or constraint satisfaction problem solvers. An example of a constraint solver dedicated to resolution of biological problems is BioASP[112], which implements methods for analysing metabolic, signalling and GRNs.

Two classes of target problems for CP can be listed, addressing the modelling of either discrete or continuous dynamics. As seen in Chapter 3, modelling of continuous dynamics can be challenging, especially when lacking precise quantitative knowledge. Modelling of discrete dynamics implies that some simplifications are made, for example the use of a finite domain of values, a finite number of states and reactions, etc. See [113] and [111] for examples. CLP is perfectly suited to address problems of finite domain values and yet largely used to modelling problems in finite-domain. The programmer can more easily model a constraint's problem when the set of possible assignable values is limited, at the opposite of problems involving continuous domain values. In addition to being an advantage for the programmer, real problems are also simpler to solve when the decisions they involve can be replaced by the tasks of choosing the value to assign among a limited range of integers.

4.3.7 Complexity

The complexity of constraint problem solving is correlated with the number of variables and possible values, i.e., the amount of possible states for the problem representation. Reducing this amount also simplify the writing of rules, and consequently reduces the amount of work performed by the constraint solver. There exist similarities between CLP and other programming paradigms. First, CP requires enough practice from the programmer, since this is a different programming paradigm. Second, there are various ways to model the same problem and some strategies are required for an efficient modelling. Choices made by the programmer, such as the set of user-defined constraints and the set of variables, have a direct impact on the program efficiency. With enough practice, CLP has a clear advantage for solving optimization problems or other problems of combinatorial complexity. Finally, the reader interested in CLP and solving of CSPs can also refer to [9] and [114][chap. 5]. In addition, [10] is a good introduction to CP techniques used for the modelling of CSP and makes and interesting comparison between procedural and constraint programming paradigms for the modelling of similar problems.

4.4 Prolog

Prolog is a logic programming language developed in 1972, commonly used in field of artificial intelligence and computational linguistics. It was one of the first logic programming languages, along with Lisp (1958). It is used today for a large class of applications, such as expert systems and theorem proving, but it was first intended to be used for natural language processing. There exist other programming languages inspired at least partially by Prolog, such as Logtalk[115][1] (1998) or Mercury[116][2]. Since its creation, Prolog remains the most popular among all the logic programming languages. There exist multiple implementations of Prolog. Examples of major implementations are SWI-Prolog[3] (created in 1987), B-Prolog[4] (1994), ECLiPSe[5] (1992), GNU Prolog[6] (1999) or SICStus Prolog[117] (1988). Among those, some implementations provide a constraint solver with the support of CLP, for instance B-Prolog and SWI-Prolog. B-Prolog and SWI-Prolog have been considered in the context of developing BioNet in Chapter 5. They are free and they both provide a good support for CLP over finite domains. Some past benchmarks[118] report an advantage in performance for CLP B-Prolog while SWI-Prolog is often referenced for its maturity (it has been developed in 1987), popularity and user-friendliness (thanks to its graphical user interface). In addition to its support for CLP, SWI-Prolog supports also many other libraries and has additional capabilities such as the support of web programming. Both SWI-Prolog and B-Prolog seem to offer interesting features for developing solutions to CSPs. However, for the development of BioNet in 5, SWI-Prolog has been chosen for its maturity, its natural ability to support of rule-based logic programming through its library clp(fd) dedicated to the CLP over finite domains.

4.4.1 Usages

The literature refer to several usages of Prolog for analysing GRNs. Examples are [119] in the context of analysing the HIV-Host protein interaction network, with a motif detection algorithm implemented in Prolog and used to search for network motifs. Another example is given in [120] that describes an algorithm for determining the structure and parameters of Bayesian networks inferred from experimental data. It used the PRISM framework [121], a probabilistic logical framework based on B-prolog that in turn is a high-performance implementation of the Prolog language. In this work, the graph

[1] http://logtalk.org/
[2] http://mercurylang.org/
[3] http://www.swi-prolog.org/
[4] http://www.picat-lang.org/bprolog/index.html
[5] http://eclipseclp.org/
[6] http://www.gprolog.org/

visualization software GraphViz[7] was also used to display the networks reconstructed from real data.

4.4.2 CLP(FD)

The constraint logic programming over finite domains (CLP(FD))[122] is an extension of Prolog, also proposed as a library in SWI-Prolog. Often used to model and solve combinatorial problems such as routing, planning, scheduling and allocation tasks. It is only one of the constraint solvers available in SWI-Prolog. Others are clp(qr), clp(b) for handling respectively rational/real numbers and booleans. To activate the usage of CLP(FD) in a Prolog program compatible with the SWI-Prolog implementation, one needs simply to invoke the following command before editing/consulting a program with constraints :

```
1        use_module(library(clpfd)).
```

With respect to the CLP paradigm, and the Prolog syntax, CLP(FD) allow constraining variables to a finite domain, by using essentially arithmetic and logical operators. Introduction and examples for using this library can be found in [123, 124], as well as advanced topics related to CLP(FD).

Due to its CLP capabilities, CLP(FD) brings a lot of features compared to Prolog. Compared to Prolog, CLP(FD) is able to find solutions with a smaller amount of searches and effort for constrained problems. Additionally, it is able to find solutions for problems that Prolog cannot resolve. For example, faced to a problem such as : $X - 3 = Y + 5$, Prolog is simply stuck :

```
1 ?- X - 3 = Y + 5.
2 false.
```

Prolog answers with $false$ (no solution), while CLP(FD) answers with a solution :

```
1 ?- X - 3 #= Y + 5.
2 8+Y #= X.
```

For instance, CLP(FD) determines that $X = 8 + Y$.

[7] http://www.graphviz.org/

4.5 From problem description to the solution

Remember the previous example of map-colouring problem of section 4.3 page 57. The goal was to find an optimal assignment of colours to regions, with a finite set of possible colours as well as a finite set of possible regions. It is very similar to the goal of a CSP with finite-domain values, except that the goal is here to find an optimal assignment for a finite set of variables, with a finite set of possible values. In CLP, the process of solving a problem requires first capturing and modelling an idealised view of this problem. This simplification approach is also used for other classes of problems addressed by CLP. The task of constraint solving for finite-domain values is usually performed by a *labelling* process. It is a built-in function of a constraint solver which takes finite-domain variables as arguments and uses constraint solving techniques to find a valuation for all these variables, resulting eventually in one or more satisfiable solution(s).

At the opposite of traditional programs, where assignments of all variables are managed by the programmer, those assignments are performed themselves by the constraint solver. This make part of the high level nature of CLP languages. As a consequence, modelling a simple constraint problem requires from the programmer to adapt himself to this class of problem modelling which highly differs from the classical ones. The programmer has to translate the high-level problem description to high-level user-defined constraints constructed from primitives constraints. In addition, he programmer will have to determine the proper set of variables and values to keep the modelling as simple as possible, along with the goal to reduce as much as possible the amount of those variables. Those variables can represent known or unknown elements of the problem, in the second case variables imply an assignment is expected from the constraint solver.

4.5.1 Modelling with constraints

The first step of the modelling with constraints is to define the domain of values as well as the variables that will contain the values of the problem and its solution. Those variables can be defined independently or as a list of values. In the case of the map-colouring problem, the list will be composed of the distinct regions to colour.

We could end with the following definition :

Constraints $WA \neq NT \wedge WA \neq SA \wedge NT \neq SA \wedge NT \neq Q \wedge SA \neq Q \wedge SA \neq NSW \wedge SA \neq V \wedge Q \neq NSW \wedge NSW \neq V$

Domain of values $D(WA) = D(NT) = D(SA) = D(Q) = D(NSW) = D(V) = D(V) = D(T) = red, yellow, blue.$

Those regions will be valued with integers since the problem will be modelled for the finite domain, having each region mapped with a distinct integer. To implement this CSP definition in the equivalent CLP(FD) program, one needs to proceed to a translation. A CLP(FD) program is usually made of three parts, referred to as variable generation, constraint generation and labelling, described as follows :

Variable generation This first part generates the variables and specifies their domain values.

Constraint generation The second part generates the constraints over the variables.

Labelling This last part instantiates the variables by enumerating the values.

4.5.2 Translation and resolution in Prolog

The following Prolog program specifies the map-colouring problem, based on the previous definition :

```
1  solve(Regions) :-
2      % variable generation
3      Regions = [WA, NT, SA, Q, NSW, V, T],
4      Regions ins 1..3,
5
6      % constraint generation
7      WA #\= NT, WA #\= SA,
8      NT #\= SA, NT #\= Q,
9      SA #\= Q,  SA #\= NSW, SA #\= V,
10     Q #\= NSW,
11     NSW #\= V,
12
13     % labelling
14     label(Regions).
```

Then the call to the “solve” predicate results as follows :

```
1  ?- solve([WA, NT, SA, Q, NSW, V, T]).
2  WA = 1,
```

```
3 NT = 2,
4 SA = 3,
5 Q = 1,
6 NSW = 2,
7 V = T, T = 1.
```

Where the integers can be mapped to the distinct colours.

4.6 Application to modelling gene regulatory networks

As CLP is used to solve many real world problems, it applies as well to solving questions related to understanding of GRN dynamics. The most complex tasks related to the modelling of GRN are the reconstruction of the network itself and the elucidation of its dynamics, for instance, the identification of genes as well as their interactions. Other recurring tasks are the validation of a model, as seen previously with the introduction the model-checkers. Those tasks could be described as CSPs for finite domains. For example, the problem of identifying the genes and interactions is similar to the problem of identifying the values of unknown variables, where variables can represent arcs and nodes in a network. Additionally, the tasks performed by model-checkers are similar to finding a solution to a constraint solving problem : a model can be considered as valid when all the constraints are satisfied.

Once these problems are modelled in the form of CSPs, many questions could be implicitly solved. For instance, the wrong model parametrizations could be identified and thus avoided. Similarly, the invalid network constructions can be eliminated to keep only those which satisfy the inputs of the problem. Since the biology researcher can have some knowledge or intuitions on the dynamics of a biological system, ones can imagine those hypotheses to be modelled in the form of user-defined constraints. Depending on the problem to solve and the studied biological system, the number of parameters of the CSP could vary, as well as the hypotheses. However, the solving strategies are not different. The task is usually the reconstruction of a GRN from a problem definition. This definition can take into account the experimental data extracted from Microarray analysis.

However, solving a CSP efficiently require the ability to constraint as much as possible the search space of solutions. If this search space is not reduced, we end with a problem known as the problem state explosion, faced by model-checkers when the amount of variables in the system keeps growing. For instance, this state explosion problem can be avoided in CLP approaches, by defining user-constraints on the gene expression levels,

the strengths (weights) of interactions, or by defining hypotheses on supposed existing nodes and interactions. As seen in the constraint propagation technique, existing constraints help to reduce the search space of solutions for other constraints. Remember Thomas' formalism described in Chapter 3 page 43, where a discrete model is used to describe a biological system. In this context, as well as in the context of other discrete models, CP can be useful. For instance, questions related to the understanding of regulation processes can be addressed using constraint solving techniques. Constraints can define the regulation dynamics, in a way they would help to find a valuation for the rate of synthesis of gene products as well as identifying the nature of interactions between two components of the networks. In addition, constraint solving techniques as those studied in this chapter can help to elucidate questions related to protein synthesis, gene/protein interactions or find the right parametrizations of a biological system model regarding the problem inputs.

In the literature, advanced uses cases involving the use of CP for studying GRN can be found. In [125] Corblin et al. suggests an interesting method for analysing discrete GRNs with model-checking, which can be also applied with CP. First, it consists in finding the initial constraints for a model. Second, once these constraints have been tested on a model, some other rules can be deduced. Then, if no model can be proved to be consistent with those constraints, a few ones are relaxed, until the overall consistency can be proven. As a consequence, a valid model is generated. When a model is eligible, the next step is finding (infer) predictable properties of the model. If a property is found to be true for many models, it is tested in experiments and then retained as a new constraint for further selection of valid models. The last proposed step of this method implies a restart of the model analysis. First, some constraints are removed from the set of the previous analysis. Second, is performed a new analysis with this reduced set of constraints which help to see which properties are conserved among various subsets and/or models. If some properties are lost, some constraints can then be restored. Another proposed improvement is to find which properties are conserved among those various subsets and/or models and add them as new constraints.

A way to elucidate questions related to a biological can be to use CLPs to develop tool with more user friendliness for the biology researcher. Since the constraints solving techniques are generic and already implemented, they can be used even if the user is not an expert in CP. Rather than limiting their scope to model checking or elucidations of parameters, such a tool can be useful for suggesting a valid network construction, with the purpose to allow the user to visualize it immediately. The GRN could be displayed as a graph, such as those studied in previous chapters. In this context, the user would be able to interact with the model-checker until the network and the models would seem right. Based on the graph, the user could confirm or reject his initial assumptions on the

model and its parameters and the tool could even help to find new candidate parameters and user-defined constraints to apply to the current model for further experiments. In the Chapter 5 page 75, those ideas are used as a base for the design of a new support tool for GRNs.

An issue with solving CSPs is that a problem can have more than one solution. Indeed, if a problem is under-constrained, multiple valuations can satisfy the initial constraints. But it is also a clue that some additional constraints are required in order to restrain the amount of solutions. Depending on the need of choosing between multiple alternatives, the possibility of having multiple proposed solutions can be advantageous.

4.7 Conclusion

This chapter described how CP techniques can be used for solving real world problems as well as biological problems. CLP has a real advantage compared to other programming paradigms since it solves problems without the need to write specific step-by-step instructions for solving a problem. This is the reason why this programming paradigm is used in various fields such as civil or mechanical engineering, air traffic control, finance as well as many others. Targets for using CLP and constraint solving techniques are often optimization problems and consistency checking. Examples are constraints satisfaction, scheduling/routing, configuration optimization as well as computer-aided decision tools such as guided problem diagnosis. Many real worlds problems can be modelled as CSPs. In this chapter the emphasis was set on resolution of CSPs for finite domains. However, there exist other use cases of CLPs which are not covered in this thesis. For instance the usage of CLPs for domains of boolean or continuous values. It has been shown that CLP is well-suited to address problems involving inference of parameters. Examples are the map-colouring problem and the reconstruction of a GRN where the elucidation of the best solution implies defining user-constraints, dealing with unknown parameters and electing the best assignment options. In the context of modelling the dynamics of GRNs which can be complex to model, CLP can be very beneficial. It combines the advantages of logical reasoning and the ability to deal with the uncertainty and constraints of complex problems. However, it requires learning a new programming paradigm and new way of solving problems, which can be a good investment given the advantages, but may be dissuading in the community of biology researchers.

Chapter 5

Design of a support tool for gene regulatory networks

5.1 Introduction

This chapter presents a new tool, BioNet, developed along with the writing of this thesis. This is essentially a prototype aiming of helping the biologist researcher at his work on regulatory networks. It uses CLP over finite domains for the solving of the dynamics of a GRN. Starting from a set of user requirements, we describe the model and user interface of BioNet, as well as the process used for the modelling of a gene regulatory system in BioNet. At the same time, we describe the designed features. To illustrate the features, we proceed to the modelling of gene regulation in the *lac* operon. Then, we explain the tools and technologies chosen for building BioNet and we describe its implementation. Finally, the chapter describes the advantages of BioNet, and some clues are proposed for its further improvement.

5.2 User requirements

The goal of BioNet is to expect the least technical knowledge from the end-user, and let him use the application via a simple user interface. The tool should produce results, such as an inferred network, based on a specification provided by the user. The user requirements of a such tool can be described as follows :

1. Since BioNet has the aiming of helping the biologist researcher to reconstruct a GRN, the tool should at least help the user to verify the results. A way to achieve

that goal is to let the user visualize the network topology as well as the interactions between components.

2. Based on the user inputs and the provided experimental data, the tool can simulate additional experiments and generate experimental data. Then, the user should be able to visualize those experimental data, and correct his inputs consequently.
3. A biologist researcher has often to deal with uncertain and incomplete information over the regulatory interactions, i.e., the nature of interactions between components of the network. The tool should let the user to encode verified information as well as unverified information. The verified information consists of real experimental data and definition of regulatory interactions, while unverified information consists of hypotheses on those interactions. The tool can produce results from all these informations, then the obtained results should help the user adapt his inputs consequently for next simulations.
4. The specification of user inputs should be a process as simple as possible, so that they can be modified easily and the user could easier verify new assumptions.
5. The modifications of user input should be taken into account as soon as possible, so that the user can quicker verify new assumptions.
6. Finally, the tool must help the user to adapt his inputs by proposing him some clues and by inferring some properties from the results. For example, the tool could propose to add some facts or hypothesis in the user inputs, based on the previous obtained results.

5.2.1 Model

From a problem description, the tool must proceed to the reconstruction of the network and the inference of unknown parameters, with respect to the user inputs. Based on the user requirements, we can infer the need of the following inputs to model a problem :

Gene expression data The complete set of gene expression levels measured through experiments (via Microarray technologies). These data can be used for the reconstruction of the network and consequently they should be as reliable and precise as possible. Typically, the gene expression level of a gene can be defined as a triplet. It consists of the gene identifier, the experiment identifier, and the expression value.

