FEDERICO II UNIVERSITY OF NAPLES

Ph.D. in COMPUTATIONAL BIOLOGY and BIOINFORMATICS

XXVI cycle

Mara Sangiovanni

Model Checking of Metabolic Networks: Application to Metabolic Diseases

Ph.D. Thesis

Advisor:
prof. Adriano Peron

Coordinator:
prof. Sergio Cocozza

Federico II University of Naples