Regulation rules Some rules over the regulation dynamics can be specified by the user. For example a particular rule can define that when two specific genes a and

b are expressed, a certain gene c should be inhibited/activated. As well, the rate of inhibition/activation for this regulation dynamic can be specified.

User hypotheses Some hypotheses can be specified over the importance (weight) of some interactions (relationships). For example, if we suppose the gene a has some influence on gene b, we can define this as a hypothesis, and we can specify the supposed importance of this influence. Then the tool can use this information to compute the potential weights and deduce the expression values that could be expected in future experiments.

Configuration Some additional parameters can be specified to constrain the solution, such as the minimum/maximum weights of interactions, the minimum/maximum node expression levels, the default rate of inhibition of some genes in the regulation processes as well as any useful parameter that can differ from a modelling to another.

An example of encoded user input is illustrated in Figure 5.1 page 78.

5.2.2 User interface

We propose in BioNet a user interface quite simple, and even minimalist, based on the user requirements described in section 5.2 page 75 and the user inputs described in section 5.2.1 page 76. It consists in a webpage letting the user entering a problem description, visualize the inferred GRN and the generated gene expression data. In addition, some rules proposed by the tool may appear in the main page.

A advanced editor is proposed to let the user enter a problem description in addition to experimental data, or adapt the BioNet configuration :

The Network visualizer displays the network inferred from the user inputs and configuration. An example is illustrated in Figure 5.2 page 79.

Finally, the gene expression levels computed by BioNet are displayed in a chart. It shows the values of gene expression levels through the known experiments. An example is illustrated in Figure 5.3 page 80.

5.3 Modelling process in BioNet

The following subsections describe in details how the user inputs described in section 5.2 are modelled in BioNet.

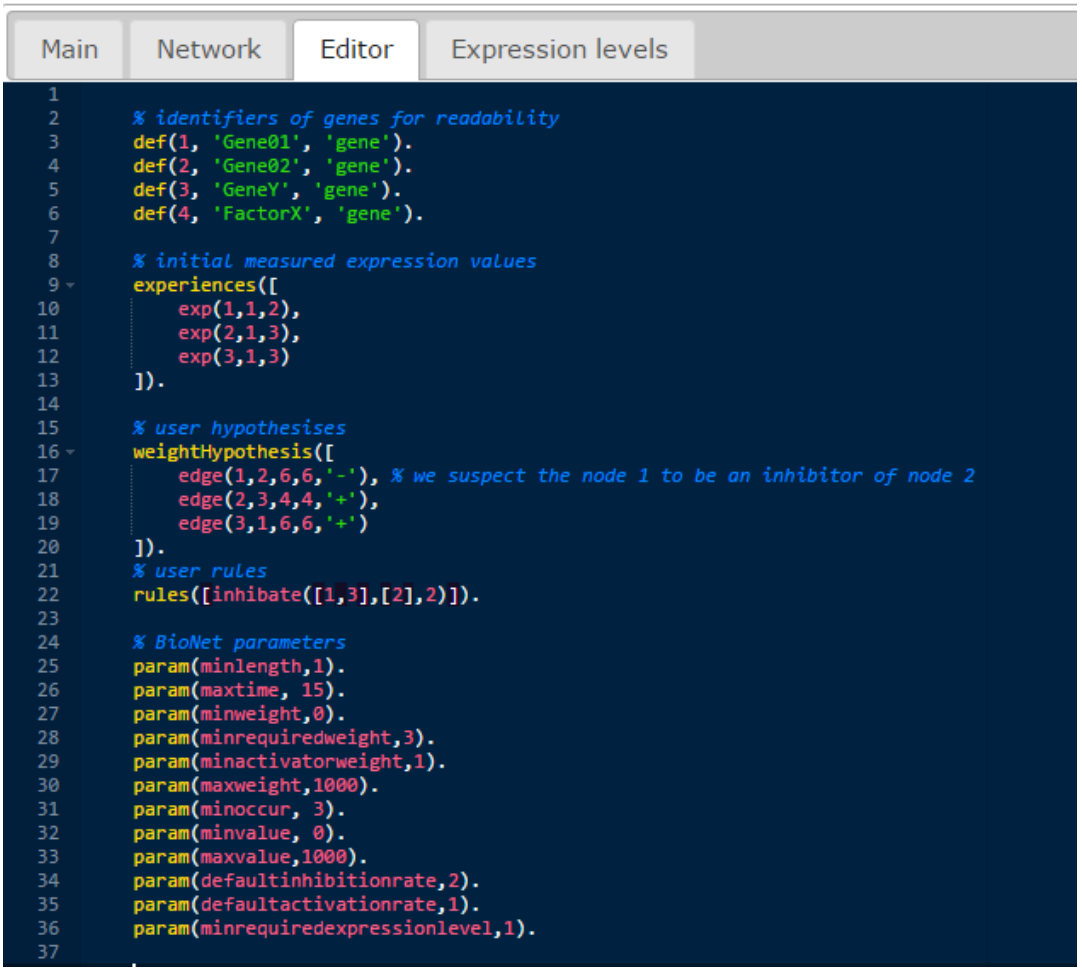

FIGURE 5.1: BioNet - User inputs and configuration

5.3.1 Gene expression data

BioNet infers a network from the given expression data provided by the user. Since CLP(FD) works with integers, the values of gene expression levels need to be in the form of integer values. However, most of the time, the gene expression datasets available in the literature and existing databases provide values in the domain of continuous values. As a consequence, those data need to be normalized before being used in BioNet or any other tool that works only with integer values. This issue can be solved by using various existing techniques from the fields of statistics, as seen in the previous chapter 3. There exist numerous techniques with arbitrary parametrizations and different results, especially for the elimination of inherent noise. It is also complex to determine what is the best technique to apply. Consequently, rather than normalizing the gene expression data with a specific technique, BioNet lets the user choosing his normalization tool and performing by his side. As a consequence, no such technique is applied in BioNet on the inputs. Those data should be normalized and given to BioNet in the form of integer values. Fortunately, BioNet let the user specifying the minimum and maximum

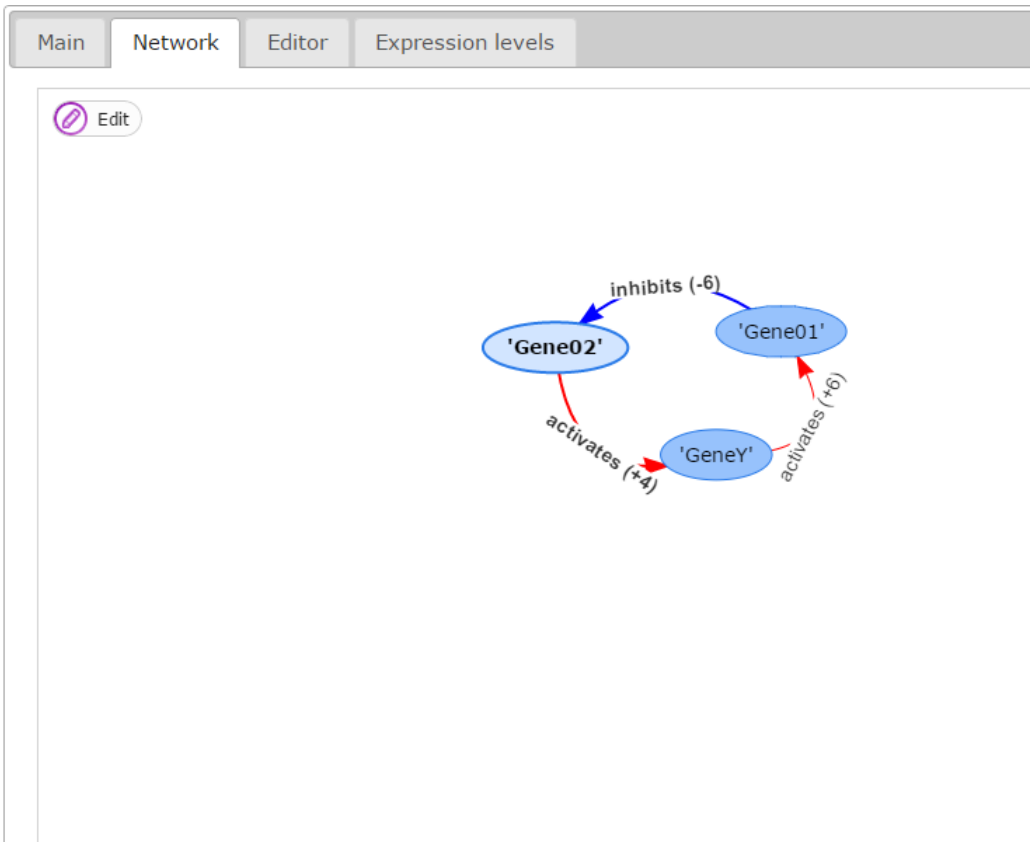

FIGURE 5.2: BioNet - Visualization of a reconstructed network

bounds of values in the BioNet' parameters, which leaves for the user a certain room for manoeuvre to constrain the number of different possible accepted input values.

In BioNet, the expression level of a gene $x_i \in X$ at time $j \in T$, also described as $v_i(t_j)$, can be described as $exp(x_i, j, v_i(t_j))$ or, in Prolog syntax :

```
1 exp(1,2,3)
```

It describes that the expression level of gene 1 at time 2 is equal to 3. Here, the node is identified by a number since CLP(FD) work with integers.

While the expression levels bounds are some independent parameters and can be specified as the following :

```
1 param(minvalue, 0)
2 param(maxvalue, 1000)
```

The full set of supported parameters is described in subsection 5.3.5 page 82.

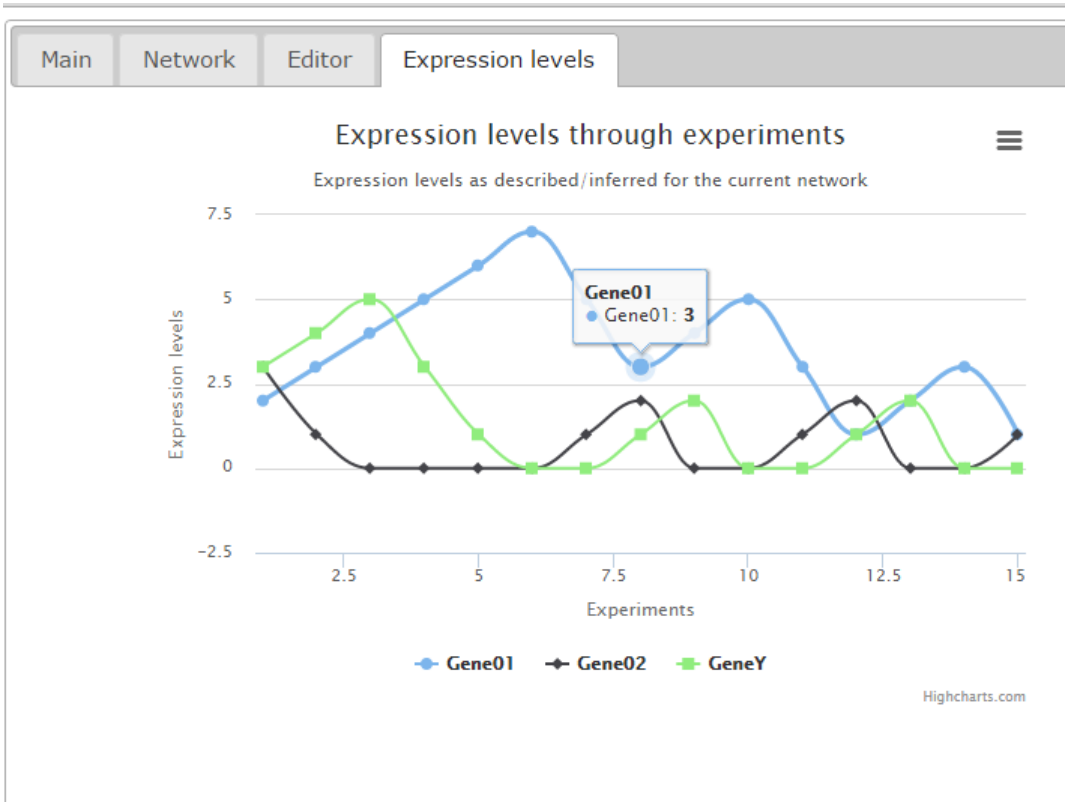

FIGURE 5.3: BioNet - Visualization of inferred gene expression data

5.3.2 User hypotheses

As described previously, the user has the ability to specify some hypotheses over the importance/weight of the relationships between nodes. For example, we can define that a node a has an influence on a node b and that this influence has a weight supposed between 2 and 4. This hypothesis can be defined as $edge(a, b, 2, 4), a, b \in X$, while it can be defined in Prolog syntax as follows :

```
1 edge(1,3,2,4)
```

Where the numbers 1 and 3 serve to identify the nodes by their numerical identifier.

5.3.3 Regulation rules

The user has the ability to enforce some rules in BioNet. Those rules describe dynamics that could not be described by simple hypotheses. Those rules can be defined in Prolog syntax as follows :

```
1 activate([1,2],[3],1)
```

This rule defines that if both nodes 1 and 2 are expressed (node expression level > 0) in an experiment at time t, it will increase the expression level of the node 3 in the next experiment at time $t + 1$. In other words, the expression level of node x_i is defined by $v_i(t_{j+1}) >= v_i(t_j) + k, j \in T, x_i \in X$ where k denotes the rate of activation.

The same logic can apply for the following rule, whereas the simultaneous expression of nodes 1 and 3 in expression values measured at time t decreases the level of targeted node 2 in the expression values measured at time $t + 1$.

```
1 inhibate([1,3],[2],2)
```

In other words, the expression level of node x_i is defined by $v_i(t_{j+1}) <= v_i(t_j) - k, j \in T, x_i \in X$ where k denotes the rate of inhibition.

Finally, the following rules defines that a node x_i can activate himself in the current experiment when its expression value is equal to 0 in the previous experiment.

```
1 autoactivate([1],[1],4)
```

5.3.4 Resolution with CLP(FD)

First, the user inputs and the BioNet configuration are encoded by the user, and then they are read and checked by BioNet before the resolution process. Then, BioNet initiates to the resolution itself. During this process, BioNet attempts to identify first the suspected relationships (the existing edges between nodes). Then, he identifies the types (activation, inhibition) and the importance (weight) of any of those relationships. Third, BioNet computes the possible gene expression data of future experiments. BioNet is able to generate gene expression data for as many experiments as asked by the user. Finally, BioNet identifies some additional rules that could be interesting for future experiments. Practically, BioNet proposes a rule only if this rule can explain a recurrent phenomenon that can be observed in the given or generated gene expression data. For example, if a gene seems to be inhibited as soon as two other genes are expressed, and if this property can be verified by BioNet among multiple experiments, BioNet may propose a rule as follows :

```
1 inhibate([1,3],[2],2)
```

Where 1 and 3 identify the inhibitors, and 2 identifies the inhibited gene.

The resolution ends as soon as BioNet has found a satisfiable solution for those parameters. At the end, the content of this resolution consists of :

Inferred network The reconstructed GRN as proposed by BioNet. It consists in the set of relationships (edges) between the nodes and the properties of those relationships. An edge, which defines a relationship between two nodes, is associated with a weight and the nature of the relationship (activation, inhibition, auto-activation). Some relationships can be suggested by BioNet, while other relationships are simply taken from the hypotheses of relationships as encoded by the user in the problem input.

Gene expression data The combination of the initial expression data and the generated expression data if asked by the user. It consists in a set of experiments, and in turn each experiment consists in a set of expression levels for distinct genes. The values of those expression levels are comprised between the intervals specified by the user in the BioNet configuration (see subsection 5.3.5 page 82).

Suggested rules It consists in a set of rules proposed to the user by BioNet. Those rules define some properties which have been verified a certain amount of times in the experiments.

The full resolution process is performed by SWI-Prolog by using the CLP(FD) library. As a consequence, the resolution process can be called from the SWI-Prolog command-line or even from the SWI-Prolog user interface. As well, the user inputs and the result of a resolution are in the syntax of Prolog. Examples of such inputs and outputs can be shown in appendix A section A.1 page 105.

5.3.5 Configuration

The values of BioNet configuration can be specified along with the user inputs. This configuration has a significant impact on the network reconstruction, the form of generated expression data as well as the set of possible values for the inferred weights of relationships. The following list defines the set of parameters taken into account by BioNet :

maxtime defines the number of experiments which need to be used or even generated. BioNet is able to generate the missing experiment values if needed, based on the provided expression values, the user hypotheses and the user rules. BioNet is able to generate as many expression data as allowed by this parameter. So, if the user provides the expression values for 5 experiments, but needs a simulation of the expression values of five more experiments, it can set this parameter to 10. In BioNet, an experiment can be defined as the set of all expression values measured at a specific time $t \in T$.

minweight the minimum possible value allowed for the weight of a relationship. It also defines the minimum value that can be used for the inference of unknown relationship weights.

maxweight the maximum possible value allowed for the weight of a relationship. It also defines the maximum value that can be used for the inference of unknown relationship weights.

minvalue the minimum possible value allowed for the expression level of a node at any time. It also defines the minimum value that can be used for the inference of unknown expression levels.

maxvalue the maximum possible value allowed for the expression level of a node at any time. It also defines the maximum value that can be used for the inference of unknown expression levels.

minrequiredweight the minimum possible value taken into account for the weight of relationship in BioNet. The lowest values will be ignored during the process of network inference as well as for computing gene expression levels.

minactivatorweight this parameter defines the minimum required weight of a relationship to be taken into account.

minrequiredexpressionlevel This parameter defines the minimum required expression level for a node to be considered as expressed.

minoccur this parameter is used by BioNet to suggest only rules that could be verified as true at least n times where n is the parameter value.

weightsfromrules This parameter, when activated, forces BioNet to infer weights on relationships from the set of user-defined rules. Indeed, an user-defined rule specifies also a weight, which is this context can be used. It is useful when no weight hypothesis is defined by the user, and when the amount of provided experimental data is not sufficient for BioNet to infer weights on relationships. Allowed values are : $\{0, 1\}$.

defaultinhibitionrate if asked, the gene expression level of a node can naturally decrease each time the node is not activated by other nodes. This parameter defines the rate of this natural inhibition.

defaultactivationrate if asked, the gene expression level of a node can naturally increase each time the node is activated by other nodes. This parameter defines the rate of this natural inhibition.

The configuration of BioNet must be encoded by the user in the syntax of the Prolog language. An example of configuration in Prolog syntax can be found in appendix A section A.3 page 107.

5.4 Modelling of the *lac* operon in BioNet

To illustrate some capabilities of the current tool, we proceed to the modelling of gene regulation in the *lac* operon, already described in Chapter 2 subsection 2.11.3 page 24. The modelling is inspired by [74] where Franke et al. proceeded to the modelling of GRN by using Thomas' formalism.

A necessary step of the modelling is to proceed to identify the important components and their relationships. The important elements in this regulatory system are described below:

1. **lacZYA mRNA** : the rate of transcription of the three structural genes to mRNA.
2. **lacI** : expression level of the lac repressor lacI.
3. **lacY, lacZ** : the concentration of lacY and lacZ products (the enzymes).
4. **lactoseInt** : lactose present in the cell.
5. **lactoseExt** : lactose present in the medium of the cell.
6. **allolactose** : allolactose present in the cell.
7. **cAMP** : the activity of cAMP complex.
8. **lac operon** : considered as active when lacZ and lacY are activated.
9. **lacZYA** : represents the state of the genes lacZ, lacY and lacA.

Then, the most important interactions are summarized as follows :

1. The translation of lacZYA to mRNA and then to proteins results in the production of three enzymes : LacY permease, LacZ β-galactosidase and LacA thiogalactoside.
2. the lacZYA mRNA is only produced under the condition when LacI is inactive, not bound to the lac operator lacO, the three structural genes of *lac* operon are present and CAP binds.
3. when cAMP is bound to CAP, they form the CAP-cAMP complex, which is able to initiates the transcription of genes to mRNA.
4. The LacY permease enables the entry of the lactose from the medium into the cell.
5. Allolactose is produced as a by-product of lactose metabolism if lactose is present in the cell and if β-galactosidase LacZ is expressed.
6. The repressor LacI is activated when allolactose is present and bound to it.

The important elements and relationships of the GRN will be specified in the user inputs of BioNet, with respect to the syntax presented in section 5.3 page 77. In order to simulate the dynamics of the digest of lactose, our scenario consider the lactose as present in the medium of the cell from the beginning. Similarly, the three structural genes of the *lac* operon as well as the cAMP are considered as active and expressed from the start. At the opposite, the other components will be unexpressed from the start.

Regarding the dynamics of the network, we can expect the concentration of lactose in the medium of the cell to decrease while the enzyme produced by the gene lacY is present. We did not provided any additional data to BioNet, only the initial expression levels are encoded. They represent the expression levels at time $t = 1$. For our scenario, we ask BioNet to compute the data of 15 experiments. BioNet is able to compute the gene expression levels of experiments $2-15$ based on the experimental data, the relationships, the weights of those relationships as well as the rules and other parameters encoded by the user. To keep the *lac* operon active during those experiments, we artificially ask BioNet to increase the concentration of lactose after a while, for instance around the 10th experiment, as specified in the user inputs.

Finally, some arbitrary parameters have been determined for the current scenario. They are be illustrated in Figure 5.4 page 86. In addition, the code of those parameters is available in appendix A, section A.5 page 119.

Based on the provided parameters, BioNet infers the network shown in Figure 5.5 page 87.

BioNet displays a chart of the node expression levels through the successive experiments. Since only some experimental data are provided at the beginning (for instance, at $t = 1$), BioNet inferred the rest of experimental data. The result is illustrated in Figure 5.6 page 87. As expected, since the enzyme produced by the gene LacY is active, it enables the entry of lactose from the medium into the cell. Consequently, the concentration of lactose in the cell increases while the concentration of lactose in the medium of the cell decreases. After a while, the concentration of lactose in the medium of the cell go back to its maximum level of expression, as specified in the user inputs.

Finally, based on the observations inferred from experimental data, BioNet may suggest some rules. For instance, a rule is suggested as seen in Figure 5.7 page 88.

This example permits observing some capabilities of BioNet, for instance in the context of modelling the *lac* operon in *Escherichia coli*. From a small dataset, BioNet has proceeded to the simulation of a small regulation system for a particular scenario. As expected, the rate of lactose in the cell has increased. However, this is a simple scenario. The modifiability of BioNet parameters offers many modelling and simulations

Main | Network | Editor | Expression levels

```
def(1, 'lacZYA mRNA'). def(2, 'LacI'). def(3, 'LacZ').
def(4, 'LacY'). def(5, 'lactoseInt'). def(6, 'lactoseExt').
def(7, 'allolactose'). def(8, 'cAMP'). def(9, 'lac operon').
def(10, 'lacZYA').
experiences([exp(1,1,0), % lacZYA mRNA
    exp(2,1,0), % lacI
    exp(3,1,0), % lacZ
    exp(4,1,0), % lacY
    exp(5,1,0), % lactoseInt
    exp(6,1,20), % lactoseExt,
    exp(6,10,20), % lactoseExt
    exp(7,1,0), % allolactose
    exp(8,1,15), % cAMP,
    exp(8,10,15), % cAMP
    exp(9,1,0), % lac operon
    exp(10,1,15) % lacZYA
]).
weightHypothesis([]).
rules([ activate([1],   [3,4],  1), % lacZYA mRNA -> ++ lacZ, lacY
    inhibate([2],   [1],    3), % lacI -> -- lacZYA mRNA
    activate([8,10],[1],    1), % cAMP, lacZYA -> ++ lacZYA mRNA
    activate([6,4], [5],    1), % lactoseExt, lacY -> ++ lactoseInt (simplified view)
    inhibate([6,4], [6],    1), % lactoseExt, lacY -> -- lactoseInt (simplified view)
    activate([5,3], [7],    1), % lactoseInt, lacZ -> ++ allolactose
    activate([7],   [2],    1), % allolactose -> ++ lacI,
    activate([3,4], [9],    1) % lacZ, lacY -> ++ lac operon
]).
param(minlength,1).
param(maxtime, 15).
param(minweight,0).
param(minrequiredweight,1).
param(minactivatorweight,1).
param(maxweight,4).
param(minoccur, 3).
param(weightsfromrules, 1).
param(minvalue, 0).
param(maxvalue, 20).
param(defaultinhibitionrate,0).
param(defaultactivationrate,1).
param(minrequiredexpressionlevel,1).
```

FIGURE 5.4: BioNet - Parameters for the modelling of the lac operon

possibilities, for various scenarios. Indeed, there exist numerous parameters combinations, and the user is free to adapt them as much as he wishes, depending on the goals. Consequently, the results of a modelling must be taken with caution.

5.5 Architecture

As described in section 5.2 page 75, the goal was to expect the least technical knowledge from the end-user, to let him use the application, interact with it through a simple user interface, and access to the application via a web browser. So, the emphasis has been to let the end-user interact with the application without installing anything on his computer excepted a browser and without being aware of the existence of the technical details. The technologies and tools retained allow to achieve the construction of BioNet with respect to the user requirements. The result is the architecture shown in Figure 5.8 page 88, and its components are described below.

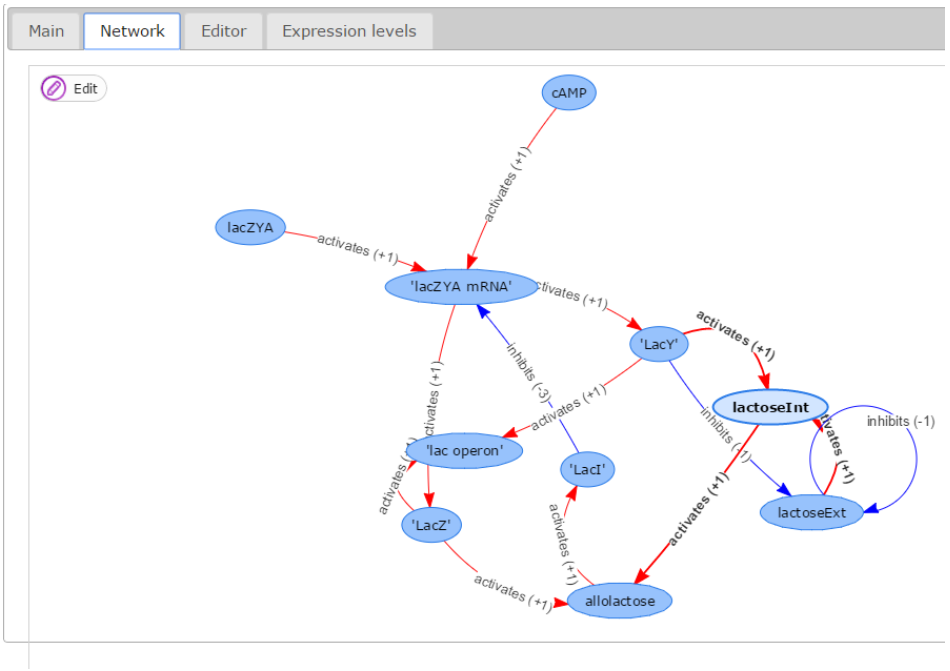

FIGURE 5.5: BioNet - Inferred network for the Lac operon

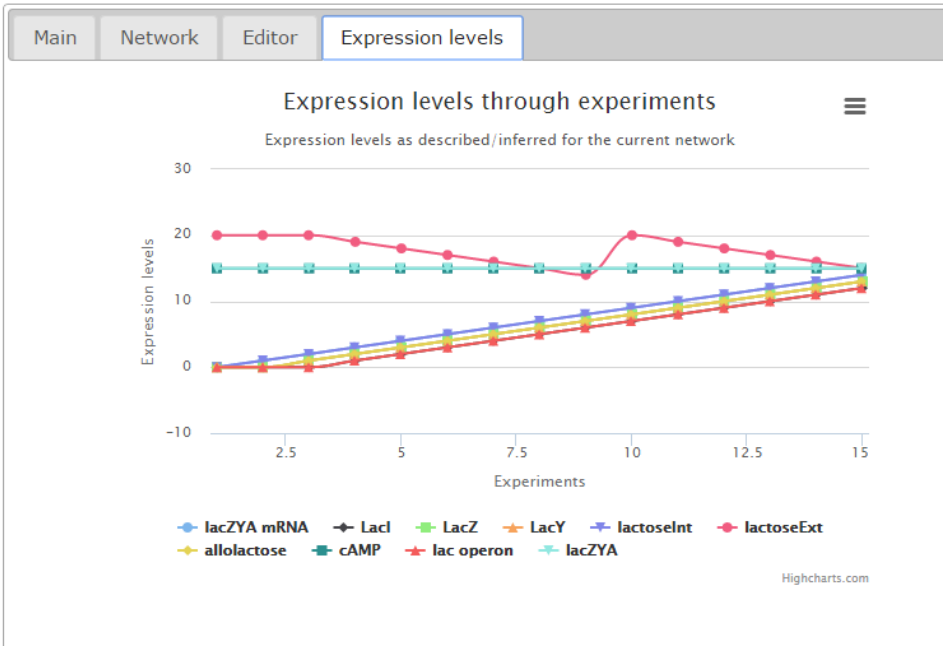

FIGURE 5.6: BioNet - experimental data measured

5.5.1 Prolog

The kernel of SWI-Prolog is used to support execution of the models written in Prolog and in addition provides the capacity of constraint solving to resolve user queries on the model. The SWI-Prolog kernel plays a central position in this architecture, since all the user inputs have to be translated into Prolog syntax language, in the form of logical constraints and predicates. Those constraints and predicates will be internally

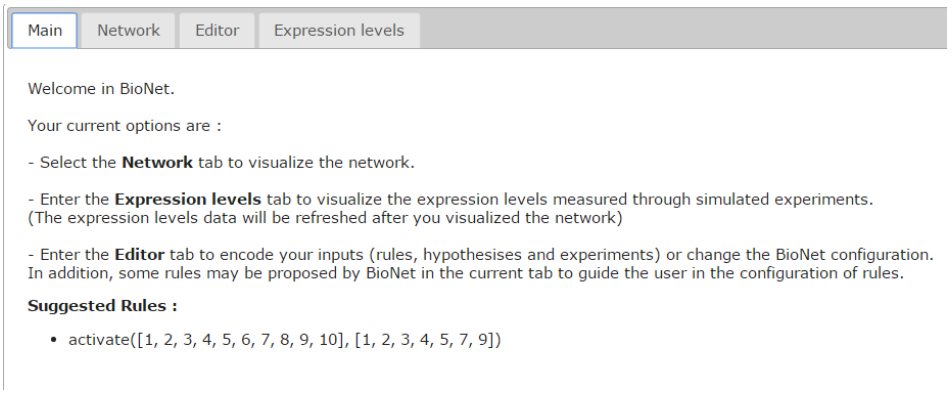

FIGURE 5.7: BioNet - Based on the generated experimental data, a rule is suggested.

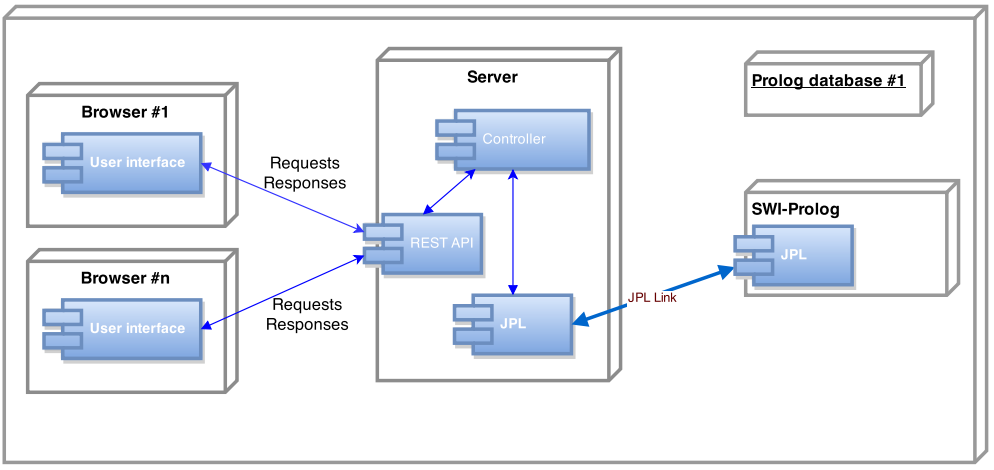

FIGURE 5.8: Global architecture of BioNet

instantiated and processed in the core SWI-Prolog. Once processed, all possible solutions will be returned in a higher level form and returned to the caller. In this context, the caller will be the server/controller, executed by the Java virtual machine (JVM) (JVM). Since Prolog and Java are different technologies that are not intended to interface each other naturally, a prerequisite is to have a mediator able to let both technologies communicate. The JPL (*Java-calls-Prolog API*) [126] hides the low level aspects and provides a bridge between Java and Prolog via a high level Application programming interface (API). Consequently, it is possible from Java to use advanced SWI-Prolog modules as the module for constraint logic programming.

5.5.2 Java

The choice of Java technology for the server part has been made regarding to its portability, popularity and maturity. Java can easily handle various sort of tasks and an Java implementation can be easily adapted to the situations as from the processing of large amounts of user queries quickly or interfacing with another language and support

changes. In addition, a lot of bridges to another technologies have been developed for Java, and even several Prolog engines provide Java bridges, including B-Prolog and SWI-Prolog engines. Some Prolog implementations have even been written entirely in Java, as tuProlog[1]. Regarding the current architecture, the Java controller plays mainly the role of carrying user queries from the user interface to the model checkers written in Prolog, in the SWI-Prolog engine, thankfully to JPL. Additionally, since the engine written in Prolog is separated from the Java part, and since the Java part is only used for the server layer, it is possible to implement a user interface independent from the Java technology. In the large, it could be possible to evolve a specific layer of the architecture without this change would even affect the others. The Java code used to interact with SWI-Prolog via JPL is partially shown in appendix A section A.6.2 page 123

5.5.3 Server

The server part is written in Java and is deployed on a local Apache Tomcat server. It serves first as a request handler for any incoming query from the user interface to the server, and from the architecture perspective, the server part is composed of a service layer and a controller. The controller is the part that check syntax and semantic of those user queries before they are processed. It also ensures the result to return the expected queries results to the effective caller (user interface, or simply user). The service layer is responsible for exposing a language-independent interface to interact with the system, under the form of a REST API. Representational State Transfer (REST) is an architectural style that has gained popularity as an alternative to existing services protocols based on SOAP and WSDL. It is a set of guidelines and recommendations for designing and creating web services. REST APIs, applied to a service layer, are essentially based on standard HTTP methods like GET, PUT, POST and DELETE to respectively execute retrieve, update, create or delete operations on the server. Resources in a REST API are usually defined and acceded with standard URIs used in combination with a specific HTTP method. For example

```
GET http://www.website.com/rest/bio/
```

where GET specify the request type, will ask the server to simply return the content at this URI, since the GET method has been used.

Out of these technical details, a REST architecture must support a set of properties and constraints. These properties include code and data portability, reliability of the system, modifiability of the components to support changing needs, and simplicity of

[1] http://tuprolog.alice.unibo.it/

exposed interfaces. Additionally, a REST architecture must follow a set of constraints, including the following :

Client-Server The client and server parts are separated and their communication with each other should involve a common interface. This separation of concerns improve the portability of the client and the modifiability of the overall system, since each part could be potentially modified internally without affect the other. The server is not concerned by the user interface, and neither is the user interface concerned with data storage. Since the interface is the only common part between these two layers, it is also the only part where changes would affect both client and server sides of the system.

Stateless The stateless constraint means that no client context or session is saved on the server between requests. Each request from the user interface needs to be self sufficient, i.e., it should be informative enough to be processed in any conditions, independently from the server state. It also means that any request could be persisted somewhere, for example in an external database, for later reuse, since a request is context-independent. As a consequence, the client is responsible to handle a bit of application state to be able to send valid requests to the server, with valid context embedded in the request. The client has to handle the logic of state transitions between the requests, which means it has to save the context related to the user session and has to choose when to make a change in this context, as well as initiate state transitions and knowing when to make new requests. It can be argued as if no context was managed at client side, the client would endlessly send the same request with the same inputs and context to the server, and in consequence would obtain the same results from the server, since nothing would have changed in the context between two requests on server side. However, it is only a minor burden for the client while the number of distinct states does not increase too much, and since it offers a lot of advantages.

Cacheable Optionally, the client could save the response received from the server, under the condition that those responses are defined as cacheable. If enabled, it can be useful to reuse a previous response to a request, but it can be disabled to prevent reuse of obsolete response and to force the client to always use up-to-date data. When enabled, each request to the server is returned with a response containing a mention of the last time that resource was updated, and in turn the client can request the server to return an up-to-date response. When fully handled, the caching can reduce or even eliminate the amount of needed requests.

Layered System In the case of a multiserver architecture, the client should not know if it is connected to an end server or an intermediary server. With several servers

involved, the system scalability would be improved. Several servers could share common resources caches and ensure failover according to each other, consequently if one server would fail or become overloaded, another would be ready to answer, without requiring any change from the client side.

These constraints should not be considered as a burden. Regarding the current architecture, these constraints have helped to develop a more robust, modifiable system in addition to increase its portability. The architecture allows any client application developed in another technology to interact with the current system through the public interface, and indirectly invoke the Prolog engine. The stateless aspect reduce the costs of handling context on the server, as well as the overall complexity of the server. However, since no context is saved on server-side, the system requires from the client to provide enough information within each request, at the cost that each request is a little heavier compared to a classical client-server architecture where all this context information is already persisted on server-side and does not have to be provided each time. It is only a problem regarding the costs related to network bandwidth, and only if it represents an expensive resource for the system supporting this architecture.

The cacheable constraint is not fully supported by the current architecture since it is optional and offers no particular value in this case, since the users will essentially work with updated resources and update existing ones. However, the caching has been implemented somewhere else on server-side, in order to reduce at least a bit the time needed to load some Prolog programs. Each time an instance of a Prolog program (or database) is required, it is also saved in cache for further reuse.

Finally, having a layered system is a great advantage when the resources are sufficient to support a multi server architecture with failover and load balancing techniques. The current system does not handle this scenario but this could be an improvement.

The Java code of a REST provider of BioNet is shown in appendix A section A.6.1 page 120. It is used to serve BioNet configuration files from the server to the client via the REST API.

5.5.4 User interface

: Separated from the Java and Prolog engines, the user interface is responsible for providing a visual representation of the model the user is working on. It allows the user to enter some queries (see Figure 5.4 page 86), forwards those queries to the server and handle the server response. For instance, it renders the network as well as a chart with the generated experimental data.

When the user submits a query, and once the server response has been treated, the network is automatically refreshed. The tools used to render the interface include *JavaScript* libraries and simple HTML pages. For instance, the display of the GRN is performed by a JavaScript library, Vis.js[2], well suited for interacting with networks. The choice of this library has been made after a comparison with many ones, such as D3.js[3], Sigma js[4], JoinJS[5] Cytoscape js[6] and linkurious.js[7]. The goal was to offer a modern tool for graph visualization in a web browser. Vis.js was retained for its qualities, since it provides the essential required features of BioNet and it is quite simple to configure.

Currently, the user interface of BioNet can only be acceded via browser based technologies but this is an advantage since it allows the user to interact without requiring from him more than a web browser. Vis.js lets the user interact in real-time with the graph, which was one of my expectations. At the opposite, numerous graph rendering tools deny any real-time modification from the user. In BioNet, we consider the interface as a playground for the user and wanted him to be able to interact as much as possible and give him as much control as possible on the model. Vis.js allows real-time modification of a graph and has other capabilities. However, Vis.js is not used to its full potential, indeed the initial goal was to let the user interact modify the model by simply manipulating the nodes, relationships and weights with his mouse. Hopefully, the code of the user interface is quite simple and easy to extend. The JavaScript code used to start the network visualizer is partially shown in appendix A section A.8.2 page 128.

In addition, the Ace[8] JavaScript editor is used to let the user encode user inputs in BioNet. The JavaScript code used to start the code editor is partially shown in appendix A section A.8.1 page 127.

Finally, the Highcharts[9] JavaScript library is used for the rendering of gene expressed data. Highcharts provides a lot of advanced features for charts visualization and was retained for its simplicity as well as its good documentation.

5.5.5 Messages

: The communication between the server-side and the client-side parts is essentially made using asynchronous requests from the user interface to the server. Those requests

[2]http://visjs.org/
[3]http://d3js.org/
[4]http://sigmajs.org/
[5]http://www.jointjs.com/
[6]http://js.cytoscape.org/
[7]https://github.com/Linkurious/linkurious.js
[8]`http://ace.c9.io/#nav=about`
[9]`http://www.highcharts.com/`

serve to transmit the user queries between the different layers, i.e., from the frontend layer (client) implemented in JavaScript to the server layer implemented in Java. For this purpose, a common message description language is required to facilitate this communication and describe the messages. The JavaScript Object Notation (JSON) format has been retained among other existing solutions like XML. It is an open standard format, derived from the JavaScript notation and consists in human readable text and which is language-independent.

An example of JSON data is shown below :

```
{"network":[{
  "name": "Lac Operon",
  "nodes":[{
    "label":"'lacZYA mRNA'", "type":"node","value": 0
  }, {
    "label":"'LacI'", "type":"node", "value": 0
  }, {
    "label":"'lac operon'", "type":"node", "value": 0
  },{
    "label":"lacZYA", "type":"node", "value": 0
  }],
  "edges":[{
      "from": {"label":"lacZYA", "type": "regulator", "value": 0},
      "to": {"label":"'lacZYA mRNA'", "type": "regulated", "value": 0},
      "label": "+",
      "weight": 1
  }],
  "rules":[
    "activate([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [1, 2, 3, 4, 5, 7, 9])"
  ],
  "measures":[{
    "experimentId":"1",
     "name":"'lacZYA mRNA'",
     "level":0
  }]
}]}
```

For instance, this example is a small part of a real message exchanged in BioNet between the server and the user interface (web-browser). It describes, partially, the model of the regulation in the *lac* operon as modelled in section 5.4 page 84.

JSON is already natively supported by some languages, and in other cases numerous languages are able to deal with JSON encoding/decoding since extensions and libraries have been developed to support this format. Even recent web browsers have included native support of encoding/decoding JSON. In the current model, each user query can be described as a message in the JSON format. This format is not new and is already used in many cases, for example in Ajax techniques for cases where new data need to be displayed in a web page after this page has already been loaded and displayed (at least in part). JSON being lighter than XML, it has spread in many cases involving web technologies.

5.6 Implementation

5.6.1 Prolog Implementation

The complete BioNet Prolog source code is shown in appendix A section A.4 page 108. The main Prolog predicate is shown below, while its definition can be found in appendix A section A.2 page 105, as well as examples of input and output of the resolution.

```
1 solve(WeightMatrix, Relationships, Predictions, SuggestedRules).
```

5.6.2 Server and user interface

Parts of the source code of server and user interface are shown in the appendix. For instance, some examples of the Java source code are shown in appendix A section A.6 page 120. In addition, some examples of the JavaScript source code are shown in appendix A section A.8 page 127.

5.6.3 Dependencies

BioNet has been developed and tested on Windows 7, with Eclipse IDE. The development environment of BioNet depends on the following tools :

1. Java SE Development Kit (JDK)[10]

[10] `http://www.oracle.com/technetwork/java/javase/downloads/index.html`

2. SWI-Prolog[11] : a complete Prolog environment for Windows, Linux and Unix.
3. JPL[12], embedded as a library (jpl.jar) in the SWI-Prolog installation directory.
4. Eclipse IDE with Java development tools [13].
5. Gradle[14] : A build automation with dependency management capability. It is used as a builder and dependency management tool for BioNet. The plugins for Eclipse IDE may also be useful. They can be found in [15].
6. Apache Tomcat (optional)[16] : an open source Servlet Container.
7. Misc. Java libraries : Once installed and configured, Gradle automatically collects the remaining Java dependencies, such as specific Java libraries required for BioNet.

The assembly of BioNet is automated by Gradle, which allows defining Java dependencies in a configuration file, as shown in appendix A section A.7 page 126. Gradle is configured to build BioNet as a web archive (.war) and then deployed on a Servlet Container such as Apache Tomcat.

5.7 Limits and perspectives

BioNet benefits from an advanced but quite simple architecture, a specific user interface addressing the user requirements, and proposes simple but configurable features for the modelling of GRNs. However, this tool is still a prototype with limited features, that have been tested for particular scenarios. Some features initially intended have not been developed for the current release and some limits may appear. This section describes some of those intended features as well as possible perspectives for the further improvement of BioNet.

Interface The encoding of user inputs as well as the configuration could be made easier. At this time, the modifications of inputs and configuration are performed via a Prolog editor embedded into BioNet as seen in Figure 5.1 page 78. The initial scope of BioNet was to provide an advanced user interface allowing the user to retrieve datasets from external databases and encode user rules and hypotheses via an advanced editor. For example, the gene expression data as well as an existing problem definition could be retrieved from BioNet. However, the development of those features would require additional work and time. In addition, the first goal

[11] `http://www.swi-prolog.org/`
[12] `http://www.swi-prolog.org/packages/jpl/`
[13] `https://eclipse.org/downloads/`
[14] `https://gradle.org/`
[15] `https://docs.gradle.org/current/userguide/eclipse_plugin.html`
[16] `http://tomcat.apache.org/`

of BioNet, which is a prototype, is providing a tool for inferring network topology and parameters from inputs, and it has been achieved. Hopefully, BioNet is a flexible tool and can still be improved if needed.

Configuration of regulation dynamics The simplicity of BioNet is an advantage, since it is perfectly suited for the qualitative modelling of simple GRNs. However, in its current configuration, BioNet only handles some regulatory interactions, such as single/cooperative activation, inhibition, as well as self-inhibition and auto-activation. Similarly, due to its deterministic nature, it cannot be used to model asynchronous interactions or probabilistic/stochastic dynamics studied in the Chapter 3. However, the dynamics of the regulation in a real biological system are more complicated, as described in Chapters 2 (page 5) 3 (page 29) and still not well understood. The modelling proposed by BioNet is simple and remains not sufficient to replace or complete the existing formal tools and simulation tools available for the modelling of biological systems. Hopefully, new regulation dynamics could be modelled in BioNet in addition to new configuration possibilities, with a bit of additional effort. In any case, it is essential to provide a large set of configurable parameters, regarding to the constant improvement in the understanding of gene regulatory networks and the need of adapted features to obtain accurate results.

Architecture In order to have a fully compliant REST architecture, some constraints could be better supported. For example load balancing and failover, or a better caching system. Those improvements involve choices and little transformations of the current architecture. For the caching system, it could imply to manage multiple access to cached Prolog databases, which involves a better interfacing with the SWI-Prolog engine. However, the JPL interface could fail in some cases at handling concurrent access to the same Prolog resources and multiple open queries at the same times. With more investigations on this part, the caching system could be improved and the application could be able to support a greater volume of requests to the same resources in shorter delays.

Data normalization and noise cleansing As seen in Chapter 3 page 31, the inherent noise in the gene expression data is a recurrent problem related to the limits of the techniques used to collect those data. It results in observable inconsistencies and contradictions when multiple datasets from different databases are compared each other. It is a problem when, for or a same experiment, a gene is shown as expressed (respectively repressed) in a dataset and unexpressed in another one. Multiple factors are responsible for this noise, and generally particular data analysis methods are required to clean this noise, with varying results in the quality

perspective. The current implementation does not provide such noise-cleaning feature, and consequently we consider the data as already normalized before being used in BioNet.

APIs The architecture was initially intended to support interactions with external APIs but it was not a fundamental prerequisite of a fully functional system and can be more considered as an extra comfort. However, with additional development, interfaces to external API could be an interesting investment of effort, as some external biological databases could be queried to provide some real data and provide a base to work with. As an example, pathways could be retrieved from KEGG [17] or datasets of expression levels could be retrieved from GEO[18].

5.8 Conclusion

This chapter presented a support tool with aiming of solving questions related to the study of GRNs. We started with a list of user requirements for this tool, then we inferred the model and user interface from the requirements. We presented BioNet, a tool designed to help biologist researchers at their work on GRN, and we described its main features, as well as the process of modelling in this tool. This tool was intended to be easy to use for the user and its user interface benefits from a simple but user friendly interface built upon advanced JavaScript libraries. The current version enables him to interact with the user interface of BioNet via a web browser. From a problem description encoded in BioNet via his web browser, the user has the possibility to visualize the inferred network and the nature of the interactions involved in that network. In addition, BioNet proceeds to the simulation of the gene experimental data and gives some clues to the user to adapt his problem description regarding to the observed experimental data.

To illustrate the features of BioNet, we studied the modelling of the gene regulation in the particular case of the lactose metabolism, for instance the digest of the lactose in the *lac* operon. We observed from this simple example that BioNet can be used to simulate interesting behaviours and improve user models. Then, we described the architecture used for the building of BioNet, including the key tools and technologies involved. For instance, we showed how Java interacts with the Prolog part of BioNet via SWI-Prolog and its CLP(FD) library, that provides the constraint solving capabilities of BioNet. Finally, we exposed the advantages and limits of BioNet, and we proposed some perspectives for further improvements.

[17] http://www.genome.jp/kegg/
[18] http://www.ncbi.nlm.nih.gov/geo/

Chapter 6

Conclusion

In this thesis, interest was focused on the study and modelling of gene regulatory networks (GRN). Many tools have been developed to model the complex mechanisms involved in the regulation of gene expression. However, the study of GRN is still complex. In this thesis, we propose a new approach to answer the complex biological questions, by using a promising programming technique. Then we present a prototype, BioNet, that we designed as a support tool with aiming of helping the biological researcher at his work on GRNs.

In the first chapters, we explored the main aspects of the study of biological systems, with the emphasis on GRNs. We defined the most important mechanisms involved in regulation of gene expression in living organisms and then we described the issues faced by the biologist researchers to understand those phenomenons. We showed that the main issues are related to the complexity and the nature of the studied mechanisms, especially when lacking consistent or quantitative data and knowledge on the studied phenomenons. To address those issues, numerous formalisms and tools have been proposed in the last decades. However, those modelling and simulation tools are limited and address only partially those issues. For instance, the modelling and study of GRNs remains a challenge, even with large amount of data available. Many questions are very difficult to answer without clues or the sufficient knowledge, and the elucidation of the nature of interactions in GRNs often requires intuition from the biologist researcher.

In this thesis, we presented the advantages of constraint logic programming (CLP), a promising programming paradigm well suited for the solving of complex problems with many unknowns. CLP is already used in various fields to solve various real world problems. Interest was focused on the ability of CLP to solve biological problems, with the emphasis on GRNs. We described the essence of constraint logic programming techniques and illustrated the modelling of a problem in the syntax of a constraint

programming language, on the basis of the problem description. Finally, we used those programming techniques for the design and the building of BioNet, a prototype of a support tool, with interesting capabilities. We presented its model and user interface, as well as the highly configurable features provided to the user for the modelling of GRNs. We illustrated the ability of BioNet to model the regulation mechanism with the example of the *lac* operon and a particular scenario related to the mechanism of the digest of the lactose in bacterium *Escherichia coli*.

From that point, we have taken the opportunity to criticize the results. Based on the simulation of experimental data performed by BioNet, we observed that BioNet is able to model and simulate simple regulation mechanisms. However, the results should be taken with caution. BioNet is still limited to a qualitative modelling of GRNs and does not cover all the features of other modelling tools. For instance, it does not support modelling of stochastic or asynchronous interactions. Hopefully, Bionet is highly configurable, and its advantageous architecture allows possible extensions. Since BioNet is still a prototype with a reduced set of features, we have taken the opportunity to propose some improvements, with the expectations that the missing features can be developed to enable the use of BioNet in the modelling of very complex scenarios.

List of Tables

List of Figures

Appendix A

Implementation description

A.1 Sample user input

BioNet requires the user input to be in the form of predicates expressed in the syntax of Prolog. An example of user input can be defined as follows :

```
1 def(1, 'Gene01').
2 def(2, 'Gene02').
3 def(3, 'GeneY').
4 def(4, 'FactorX').
5 experiences([exp(1,1,2),exp(2,1,3),exp(3,1,3)]).
6 weightHypothesis([edge(1,2,6,6,'+'),edge(2,3,4,4,'+'),edge(3,1,6,6,'+')]).
7 rules([inhibate([1,3],[2],2)]).
```

A.2 Execution of BioNet in SWI-Prolog

The main predicate of BioNet is described below. It infers the results of the resolution in his parameters, with respect to the user inputs and BioNet configuration provided by the user.

```
1 solve(WeightMatrix, Relationships, Predictions, SuggestedRules).
```

WeightMatrix a list of lists, which can be described as a matrix. The matrix contains a row for each regulator, and a column for each target. The values are the weights of relationships.

Relationships a list of edges. An edge is defined by two related nodes, the min/max weights and a regulation sign such as '+" or '-' respectively for activation or inhibition

Predictions a list of lists, which can be described as a matrix. The matrix contains a row for each experiment, and a column for each node. The values in the matrix are the node expression levels.

Rules a list of rules proposed by BioNet, regarding to properties observed a certain amount of times (this amount is related to the value of *minoccur* parameter in the BioNet configuration).

A preliminary step to the invocation of the above predicate is to load the module for CLP(FD), with the following command :

```
use_module(library(clpfd)).
```

Then, the bionet engine as well as the user inputs can be loaded as follows, under the condition that bionet.pl and userinputs.pl are well defined Prolog files available in the working directory of SWI-Prolog :

```
load_files([bionet, userinputs]).
```

An example of user input file content is shown in section A.1. Additionally, the BioNet configuration must be loaded, in the user input file or in another file. An example of BioNet configuration file content is shown in section A.3.

The result of the resolution of BioNet for the of the previously described input in SWI-Prolog is shown below :

```
Rules list :
inhibate([1,3],[2],2)
Weight Matrix for nodes relationships (a line for each gene, a column
  ↪  for each target) :
[0,6,0]
[0,0,4]
[6,0,0]
List of edges and their weights :
edge(1,2,6,6)
edge(2,3,4,4)
```

```
edge(3,1,6,6)
Predicted expression Values (a line for each measured time, a column for each gene) :
[2,3,3]
[3,1,4]
[4,0,5]
[5,0,3]
[6,0,1]
[7,0,0]
[5,1,0]
[3,2,1]
[4,0,2]
[5,0,0]
[3,1,0]
[1,2,1]
[2,0,2]
[3,0,0]
[1,1,0]
Suggested properties:
activate([1],[2])
activate([1,2,3],[1,3])
activate([1,3],[1])
autoactivate([2],[2])
inhibate([1],[3])
inhibate([1,2,3],[2])
inhibate([1,3],[2,3])
true.
```

A.3 Configuration

```
param(minlength, 1).
param(maxtime, 15).
param(minweight, 0).
param(minrequiredweight, 3).
param(minactivatorweight, 1).
param(maxweight, 1000).
param(minoccur, 3).
param(weightsfromrules, 1).
```

```
param(minvalue, 0).
param(maxvalue, 1000).
param(defaultinhibitionrate, 2).
param(defaultactivationrate, 1).
param(minrequiredexpressionlevel, 1).
```

A.4 BioNet Prolog source code

```
/** Please invoke the following first **/
%use_module(library(clpfd)).

% at first execute use_module(library(clpfd)).

/** START HERE */
% - solve(WeightMatrix, Relationships, WeightInteractions,
  ↪ PredictedExpressionValue).

solve(WeightMatrix, Relationships, Predictions, SuggestedRules) :-
        % get initial experiments list
        experiences(Exps),
        rules(Rules),
        weightHypothesis(HypothesisRelationships),
        ((param(weightsfromrules, 1),
         rules2relationships(Rules, RulesRelationships),
         append(HypothesisRelationships, RulesRelationships,
          ↪ WeightHypothesis))
        ;append(HypothesisRelationships, [], WeightHypothesis)
        ),
        param(minlength, MinLength),
        param(minoccur, MinOccur),

        % sort experiments, remove duplicates
        findall(A, member(exp(A,_,_),Exps), As),
        sort(As, Ss),
        length(Ss, MaxLength),
        MaxLength >= MinLength,

```

```
28        % generate eligible relationships (nodes) with weight
29        make_graph(Exps, MaxLength, WeightMatrix, Relationships,
          ↪ WeightHypothesis),
30        % generate eligible expression values for nodes until MaxTime
31        make_predictions(Exps, WeightMatrix, Relationships, Rules,
          ↪ Predictions, MaxLength),
32
33        % labelling
34        labelRows(WeightMatrix),
35        labelRows(Predictions),
36
37        !, % stop after first succesful labeling/inference
38        writeln('User rules :'),
39        maplist(writeln, Rules),
40        writeln('Inferred weight matrix for nodes relationships (a line
          ↪ for each gene, a column for each target) :'),
41        maplist(writeln, WeightMatrix),
42        writeln('List of inferred edges with their relationship
          ↪ weights/signs :'),
43        maplist(writeln, Relationships),
44        writeln('Inferred expression levels (a line for each experiment,
          ↪ a column for each node) :'),
45        maplist(writeln, Predictions),
46        suggestRules(Predictions, MinOccur, SuggestedRules),
47        writeln('Suggested rules:'),
48        maplist(writeln,SuggestedRules)
49        .
50
51 suggestRules(Rows, MinPropOccurence, SortedRules) :-
52    param(minrequiredexpressionlevel, MinRequiredExpressionLevel),
53    findall(inhibate(Inhibitors, Inhibited),(
54     nth1(ExpId, Rows, Exp),
55     findall(Key, (nth1(Key, Exp, Value), Value >=
       ↪ MinRequiredExpressionLevel), Inhibitors),
56     ExpId2 is ExpId + 1, nth1(ExpId2, Rows, Exp2),
57     findall(Key, (nth1(Key, Exp2, V2),V2 < MinRequiredExpressionLevel),
       ↪ Inhibited),
58     length(Inhibitors, CountInhibitors), length(Inhibited,
       ↪ CountInhibited), CountInhibitors>0, CountInhibited>0)
```

```
59      , SuggestIRules),
60
61      findall(activate(Activators, Activated),(
62      nth1(ExpId, Rows, Exp),
63      ExpId2 is ExpId + 1, nth1(ExpId2, Rows, Exp2),
64      findall(Key, (nth1(Key, Exp, Value), Value >=
        ↪ MinRequiredExpressionLevel), Activators),
65      findall(Key, ((nth1(Key, Exp2, V2)), nth1(Key, Exp, V1), V1<V2),
        ↪ Activated),
66      length(Activators, CountActivators),
        ↪ length(Activated,CountActivated), CountActivators>0,
        ↪ CountActivated>0)
67      , SuggestARules),
68
69      findall(autoactivate(Activators, Activators),(
70      nth1(ExpId, Rows, Exp),
71      ExpId2 is ExpId + 1, nth1(ExpId2, Rows, Exp2),
72      findall(Key, (nth1(Key, Exp, Value), Value <
        ↪ MinRequiredExpressionLevel, nth1(Key, Exp2,
        ↪ V2),V2>=MinRequiredExpressionLevel), Activators),
73      length(Activators, CountActivators), CountActivators>0)
74      , SuggestAutoRules),
75
76      append(SuggestAutoRules,SuggestARules, ActivationRules),
77      append(SuggestIRules,ActivationRules,SuggestRules),
78
79      findall(Rule, (member(Rule, SuggestRules),
80      count([Rule], SuggestRules, Count),
        ↪ Count>=MinPropOccurence),FilteredRules),
81     sort(FilteredRules, SortedRules).
82
83  count(Elem, List, Count) :-
84      subtract(List, Elem, Deleted), length(List,Before),length(Deleted,
        ↪ After),Count is Before - After.
85
86  labelRows([]).
87  labelRows([Row|Rows]) :- label(Row), labelRows(Rows).
88
89  length_list(Length, List) :- length(List, Length).
```

```
90
91 make_graph(Exps, MaxLength, WeightMatrix, Nodes, WeightHypothesis) :-
92     param(minweight,MinWeight),param(maxweight,MaxWeight),
93     length(WeightMatrix, MaxLength),
94     maplist(length_list(MaxLength), WeightMatrix),
95     append(WeightMatrix, Vs), Vs ins MinWeight..MaxWeight,
96     from(1, Exps, WeightMatrix, MaxLength, MinWeight, MaxWeight, Nodes,
       ↪ WeightHypothesis).
97
98 rules2relationships(Rules, Relationships) :-
99     findall(edge(From, To, Weight, Weight, '+'), (
100      member(activate(Activator, Target, Weight), Rules),
101      member(From, Activator),
102      member(To, Target)),
103    Activators),
104    findall(edge(From, To, Weight, Weight, '-'), (
105      member(inhibate(Inhibitor, Target, Weight), Rules),
106      member(From, Inhibitor),
107      member(To, Target)),
108    Inhibitors),
109    append(Activators, Inhibitors, Edges),
110    merge_edges(Edges, [], Relationships).
111
112 get_edges(Graph, GeneFrom, GeneTo, Edges) :- findall(edge(GeneFrom,
   ↪ GeneTo, B, C, Label), member(edge(GeneFrom, GeneTo, B, C, Label),
   ↪ Graph), Edges).
113
114 merge_edges(Edges, Acc, Merged) :-
115    nth1(1, Edges, edge(From,To,_,_,_)),
116    get_edges(Edges, From, To, TreatedEdges),
117    gen_interval(TreatedEdges, BestInterval),
118    subtract(Edges, TreatedEdges, RemainedEdges),
119    append([BestInterval], Acc, TempMerged),
120    merge_edges(RemainedEdges, TempMerged, Merged), !.
121 merge_edges([], N, N).
122
123 gen_interval([edge(From, To, WeightMin, WeightMax, Label)|T], edge(From,
   ↪ To, BestMin, BestMax, StrongestLabel)) :-
124    gen_interval(T, edge(From, To, SubMin, SubMax, SubLabel)),
```

```
125     min_value(WeightMin, SubMin, BestMin),
126     max_value(WeightMax, SubMax, BestMax),
127     (BestMax is WeightMax, StrongestLabel = Label; StrongestLabel =
        ↪ SubLabel).
128
129 gen_interval([edge(From,To,Min,Max, Label)], edge(From,To,Min,Max,
    ↪ Label)).
130 gen_interval([], _) :- fail.
131 min_value(A, B, A) :- A =< B, !.
132 min_value(B, A, A) :- A =< B, !.
133 max_value(A, B, A) :- A >= B, !.
134 max_value(B, A, A) :- A >= B, !.
135
136 make_predictions(Exps, WeightMatrix, Relationships, Rules, Predictions,
    ↪ MaxLength) :-
137     param(maxtime,MaxTime),
138     length(Predictions, MaxTime),
139     length(WeightMatrix, MaxLength),
140     maplist(length_list(MaxLength), Predictions),
141     predict_at(1, Exps, WeightMatrix, Relationships, Rules, [],
        ↪ Predictions, MaxLength).
142
143 matrix_at(Matrix, RowIdx, ColIdx, Value) :- nth1(RowIdx, Matrix, Row),
    ↪ nth1(ColIdx, Row, Value).
144
145 max_list_clpfd(List, Max) :-
146 findall(Inf, (member(M, List),fd_inf(M,Inf)),R), findall(Sup,
    ↪ (member(M,List),fd_sup(M,Sup)),RSup), max_list(R, Maax), Max #>=
    ↪ Maax, max_list(RSup, Maax2), Max #=< Maax2.
147
148 max_influencors(Target, WeightMatrix, Influencors, Max) :-
149   findall(Origin-Value, (
150     nth1(Origin, WeightMatrix, Targets),
151     nth1(Target, Targets, Value)), Vs),
152   pairs_keys_values(Vs, Keys, Vvv),
153   max_list_clpfd(Vvv, Max),
154   findall(Key, (member(Key,Keys), nth1(Key,Vvv,MaxKey),
        ↪ fd_inf(MaxKey,Inf), Inf #= max(Max, Inf)), Influencors).
155
```

```
156 predict_at(_, _, _, _, _, _, [], _).
157 predict_at(FromTime, Exps, WeightMatrix, Relationships, Rules,
    ↪ PreviousPrediction, [CurrentPredictions|RestPredictions], MaxLength)
    ↪ :-
158     param(maxvalue,MaxValue),param(minvalue,MinValue),
159     CurrentPredictions ins MinValue..MaxValue,
160     NextTime is FromTime + 1,
161     predict_at_for(FromTime, 1, Exps, WeightMatrix, Relationships,
        ↪ Rules, PreviousPrediction, CurrentPredictions, MaxLength),
162     predict_at(NextTime, Exps, WeightMatrix, Relationships, Rules,
        ↪ CurrentPredictions, RestPredictions, MaxLength).
163
164 get_active_influencors(CurrentTime, CurrentId, Exps,
    ↪ PreviousPredictions, WeightMatrix, ActiveInfluencors, Weight) :-
165     param(minrequiredexpressionlevel,MinRequiredExpressionLevel),
166     param(minactivatorweight, MinActivatorWeight),
167     max_influencors(CurrentId, WeightMatrix, Influencors, Weight),
168     Weight #>= MinActivatorWeight,
169     PreviousTime is CurrentTime - 1,
170     findall(Influencor, (
171        member(Influencor, Influencors),
172        (   member(exp(Influencor, PreviousTime, InfluencorValue), Exps)
173          ;(nth1(Influencor, PreviousPredictions, V), fd_inf(V,
              ↪ InfluencorValue))
174        ),
175        InfluencorValue >= MinRequiredExpressionLevel
176     ),
177     ActiveInfluencors).
178
179 get_expressed_nodes(Expressions, Sorted, MinExpr, MaxExpr) :-
180     findall(K, (nth1(K, Expressions, V), V #>= MinExpr, V #=< MaxExpr),
        ↪ Expressed),
181     sort(Expressed, Sorted).
182
183 activatorRules(NodeId, Rules, SelectedRules) :-
184     findall(Rule, (Rule=activate(_, To, _), member(Rule,Rules),
        ↪ subset([NodeId], To)), SelectedRules).
185 inhibitorRules(NodeId, Rules, SelectedRules) :-
```

```
186     findall(Rule, (Rule=inhibate(_, To, _), member(Rule,Rules),
        ↪ subset([NodeId], To)), SelectedRules).
187
188 predict_at_for(_, _, _, _, _, _, _, [], _).
189 predict_at_for(CurrentTime, CurrentId, Exps, WeightMatrix,
    ↪ Relationships, Rules, PreviousPredictions, [Prediction|Predictions],
    ↪ MaxLength) :- (
190 param(maxtime, MaxTime),
    ↪ param(maxvalue,MaxValue),param(minvalue,MinValue),
191 ( CurrentTime =< MaxTime,
192     CurrentId =< MaxLength,
193     NextId is CurrentId + 1,
194     (
195      % retrieve the previous experiments values as current predictions
196       member(exp(CurrentId, CurrentTime, Value),Exps),
197       nth1(1, [Prediction|Predictions], Value)
198      ;predict_value_at_for_laws(CurrentTime, CurrentId, Exps, Rules,
        ↪ PreviousPredictions, Prediction)
199      ;predict_value_at_for_weights(CurrentTime, CurrentId, Exps,
        ↪ Relationships, WeightMatrix, PreviousPredictions, Prediction)
200     ), !,
201     predict_at_for(CurrentTime, NextId, Exps, WeightMatrix,
        ↪ Relationships, Rules, PreviousPredictions, Predictions,
        ↪ MaxLength))
202 ); true.
203
204 predict_value_at_for_laws(CurrentTime, CurrentId, Exps, Rules,
    ↪ PreviousPredictions, Prediction) :-
205  param(maxvalue,MaxValue),param(minvalue,MinValue),
206  param(minrequiredexpressionlevel,MinRequiredExpressionLevel),
207    PreviousTime is CurrentTime - 1,
208 (  member(exp(CurrentId, PreviousTime, PreviousExpr), Exps)
209   ;nth1(CurrentId, PreviousPredictions, PreviousExpr)
210 ),(
211    (get_expressed_nodes(PreviousPredictions, Influencors,
       ↪ MinRequiredExpressionLevel, MaxValue),
212     activatorRules(CurrentId, Rules, SelectedRules),
213     member(activate(From, To, Strength), SelectedRules),
214     subset(From, Influencors),
```

```
215     Prediction #= PreviousExpr + Strength
216    )
217   ;(get_expressed_nodes(PreviousPredictions, Influencors,
      ↪  MinRequiredExpressionLevel, MaxValue),
218     inhibitorRules(CurrentId, Rules, SelectedRules),
219     member(inhibate(From, To, Strength), SelectedRules),
220     subset(From, Influencors),
221     (fd_sup(PreviousExpr, Sup), (Sup - Strength >= MinValue), Prediction
        ↪  #= PreviousExpr - Strength;Prediction #= MinValue)
222    )
223   ;(MaxNotExpressedLevelValue is MinRequiredExpressionLevel -1,
224     get_expressed_nodes(PreviousPredictions, Influencors, MinValue,
        ↪  MaxNotExpressedLevelValue),
225     findall(Rule, (Rule=autoactivate(_, To, _), member(Rule,Rules),
        ↪  subset([CurrentId], To)), SelectedRules),
226     member(autoactivate(From, To, Strength), SelectedRules),
        ↪  subset(From, Influencors),
227     (fd_sup(PreviousExpr, MaxNotExpressedLevelValue), Prediction #=
        ↪  PreviousExpr + Strength)
228    )
229 ).
230
231 predict_value_at_for_weights(CurrentTime, CurrentId, Exps,
    ↪  Relationships, WeightMatrix, PreviousPredictions, Prediction) :-
232  param(maxvalue,MaxValue),param(minvalue,MinValue),
233    PreviousTime is CurrentTime - 1,
234    (member(exp(CurrentId, PreviousTime, PreviousExpr), Exps)
235    ;nth1(CurrentId, PreviousPredictions, PreviousExpr),
236
237 (% Change expression value regarding to relationships
    ↪  (activators/inhibitors, weight ...)
238    (get_active_influencors(CurrentTime, CurrentId, Exps,
       ↪  PreviousPredictions, WeightMatrix, ActiveInfluencors, Weight),
239      length(ActiveInfluencors, CountActiveInfluencors),
         ↪  CountActiveInfluencors > 0,
240      (param(minactivatorweight,
         ↪  MinActivatorWeight),Weight#>=MinActivatorWeight,
241       param(defaultactivationrate, Activation),
```

```
242       (member(InfluencorId, ActiveInfluencors),member(edge(InfluencorId,
          ↪ CurrentId, MinInfluenceRate, _, Sign),Relationships)),
243        ((Sign == '+', (fd_sup(PreviousExpr,Sup), Sup + Activation =<
           ↪ MaxValue, Prediction #= PreviousExpr + Activation; Prediction
           ↪ #= MaxValue))
244       ; Sign == '-', (fd_sup(PreviousExpr,Sup), Sup > MinValue,
          ↪ param(defaultinhibitionrate, Inhibition), Prediction #=
          ↪ PreviousExpr - Inhibition; Prediction #= MinValue)))
245    )
246 ;% no influencor -> Value decreases naturally.
247    (fd_sup(PreviousExpr,Sup), Sup > MinValue,
       ↪ param(defaultinhibitionrate, Inhibition),
248     Prediction #= PreviousExpr - Inhibition; Prediction #= MinValue)
249 ).
250
251 from(_, _, [], _, _, _, [], _).
252 from(FromId, Exps, [CurrentTargets|RestTargets], MaxLength, MinWeight,
    ↪ MaxWeight, Nodes, WeightHypothesis) :-
253     CurrentTargets ins MinWeight..MaxWeight,
254     NextFromId is FromId + 1,
255     from_to(FromId, 1, Exps, CurrentTargets, MaxLength, TargetNodes,
        ↪ WeightHypothesis),
256     from(NextFromId, Exps, RestTargets, MaxLength, MinWeight, MaxWeight,
        ↪ Ns, WeightHypothesis),
257     append(TargetNodes, Ns, Nodes).
258
259 from_exp_to(_,_,_,[],_).
260 from_exp_to(FromId, TargetId, Exps, [Target|Targets], Relationships) :-
261     nth1(TargetId, [Target|Targets], Weight),
262     check_in(edge(FromId, TargetId, Weight, Weight, Label), Exps, _),
263     append([edge(FromId, TargetId, Weight, Weight, Label)], [],
        ↪ Relationships).
264
265 from_hyp_to(FromId, TargetId, [Target|Targets], Relationships,
    ↪ WeightHypothesis) :-
266     member(edge(FromId, TargetId, MinWeight, MaxWeight, Label),
        ↪ WeightHypothesis),
267     param(minrequiredweight, MinRequiredWeight), MinWeight >=
        ↪ MinRequiredWeight,
```

```
268     nth1(TargetId, [Target|Targets], Weight),
269     (Weight in MinWeight..MaxWeight),
270     append([edge(FromId, TargetId, MinWeight, MaxWeight, Label)], [],
        ↪ Relationships).
271
272 from_to(_,_,_,[],_, [], _).
273 from_to(FromId, TargetId, Exps, [Target|Targets], MaxLength, Nodes,
    ↪ WeightHypothesis) :- (
274     TargetId =< MaxLength, FromId =< MaxLength,
275     (from_hyp_to(FromId, TargetId, [Target|Targets], CurrentNodes,
        ↪ WeightHypothesis)
276      ;from_exp_to(FromId, TargetId, Exps, [Target|Targets],
         ↪ CurrentNodes)
277      ;true
278     ), !,
279     NextTargetId is TargetId + 1,
280     from_to(FromId, NextTargetId, Exps, [Target|Targets], MaxLength, Ns,
        ↪ WeightHypothesis),
281     append(CurrentNodes, Ns, Nodes)
282 );true.
283
284 check_in(edge(A,B, Weight, Weight, Label), Exps,
285   [exp(A,Step1,ValueA1),
286    exp(B,Step1,ValueB1),
287    exp(A,Step2,ValueA2),
288    exp(B,Step2,ValueB2)]) :-
289   member(exp(A,Step1,ValueA1), Exps),
290   member(exp(B,Step1,ValueB1), Exps),
291   Step2 is Step1 + 1,
292   A \== B,
293   member(exp(A,Step2,ValueA2), Exps),
294   member(exp(B,Step2,ValueB2), Exps),
295   ValueA2 \== ValueA1,
296   ValueB2 \== ValueB1,
297   Diff1 is ValueA2 - ValueA1,
298   Diff2 is ValueB2 - ValueB1,
299   Diff2 \== Diff1,
300   abs(Diff1,Diff1Abs), abs(Diff2,Diff2Abs),
301   Diff1Abs < Diff2Abs,
```

```
302   computeWeight(Diff1, Diff2, ComputedWeight, Label),
303   Weight #= ComputedWeight.
304
305 computeWeight(Diff1, Diff2, WeightAbs,Label) :-
306   Diff2 \== Diff1, Diff2 \== 0, Diff1 \==0,
307   abs(Diff1,Diff1Abs), abs(Diff2,Diff2Abs),
308   Diff1Abs < Diff2Abs,
309   (   (   Diff1 > 0, Diff2 < 0, Label = '-');
310   (   Diff1 > 0, Diff2 > 0, Label = '+');
311   (   Diff1 < 0, Diff2 < 0, Label = '+');
312   (   Diff1 < 0, Diff2 > 0, Label = '-')), !,
313   Weight is (Diff2 - Diff1), abs(Weight, WeightAbs).
```

A.5 *Lac* Operon parameters

```
def(1, 'lacZYA mRNA'). def(2, 'LacI'). def(3, 'LacZ').
def(4, 'LacY'). def(5, 'lactoseInt'). def(6, 'lactoseExt').
def(7, 'allolactose'). def(8, 'cAMP'). def(9, 'lac operon').
def(10, 'lacZYA').
experiences([
  exp(1,1,0), % lacZYA mRNA
  exp(2,1,0), % lacI
  exp(3,1,0), % lacZ
  exp(4,1,0), % lacY
  exp(5,1,0), % lactoseInt
  exp(6,1,20), % lactoseExt,
  exp(6,10,20), % lactoseExt
  exp(7,1,0), % allolactose
  exp(8,1,15), % cAMP,
  exp(8,10,15), % cAMP
  exp(9,1,0), % lac operon
  exp(10,1,15) % lacZYA
]).
weightHypothesis([]).
rules([
  activate([1], [3,4], 1), % lacZYA mRNA -> ++ lacZ, lacY
  inhibate([2], [1], 3), % lacI -> -- lacZYA mRNA
  activate([8,10],[1], 1), % cAMP, lacZYA -> ++ lacZYA mRNA
  activate([6,4],  [5], 1), % lactoseExt, lacY -> ++ lactoseInt
    ↪ (simplified view)
  inhibate([6,4],  [6], 1), % lactoseExt, lacY -> -- lactoseInt
    ↪ (simplified view)
  activate([5,3],  [7], 1), % lactoseInt, lacZ -> ++ allolactose
  activate([7], [2], 1), % allolactose -> ++ lacI,
  activate([3,4],  [9], 1) % lacZ, lacY -> ++ lac operon
]).

param(minlength,1).
param(maxtime, 15).
param(minweight,0).
param(minrequiredweight,1).
```

```
35 param(minactivatorweight,1).
36 param(maxweight,4).
37 param(minoccur, 3).
38 param(weightsfromrules, 1).
39 param(minvalue, 0).
40 param(maxvalue, 20).
41 param(defaultinhibitionrate,0).
42 param(defaultactivationrate,1).
43 param(minrequiredexpressionlevel,1).
```

A.6 BioNet - Java source code

A.6.1 REST Provider to serve BioNet configuration files

Part of the code used to serve BioNet configuration files to the client via the REST API, in the JSON format.

```
1  package be.unamur.bionets.rest.provider;
2
3  import javax.annotation.security.PermitAll;
4  import javax.ws.rs.Consumes;
5  import javax.ws.rs.GET;
6  import javax.ws.rs.POST;
7  import javax.ws.rs.Path;
8  import javax.ws.rs.PathParam;
9  import javax.ws.rs.Produces;
10 import javax.ws.rs.core.MediaType;
11
12 import org.apache.commons.lang3.StringEscapeUtils;
13 import org.slf4j.Logger;
14 import org.slf4j.LoggerFactory;
15
16 import be.unamur.bionets.quizz.BioNetsException;
17 import be.unamur.bionets.rest.Response;
18 import be.unamur.bionets.rest.model.Query;
19 import be.unamur.bionets.swipl.SwiplFacade;
```

```
20
21 /**
22  * @author Morgan Wattiez
23  */
24 @Path("/prolog")
25 @PermitAll
26 @Consumes(MediaType.APPLICATION_JSON)
27 @Produces(MediaType.APPLICATION_JSON)
28 public class Prolog {
29
30    private static final Logger LOG =
    ↪  LoggerFactory.getLogger(Prolog.class);
31    @GET
32    @Path("file/{program}")
33    public Response getJobConfigurations(@PathParam("program") final
    ↪  String program) throws BioNetsException {
34       final String message = "received request to access Prolog program
       ↪  %s";
35       LOG.info(String.format(message, program));
36       final String programContent =
       ↪  SwiplFacade.getSwipl().getProgram(program);
37       return new Response(programContent);
38    }
39
40    @POST
41    @Path("save/{program}")
42    public Response saveProgram(@PathParam("program") final String
    ↪  program, final String source) throws BioNetsException {
43       final String message = "received request to save Prolog program %s
       ↪  with source %s";
44       final String formattedSource =
       ↪  StringEscapeUtils.escapeJson(source);
45
46       LOG.info(String.format(message, program, formattedSource));
47       final String programContent =
       ↪  SwiplFacade.getSwipl().saveCode(program,
       ↪  formatProgramSource(source.replaceAll("\\n",
       ↪  System.getProperty("line.separator"))));
```

```
48        final String revertFormatProgram =
          ↪ revertSourceFormat(programContent);
49        return new Response(revertFormatProgram);
50    }
51
52    // ... some utility methods
53
54    @POST
55    @Path("exec/{program}")
56    public Response execProgram(@PathParam("program") final String
      ↪ program, final Query query) throws BioNetsException {
57        final String message = "received request to exec Prolog program %s
          ↪ with query %s";
58        LOG.info(String.format(message, program, query));
59        final String solution =
          ↪ SwiplFacade.getSwipl().executeQuery(program,
          ↪ query.getCommand(), query.getKeys().toArray(new String[0]));
60        return new Response(solution);
61    }
62 }
```

A.6.2 SWI-Prolog initializer

Part of the code used to invoke SWI-Prolog via JPL.

```
1  /**
2   * @author Morgan Wattiez
3   */
4  public class Swipl {
5
6     private static final Logger LOG =
       ↪  LoggerFactory.getLogger(Swipl.class);
7
8     static Map<String, Query> queries = new HashMap<>();
9     static Query cachedQuery = null;
10    static Query bionet = null;
11    static boolean enableCache = true;
12    static String cachedQueryName = null;
13    static volatile Integer runningQueries = 0;
14    Swipl() {}
15
16    private Query programQuery(String program, final String mode, final
       ↪  boolean cache) throws BioNetsException {
17       Query query = null;
18       LOG.debug(String.format("[%d] Received %s query for file %s, with
          ↪  cache mode = %s", Thread.currentThread().getId(), program,
          ↪  mode, Boolean.toString(cache)));
19       if (mode.equals("unload_file") || cache && cachedQueryName != null
          ↪  && !cachedQueryName.equals(program)) {
20          clearCachedQuery();
21       }
22       if (bionet == null) {
23          loadBionet();
24       }
25       if (!isCached(program)) {
26          final URI path = getProgramPath(program);
27          if (Files.exists(Paths.get(path))) {
28             LOG.debug(String.format("[%d] Searching file %s at %s",
                ↪  Thread.currentThread().getId(), program, path));
```

```
29              query = consultQuery(mode, path);
30              query.oneSolution();
31              if (!cache) query.close();
32              if (cache) cacheQuery(program, query);
33              LOG.debug(String.format("program %s loaded from file %s",
                ↪  program, path));
34          } else {
35              throw new BioNetsException(String.format("program %s has not
                ↪  been found", program));
36          }
37      } else if (cache) {
38          LOG.debug(String.format("retrieved query %s from cache",
            ↪  program));
39          query = queries.get(program);
40      }
41      return query;
42  }
43
44  private void loadBionet() throws BioNetsException {
45      final Query queryClpFD = new Query("use_module(library(clpfd)).");
46      queryClpFD.oneSolution();
47      queryClpFD.close();
48      final URI bionetEnginePath = getProgramPath("bionet.pl");
49      if (Files.exists(Paths.get(bionetEnginePath))) {
50          LOG.debug(String.format("[%d] Loading BioNet Prolog file %s at
            ↪  %s", Thread.currentThread().getId(), "bionet.pl",
            ↪  bionetEnginePath));
51          final Query bionetloadingQuery = consultQuery("consult",
            ↪  bionetEnginePath);
52          bionetloadingQuery.oneSolution();
53          bionetloadingQuery.close();
54          bionet = bionetloadingQuery;
55      } else {
56          throw new BioNetsException(String.format("program %s has not
            ↪  been found", "bionet.pl"));
57      }
58  }
59
60  private Query consultQuery(final String mode, final URI path) {
```

```
61      Query query;
62      final Term[] arg = {
63      /**
64       * Please be careful with the choice of Apache Tomcat installation
    ↪  path
65       * : The following statements will fail if the deployment path
    ↪  contains
66       * any white space or unattended character
67       */
68        new Atom(new File(path).toString())
69      };
70      query = new Query(mode, arg);
71      return query;
72    }
73
74    public synchronized String getProgram(final String program) throws
      ↪  BioNetsException {
75      LOG.debug(String.format("[%d] entering getProgram with parameter
        ↪  program=%s", Thread.currentThread().getId(), program));
76      if (StringUtils.isBlank(program)) {
77        LOG.error(String.format("Invalid program's name '%s'",
          ↪  program));
78        return "";
79      }
80      final URI path = getProgramPath(program);
81      try {
82        if (Files.exists(Paths.get(path))) {
83          return FileUtils.readFileToString(new File(path));
84        } else {
85          throw new BioNetsException(String.format("program %s has not
            ↪  been found", program));
86        }
87      } catch (final IOException ex) {
88        LOG.error(String.format("Unable to load program %s at %s",
          ↪  program, path), ex);
89        throw new BioNetsException(String.format("Unable to load
          ↪  program %s", program));
90      }
91    }
```

```
92    // ...
93 }
```

A.7 BioNet - Java dependencies

The Java dependencies for BioNet Eclipse project are automatically collected by the build automation Gradle. Below, the gradle configuration file used in BioNet to collect the Java libraries required in the classpath.

```
1  apply plugin: 'java'
2  apply plugin: 'eclipse'
3  apply plugin: 'war'
4
5  sourceCompatibility = 1.7
6  version = '1.0'
7  jar {
8      manifest {
9          attributes 'Implementation-Title': 'Gradle Quickstart',
           ↪  'Implementation-Version': version
10     }
11 }
12 repositories {
13    flatDir {
14        dirs 'libs'
15    }
16 }
17 repositories { mavenCentral() }
18 dependencies {
19     testCompile 'org.testng:testng:6.8.8'
20     testCompile 'org.jbehave:jbehave-core:3.9.2'
21    compile 'com.google.code.gson:gson:2.2.4'
22     compile
       ↪  'org.glassfish.jersey.containers:jersey-container-servlet-core:2.8'
23     compile 'javax.servlet:javax.servlet-api:3.0.1'
24     testCompile('org.glassfish.jersey.test-framework.providers:
       ↪  jersey-test-framework-provider-external:2.9.1') {
```

```
25          exclude module: 'javax.servlet-api'
26     }
27    compile 'commons-io:commons-io:2.4'
28     compile 'org.apache.commons:commons-lang3:3.3.2'
29     compile 'log4j:log4j:1.2.17'
30    compile 'org.slf4j:slf4j-api:1.7.7'
31     compile 'org.slf4j:slf4j-log4j12:1.7.7'
32     compile fileTree(dir: 'libs', include: '*.jar')
33 }
34 test { useTestNG() }
35 buildscript {
36     repositories {
37         mavenCentral()
38         maven { url 'http://dl.bintray.com/robfletcher/gradle-plugins' }
39     }
40     dependencies {
41         classpath 'org.gradle.plugins:gradle-compass:1.0.7'
42     }
43 }
```

A.8 BioNet - JavaScript source code

A.8.1 Configuration editor

Part of the JavaScript code used to instantiate the BioNet configuration editor :

```
1 var initEditor = function(programName) {
2    ace.require("ace/ext/language_tools");
3    var editor = ace.edit("editor");
4    editor.setTheme("ace/theme/cobalt");
5    editor.getSession().setMode("ace/mode/prolog");
6    editor.setOptions({
7       enableBasicAutocompletion: true,
8       wrap: 140
9    });
10   $.ajax({
```

```
        url: '/BioNets/rest/prolog/file/' + programName,
        cache: false,
        dataType: 'json'
    }).success(function(data){
        editor.setValue(data.message);
    }).error(function(jqXHR, status, err) {
        alert('unable to find program ' + programName);
    });
}
```

A.8.2 Network visualizer

Part of the JavaScript code used to instantiate the BioNet network visualizer :

```
function refreshNetwork(configuration) {
    destroyNetwork();
    for (nodeKey in configuration.nodes) {
        var node = configuration.nodes[nodeKey];
        nodes.push({
            id: node.label,
            label: node.label,
            value: node.value
        });
    }
    for (edgeKey in configuration.edges) {
        var edge = configuration.edges[edgeKey];
        edges.push({
            from: edge.from.label,
            to: edge.to.label,
            value: edge.weight,
            label: '+' === edge.label? 'activates (+' + edge.weight + ')' :
            ↪  'inhibits (-' + edge.weight + ')',
            labelAlignment:'line-center',
            style: 'arrow',
            color: '+' === edge.label? 'red' : 'blue',
            length: 220,
            width: 1,
```

```
23          title: edge.from.label + ' ' + ('+' === edge.label?
            ↪  'activates':'inhibits') + ' '+ edge.to.label
24      });
25    }
26 }
```

Bibliography

[1] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J. Theor. Biol.*, 22(3):437–467, Mar 1969.

[2] S. Kauffman. Homeostasis and differentiation in random genetic control networks. *Nature*, 224(5215):177–178, Oct 1969.

[3] R. Thomas. Boolean formalization of genetic control circuits. *J. Theor. Biol.*, 42 (3):563–585, Dec 1973.

[4] R. Thomas. *Kinetic Logic: A Boolean Approach to the Analysis of Complex Regulatory Systems*, volume 29 Lecture Notes in Biomathematics. Springer-Verlag, 1979.

[5] Adrien Richard, Jean-Paul Comet, and Gilles Bernot. R. thomas' logical method, 2000.

[6] H. de Jong. Modeling and simulation of genetic regulatory systems: a literature review. *J. Comput. Biol.*, 9(1):67–103, 2002.

[7] Vladimir Filkov. *Handbook of Computational Molecular Biology*, chapter Identifying Gene Regulatory Networks from Gene Expression Data, pages 27/1–27/30. Chapman & Hall/CRC Computer and Information Science Series, 2005.

[8] Joxan Jaffar and J-L Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119. ACM, 1987.

[9] Joxan Jaffar and Michael J Maher. Constraint logic programming: A survey. *The journal of logic programming*, 19:503–581, 1994.

[10] Peter Stuckey Kimbal Marriott. *Programming with Constraints: An Introduction.* MIT, 1998. ISBN 0262133415,9780262133418.

[11] Peter Raven, George Johnson, Kenneth Mason, Jonathan Losos, and Susan Singer. *Biology.* McGraw-Hill Science/Engineering/Math, 2010. ISBN 0077350022.

[12] Robert J. Brooker. *Concepts of Genetics*. McGraw-Hill Science/Engineering/-Math, 1 edition, 1 2011. ISBN 9780073525334.

[13] Leland Hartwell, Leroy Hood, Michael Goldberg, Ann Reynolds, and Lee Silver. *Genetics: From Genes to Genomes (Hartwell, Genetics)*. McGraw-Hill Science/Engineering/Math, 2010. ISBN 007352526X.

[14] Daniel L. Hartl and Maryellen Ruvolo. *Genetics: Analysis of Genes and Genomes*. Jones & Bartlett Learning, 2011. ISBN 1449635962.

[15] The genetic code, 2014. URL `http://fr.slideshare.net/mdonohue/dna-power-point-final-8009638`. [Accessed January 6th, 2015].

[16] Jocelyn Krebs. *Lewin's genes XI*. Jones & Bartlett Learning, Burlington, Mass, 2014. ISBN 9781449659851.

[17] Sandy B. Primrose and Richard Twyman. *Principles of Gene Manipulation and Genomics*. Blackwell, 2006. ISBN 1405135441.

[18] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921, Feb 2001.

[19] D. C. Weaver, C. T. Workman, and G. D. Stormo. Modeling regulatory networks with weight matrices. *Pac Symp Biocomput*, pages 112–123, 1999.

[20] Roberto Tamassia. *Handbook of Graph Drawing and Visualization (Discrete Mathematics and Its Applications)*. Chapman & Hall/CRC, 2007. ISBN 1584884126.

[21] Natarajan et al. nvasion-network, 2016. URL `http://www.biomedcentral.com/1471-2105/7/373/figure/F4`. [Accessed January 26th, 2015].

[22] M. Kanehisa and S. Goto. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Res.*, 28(1):27–30, Jan 2000.

[23] M. Kanehisa, S. Goto, Y. Sato, M. Kawashima, M. Furumichi, and M. Tanabe. Data, information, knowledge and principle: back to metabolism in KEGG. *Nucleic Acids Res.*, 42(Database issue):199–205, Jan 2014.

[24] Kanehisa Laboratories. Kegg genes database, 2014. URL `http://www.kegg.jp/dbget-bin/www_bfind?genes`. [Accessed December 29th, 2014].

[25] Kanehisa Laboratories. Kegg species list, 2014. URL `http://www.genome.jp/kegg/catalog/org_list4.html`. [Accessed December 29th, 2014].

[26] G. Sherlock, H. Boussard, A. Kasarskis, G. Binkley, J. C. Matese, S. S. Dwight, M. Kaloper, S. Weng, H. Jin, C. A. Ball, M. B. Eisen, P. T. Spellman, P. O. Brown, D. Botstein, and J. M. Cherry. The Stanford Microarray Database. *Nucleic Acids Res*, 29(1):152–5, 2001.

[27] Ingrid M. Keseler, Amanda Mackie, Martin Peralta-Gil, Alberto Santos-Zavaleta, Socorro Gama-Castro, César Bonavides-Martínez, Carol Fulcher, Araceli M. Huerta, Anamika Kothari, Markus Krummenacker, Mario Latendresse, Luis Muñiz Rascado, Quang Ong, Suzanne Paley, Imke Schröder, Alexander G. Shearer, Pallavi Subhraveti, Mike Travers, Deepika Weerasinghe, Verena Weiss, Julio Collado-Vides, Robert P. Gunsalus, Ian Paulsen, and Peter D. Karp. EcoCyc: fusing model organism databases with systems biology. *Nucleic Acids Research*, 41(D1):D605–D612, January 2013. ISSN 1362-4962. doi: 10.1093/nar/gks1027. URL `http://dx.doi.org/10.1093/nar/gks1027`.

[28] J. M. Cherry, C. Adler, C. Ball, S. A. Chervitz, S. S. Dwight, E. T. Hester, Y. Jia, G. Juvik, T. Roe, M. Schroeder, and Others. SGD: Saccharomyces Genome Database. *Nucleic Acids Research*, 26(1):73–79, 1998.

[29] Hanhae Kim, Junha Shin, Eiru Kim, Hyojin Kim, Sohyun Hwang, Jung Eun E. Shim, and Insuk Lee. YeastNet v3: a public database of data-specific and integrated functional gene networks for Saccharomyces cerevisiae. *Nucleic acids research*, October 2013. ISSN 1362-4962. doi: 10.1093/nar/gkt981. URL `http://www.inetbio.org/yeastnet/`.

[30] Chris Stark, Bobby-Joe J. Breitkreutz, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, and Mike Tyers. BioGRID: a general repository for interaction datasets. *Nucleic acids research*, 34(Database issue):D535–D539, January 2006. ISSN 1362-4962. doi: 10.1093/nar/gkj109. URL `http://thebiogrid.org/`.

[31] Gary D. Bader, Doron Betel, and Christopher W. V. Hogue. BIND: the Biomolecular Interaction Network Database. *Nucleic Acids Research*, 31(1):248–250, January 2003. ISSN 1362-4962. doi: 10.1093/nar/gkg056. URL `http://dx.doi.org/10.1093/nar/gkg056`.

[32] Ioannis Xenarios, Danny W. Rice, Lukasz Salwinski, Marisa K. Baron, Edward M. Marcotte, and David Eisenberg. DIP: the Database of Interacting Proteins. *Nucl. Acids Res.*, 28(1):289–291, January 2000. doi: 10.1093/nar/28.1.289. URL `http://dx.doi.org/10.1093/nar/28.1.289`.

[33] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald,

G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature genetics*, 25(1):25–29, May 2000. ISSN 1061-4036. doi: 10.1038/75556. URL `http://geneontology.org/`.

[34] R. Overbeek, N. Larsen, T. Walunas, M. D'Souza, G. Pusch, E. Selkov, K. Liolios, V. Joukov, D. Kaznadzey, I. Anderson, A. Bhattacharyya, H. Burd, W. Gardner, P. Hanke, V. Kapatral, N. Mikhailova, O. Vasieva, A. Osterman, V. Vonstein, M. Fonstein, N. Ivanova, and N. Kyrpides. The ERGO genome analysis and discovery system. *Nucleic Acids Res*, 31(1):164–71+. URL `http://www.igenbio.com/ergo`.

[35] Atanas Kamburov, Ulrich Stelzl, Hans Lehrach, and Ralf Herwig. The ConsensusPathDB interaction database: 2013 update. *Nucleic acids research*, 41(Database issue):D793–D800, January 2013. ISSN 1362-4962. doi: 10.1093/nar/gks1055. URL `http://dx.doi.org/10.1093/nar/gks1055`.

[36] Atanas Kamburov, Christoph Wierling, Hans Lehrach, and Ralf Herwig. ConsensusPathDB—a database for integrating human functional interaction networks. *Nucleic Acids Research*, 37(suppl 1):D623–D628, January 2009. ISSN 1362-4962. doi: 10.1093/nar/gkn698. URL `http://dx.doi.org/10.1093/nar/gkn698`.

[37] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res.*, 13(11):2498–2504, Nov 2003. URL `http://www.cytoscape.org/`.

[38] Zhenjun Hu, Jui-Hung Hung, Yan Wang, Yi-Chien Chang, Chia-Ling Huang, Matt Huyck, and Charles DeLisi. VisANT 3.5: multi-scale network visualization, analysis and inference based on the gene ontology. *Nucleic Acids Research*, 37(suppl 2):W115–W121, July 2009. ISSN 1362-4962. doi: 10.1093/nar/gkp406. URL `http://visant.bu.edu/`.

[39] Pathway Studio: Software for Visualization and Analysis of Biological Pathways. URL `http://www.elsevier.com/online-tools/pathway-studio`.

[40] C. A. H. Baker, M. S. T. Carpendale, P. Prusinkiewicz, and M. G. Surette. GeneVis: visualization tools for genetic regulatory network dynamics. In *Visualization, 2002. VIS 2002. IEEE*, pages 243–250. IEEE, November 2002. ISBN 0-7803-7498-3. doi: 10.1109/visual.2002.1183781. URL `http://www.win.tue.nl/~mwestenb/genevis/`.

[41] E. Demir, O. Babur, U. Dogrusoz, A. Gursoy, G. Nisanci, R. Cetin-Atalay, and M. Ozturk. Patika: an integrated visual environment for collaborative construction

and analysis of cellular pathways. *Bioinformatics*, 18(7):996–1003, July 2002. ISSN 1460-2059. doi: 10.1093/bioinformatics/18.7.996. URL `http://www.patika.org/software/`.

[42] Florian Iragne, Macha Nikolski, Bertrand Mathieu, David Auber, and David Sherman. ProViz: protein interaction visualization and exploration. *Bioinformatics*, 21 (2):272–274, January 2005. ISSN 1460-2059. doi: 10.1093/bioinformatics/bth494. URL `http://sourceforge.net/projects/proviz/`.

[43] M. Suderman and M. Hallett. Tools for visually exploring biological networks. *Bioinformatics*, 23(20):2651–2659, Oct 2007.

[44] C. Wierling, R. Herwig, and H. Lehrach. Resources, standards and tools for systems biology. *Brief Funct Genomic Proteomic*, 6(3):240–251, Sep 2007.

[45] J.M. Bower and H. Bolouri. *Computational Modeling of Genetic and Biochemical Networks*. Bradford Books. ISBN 9780262524230.

[46] I. M. Keseler, A. Mackie, M. Peralta-Gil, A. Santos-Zavaleta, S. Gama-Castro, C. Bonavides-Martinez, C. Fulcher, A. M. Huerta, A. Kothari, M. Krummenacker, M. Latendresse, L. Muniz-Rascado, Q. Ong, S. Paley, I. Schroder, A. G. Shearer, P. Subhraveti, M. Travers, D. Weerasinghe, V. Weiss, J. Collado-Vides, R. P. Gunsalus, I. Paulsen, and P. D. Karp. EcoCyc: fusing model organism databases with systems biology. *Nucleic Acids Res.*, 41(Database issue):D605–612, Jan 2013. URL `http://ecocyc.org/`.

[47] I. M. Keseler, J. Collado-Vides, A. Santos-Zavaleta, M. Peralta-Gil, S. Gama-Castro, L. Muniz-Rascado, C. Bonavides-Martinez, S. Paley, M. Krummenacker, T. Altman, P. Kaipa, A. Spaulding, J. Pacheco, M. Latendresse, C. Fulcher, M. Sarker, A. G. Shearer, A. Mackie, I. Paulsen, R. P. Gunsalus, and P. D. Karp. EcoCyc: a comprehensive database of Escherichia coli biology. *Nucleic Acids Res.*, 39(Database issue):D583–590, Jan 2011.

[48] P. D. Karp, M. Riley, S. M. Paley, and A. Pelligrini-Toole. EcoCyc: an encyclopedia of Escherichia coli genes and metabolism. *Nucleic Acids Res.*, 24(1):32–39, Jan 1996.

[49] H. Parkinson, M. Kapushesky, M. Shojatalab, N. Abeygunawardena, R. Coulson, A. Farne, E. Holloway, N. Kolesnykov, P. Lilja, M. Lukk, R. Mani, T. Rayner, A. Sharma, E. William, U. Sarkans, and A. Brazma. ArrayExpress–a public database of microarray experiments and gene expression profiles. *Nucleic Acids Res.*, 35(Database issue):D747–750, Jan 2007.

[50] Y. Ye and T. G. Doak. A parsimony approach to biological pathway reconstruction/inference for genomes and metagenomes. *PLoS Comput. Biol.*, 5(8):e1000465, Aug 2009.

[51] R. Khanin and E. Wit. How scale-free are biological networks. *J. Comput. Biol.*, 13(3):810–818, Apr 2006.

[52] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. The evolutionary origins of modularity. *Proceedings of the Royal Society of London B: Biological Sciences*, 280 (1755):20122863, 2013.

[53] H. H. McAdams and A. Arkin. Simulation of prokaryotic genetic circuits. *Annu Rev Biophys Biomol Struct*, 27:199–224, 1998.

[54] M. I. Arnone and E. H. Davidson. The hardwiring of development: organization and function of genomic regulatory systems. *Development*, 124(10):1851–1864, May 1997.

[55] T. Fournier, J. P. Gabriel, C. Mazza, J. Pasquier, J. L. Galbete, and N. Mermod. Steady-state expression of self-regulated genes. *Bioinformatics*, 23(23):3185–3192, Dec 2007.

[56] T. C. Freeman, L. Goldovsky, M. Brosch, S. van Dongen, P. Maziere, R. J. Grocock, S. Freilich, J. Thornton, and A. J. Enright. Construction, visualisation, and clustering of transcription networks from microarray expression data. *PLoS Comput. Biol.*, 3(10):2032–2042, Oct 2007.

[57] Y. Y. Ho, L. Cope, M. Dettling, and G. Parmigiani. Statistical methods for identifying differentially expressed gene combinations. *Methods Mol. Biol.*, 408: 171–191, 2007.

[58] J. J. Rice, Y. Tu, and G. Stolovitzky. Reconstructing biological networks using conditional correlation analysis. *Bioinformatics*, 21(6):765–773, Mar 2005.

[59] Yiming Zuo and Mahlet G Tadesse. Reconstructing biological networks using low order partial correlation. 2013.

[60] L. F. Wessels, E. P. van Someren, and M. J. Reinders. A comparison of genetic network models. *Pac Symp Biocomput*, pages 508–519, 2001.

[61] G. Bernot, J.-P. Cormet, A. Richard, M. Chaves, J.-L. Gouzé, and F. Dayan. Modeling and analysis of gene regulatory networks. *Modeling in Computational Biology and Biomedicine: A Multidisciplinary Endeavor*, pages 47–80, 2013.

[62] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian networks to analyze expression data. *J. Comput. Biol.*, 7(3-4):601–620, 2000.

[63] Kevin Murphy, Saira Mian, et al. Modelling gene expression data using dynamic bayesian networks. Technical report, 1999.

[64] A. J. Hartemink, D. K. Gifford, T. S. Jaakkola, and R. A. Young. Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. *Pac Symp Biocomput*, pages 422–433, 2001.

[65] E. J. Moler, D. C. Radisky, and I. S. Mian. Integrating naive Bayes models and external knowledge to examine copper and iron homeostasis in S. cerevisiae. *Physiol. Genomics*, 4(2):127–135, Dec 2000.

[66] T. Mestl, E. Plahte, and S. W. Omholt. A mathematical framework for describing and analysing gene regulatory networks. *J. Theor. Biol.*, 176(2):291–300, Sep 1995.

[67] René Thomas and Richard D'Ari. Biological Feedback. *CRC Press, Inc.*, 1990.

[68] A. Garg, A. Di Cara, I. Xenarios, L. Mendoza, and G. De Micheli. Synchronous versus asynchronous modeling of gene regulatory networks. *Bioinformatics*, 24 (17):1917–1925, Sep 2008.

[69] Nedumparambathmarath Vijesh, Swarup Kumar Chakrabarti, Janardanan Sreekumar, et al. Modeling of gene regulatory networks: A review. *Journal of Biomedical Science and Engineering*, 6(02):223, 2013.

[70] G. Batt, M. Page, I. Cantone, G. Gössler, P.T. Monteiro, and H. de Jong. Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. *Bioinformatics*, 26(18), 2010.

[71] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. Hypergraphs and cellular networks. *PLoS Comput Biol*, 5(5):e1000385, 05 2009. doi: 10.1371/journal.pcbi.1000385. URL `http://dx.doi.org/10.1371%2Fjournal.pcbi.1000385`.

[72] Ting Chen, Vladimir Filkov, and Steven Skiena. Identifying gene regulatory networks from experimental data. *Parallel Computing*, 27(1-2):141–162, 2001. URL `http://dblp.uni-trier.de/db/journals/pc/pc27.html#ChenFS01`.

[73] Tatsuya Akutsu, Satoru Kuhara, Osamu Maruyama, and Satoru Miyano. Identification of gene regulatory networks by strategic gene disruptions and gene overexpressions. pages 695–702, 1998. URL `http://dl.acm.org/citation.cfm?id=314613.315050`.

[74] R. Franke, F. J. Theis, and S. Klamt. From binary to multivalued to continuous models: the lac operon as a case study. *J Integr Bioinform*, 7(1), 2010.

[75] René Thomas. Regulatory networks seen as asynchronous automata : a logical description. *J. Theor. Biol. 153*, 1991.

[76] Z. Khalis, G. Bernot, and J.-P. Comet. *Proc. of the Nice Spring school on Modelling complex biological systems in the context of genomics*, chapter Gene Regulatory Networks: Introduction of multiplexes into R. Thomas' modelling, pages 139–151. EDP Science, ISBN : 978-2-7598-0437-5, 2009.

[77] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang. Probabilistic Boolean Networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, Feb 2002.

[78] Huai Li, Jianhua Xuan, Yue Wang, and Ming Zhan. Inferring regulatory networks. *Front Biosci*, 13(263):75, 2008.

[79] H. Li, J. Whitmore, E. Suh, M. Bittner, and S. Kim. Learning context-sensitive boolean network from steady-state observations and its analysis. 2004.

[80] Ranadip Pal, Aniruddha Datta, Michael L. Bittner, and Edward R. Dougherty. Intervention in context-sensitive probabilistic boolean networks. *Bioinformatics*, 21(7):1211–1218, 2005. doi: 10.1093/bioinformatics/bti131.

[81] A. S. Ribeiro and J. Lloyd-Price. SGN Sim, a stochastic genetic networks simulator. *Bioinformatics*, 23(6):777–779, Mar 2007.

[82] A. S. Ribeiro and J. Lloyd-Price. Sgnsim ensemble generator manual, 2006. URL `http://www.cs.tut.fi/~sanchesr/SGN/manual_sgne.pdf`. [Accessed January 11th, 2015].

[83] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang. Probabilistic Boolean Networks: a rule-based uncertainty model for gene regulatory networks. *Bioinformatics*, 18(2):261–274, Feb 2002.

[84] Franziska Hinkelmann, Madison Brandon, Bonny Guang, Rustin McNeill, Grigoriy Blekherman, Alan Veliz-Cuba, and Reinhard Laubenbacher. Adam: analysis of discrete models of biological systems using computer algebra. *BMC bioinformatics*, 12(1):295, 2011.

[85] Elisabeth Remy, Brigitte Mossé, Claudine Chaouiya, and Denis Thieffry. A description of dynamical graphs associated to elementary regulatory circuits. In *ECCB*, pages 172–178, 2003. URL `http://dblp.uni-trier.de/db/conf/eccb/eccb2003.html#RemyMCT03`.

[86] Alexander Hartemink. Banjo: Bayesian Network Inference with Java Objects. URL `http://www.cs.duke.edu/~{}amink/software/banjo`.

[87] Thomas Schlitt and Alvis Brazma. Current approaches to gene regulatory network modelling. *BMC Bioinformatics*, 8(Suppl 6):S9, 2007.

[88] Hidde De Jong and Johannes Geiselmann. Modeling and simulation of genetic regulatory networks by ordinary differential equations.

[89] Katsumi Inoue. Logic programming for boolean networks. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 924–930. AAAI Press, 2011. ISBN 978-1-57735-514-4. doi: 10.5591/978-1-57735-516-8/IJCAI11-160. URL `http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-160`.

[90] P. Mendes. GEPASI: a software package for modelling the dynamics, steady states and control of biochemical and other systems. *Comput. Appl. Biosci.*, 9(5):563–571, Oct 1993.

[91] Enzymology group. Power law analysis and simulation, 2015. URL `http://enzymology.fc.ul.pt/software/plas/`. [Accessed February 29th, 2015].

[92] H. de Jong, J. Geiselmann, C. Hernandez, and M. Page. Genetic Network Analyzer: qualitative simulation of genetic regulatory networks. *Bioinformatics*, 19(3):336–344, Feb 2003.

[93] M. Carrillo, P. A. Gongora, and D. A. Rosenblueth. An overview of existing modeling tools making use of model checking in the analysis of biochemical networks. *Front Plant Sci*, 3:155, 2012.

[94] Pedro T. Monteiro, Wassim Abou-Jaoudé, Denis Thieffry, and Claudine Chaouiya. Model checking logical regulatory networks. In Jean-Jacques Lesage, Jean-Marc Faure, Jose E. R. Cury, and Bengt Lennartson, editors, *12th IFAC - IEEE International Workshop on Discrete Event Systems WODES'12 (ENS Cachan, France)*, pages 170–175, 2014.

[95] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NuSMV: A New Symbolic Model Verifier. In *CAV '99: Proceedings of the 11th International Conference on Computer Aided Verification*, pages 495–499, London, UK, 1999. Springer-Verlag. ISBN 3-540-66202-2. URL `http://portal.acm.org/citation.cfm?id=647768.733923`.

[96] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV 2: An OpenSource tool for symbolic

model checking. In *Proceeding of International Conference on Computer-Aided Verification (CAV 2002)*, Copenhagen, Denmark, July 2002.

[97] BIOCHAM web page. URL `http://contraintes.inria.fr/BIOCHAM/`.

[98] J.-P. Comet et A. Richard. Smbionet: a tool for modeling biological regulatory networks driven by temporal behavior. *none*, 2003. ECCB'03, Paris, France, September 27-30, 2003.

[99] H.A. Gabbar. *Modern Formal Methods and Applications*. Modern Formal Methods and Applications. Springer, 2006. ISBN 9781402042225. URL `http://books.google.be/books?id=rITfyjh0ipoC`.

[100] A. Gonzalez Gonzalez, A. Naldi, L. Sánchez, D. Thieffry, and C. Chaouiya. GINsim: a software suite for the qualitative modelling, simulation and analysis of regulatory networks. *Bio Systems*, 84(2):91–100, May 2006. ISSN 0303-2647. doi: 10.1016/j.biosystems.2005.10.003. URL `http://dx.doi.org/10.1016/j.biosystems.2005.10.003`.

[101] G. Batt, H. de Jong, Geiselmann J., and Page M. *Genetic Regulatory Networks : a Model-Checking Approach*.

[102] Grégory Batt, Delphine Ropers, Hidde De Jong, Johannes Geiselmann, Radu Mateescu, Michel Page, Dominique Schneider, et al. Analysis and verification of qualitative models of genetic regulatory networks: A model-checking approach. 2005.

[103] E. Jeong, M. Nagasaki, A. Saito, and S. Miyano. Cell system ontology: representation for modeling, visualizing, and simulating biological pathways. *In silico biology*, 7(6):623–638, 2007. ISSN 1386-6338. URL `http://view.ncbi.nlm.nih.gov/pubmed/18467775`.

[104] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, and J. Wang. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19 (4):524–531, Mar 2003.

[105] M Hucka, A Finney, S. Hoops, S. Keating, and N Le Novère. Systems biology markup language (sbml) level 2: structures and facilities for model definitions. *Nature Precedings*, 2007. doi: 10101/npre.2007.58.1.

[106] Catherine M. Lloyd, Matt D. Halstead, and Poul F. Nielsen. CellML: its future, present and past. *Progress in biophysics and molecular biology*, 85(2-3):433–450, July 2004. ISSN 0079-6107. doi: 10.1016/j.pbiomolbio.2004.01.004. URL `http://dx.doi.org/10.1016/j.pbiomolbio.2004.01.004`.

[107] Emek Demir, Michael P. Cary, Suzanne Paley, Ken Fukuda, Christian Lemer, Imre Vastrik, Guanming Wu, Peter D'Eustachio, Carl Schaefer, Joanne Luciano, Frank Schacherer, Irma Martinez-Flores, Zhenjun Hu, Veronica Jimenez-Jacinto, Geeta Joshi-Tope, Kumaran Kandasamy, Alejandra C. Lopez-Fuentes, Huaiyu Mi, Elgar Pichler, Igor Rodchenkov, Andrea Splendiani, Sasha Tkachev, Jeremy Zucker, Gopal Gopinath, Harsha Rajasimha, Ranjani Ramakrishnan, Imran Shah, Mustafa Syed, Nadia Anwar, Ozgun Babur, Michael Blinov, Erik Brauner, Dan Corwin, Sylva Donaldson, Frank Gibbons, Robert Goldberg, Peter Hornbeck, Augustin Luna, Peter Murray-Rust, Eric Neumann, Oliver Ruebenacker, Matthias Samwald, Martijn van Iersel, Sarala Wimalaratne, Keith Allen, Burk Braun, Michelle Whirl-Carrillo, Kei-Hoi Cheung, Kam Dahlquist, Andrew Finney, Marc Gillespie, Elizabeth Glass, Li Gong, Robin Haw, Michael Honig, Olivier Hubaut, David Kane, Shiva Krupa, Martina Kutmon, Julie Leonard, Debbie Marks, David Merberg, Victoria Petri, Alex Pico, Dean Ravenscroft, Liya Ren, Nigam Shah, Margot Sunshine, Rebecca Tang, Ryan Whaley, Stan Letovksy, Kenneth H. Buetow, Andrey Rzhetsky, Vincent Schachter, Bruno S. Sobral, Ugur Dogrusoz, Shannon McWeeney, Mirit Aladjem, Ewan Birney, Julio Collado-Vides, Susumu Goto, Michael Hucka, Nicolas Le Novere, Natalia Maltsev, Akhilesh Pandey, Paul Thomas, Edgar Wingender, Peter D. Karp, Chris Sander, and Gary D. Bader. The BioPAX community standard for pathway data sharing. *Nature Biotechnology*, 28(9):935–942, September 2010. ISSN 1087-0156. doi: 10.1038/nbt.1666. URL `http://dx.doi.org/10.1038/nbt.1666`.

[108] C. Wrzodek, A. Drager, and A. Zell. KEGGtranslator: visualizing and converting the KEGG PATHWAY database to various formats. *Bioinformatics*, 27(16):2314–2315, Aug 2011.

[109] Constraints Solving. Constraint solvers, 2015. URL `http://www.constraintsolving.com/solvers`. [Accessed March 3rd, 2015].

[110] Raymond Wisman Indiana University SE. Chapter 5 - constraint satisfaction problems, 2015. URL `http://homepages.ius.edu/RWISMAN/C463/html/Chapter5.htm`. [Accessed March 4th, 2015].

[111] Constraint programming for the dynamical analysis of biochemical systems – a survey. deliverable D1.6, ANR CALAMAR (ANR-08-SYSC-003), February 2011. URL `http://tagc.univ-mrs.fr/welcome/IMG/pdf/Livrable-1-6.pdf`.

[112] Martin Gebser, Arne König, Torsten Schaub, Sven Thiele, and Philippe Veber. The bioasp library: Asp solutions for systems biology. In *ICTAI (1)*, pages 383–389. IEEE Computer Society, 2010. ISBN 978-0-7695-4263-8. URL `http://dblp.uni-trier.de/db/conf/ictai/ictai2010-1.html#GebserKSTV10`.

[113] Damien Eveillard, Jonathan Fromentin, and Olivier Roux. Constraints Programming for Unifying Gene Regulatory Networks Modeling Approaches. In *Workshop on Constraint Based Methods for Bioinformatics (WCB08)*, page 10, Paris, France, 2008. URL `https://hal.archives-ouvertes.fr/hal-00277104`.

[114] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002. ISBN 0137903952.

[115] Paulo Moura, Paulo Moura, Ernesto Costa, and Ernesto Costa. Logtalk: Object-oriented programming in prolog, 1994.

[116] Zoltan Somogyi, Fergus James Henderson, and Thomas Charles Conway. The implementation of mercury, an efficient purely declarative logic programming language. In *IN PROCEEDINGS OF THE AUSTRALIAN COMPUTER SCIENCE CONFERENCE*, pages 499–512, 1995.

[117] Mats Carlsson, Johan Widen, Johan Andersson, Stefan Andersson, Kent Boortz, Hans Nilsson, and Thomas Sjöland. *SICStus Prolog user's manual*, volume 3. Swedish Institute of Computer Science, 1988.

[118] Afany Software. Comparison of prolog systems on cpu time., 2008. URL `http://www.picat-lang.org/bprolog/performance.htm`. [Accessed March 22nd, 2015].

[119] D. van Dijk, G. Ertaylan, C. A. Boucher, and P. M. Sloot. Identifying potential survival strategies of HIV-1 through virus-host protein interaction networks. *BMC Syst Biol*, 4:96, 2010.

[120] Kishore, Nand ; Balu, Radhakrishnan ; Karna, Shashi P. Modeling Genetic Regulatory Networks Using First-Order Probabilistic Logic. 2013.

[121] prism query@mi.cs.titech.ac.jp. Prism - programming in statistical modeling, 2015. URL `http://rjida.meijo-u.ac.jp/prism/`. [Accessed March 21st, 2015].

[122] Markus Triska. The finite domain constraint solver of SWI-Prolog. In *FLOPS*, volume 7294 of *LNCS*, pages 307–316, 2012.

[123] Anne Ogborn. Clp(fd) constraint logic programming over finite domains, 2015. URL `http://www.pathwayslms.com/swipltuts/clpfd/clpfd.html#_resources`. [Accessed April 6th, 2015].

[124] SWI-Prolog. library(clpfd): Constraint logic programming over finite domains, 2015. URL `http://www.swi-prolog.org/pldoc/man?section=clpfd`. [Accessed March 29th, 2015].

[125] F. Corblin, S. Tripodi, E. Fanchon, D. Ropers, and L. Trilling. A declarative constraint-based method for analyzing discrete genetic regulatory networks. *BioSystems*, 98(2):91–104, Nov 2009.

[126] Paul Singleton, Fred Dushin, and Jan Wielemaker. Jpl: A bidirectional prolog/java interface, 2001. URL `http://www.swi-prolog.org/packages/jpl/`. [Accessed February 14th, 2015].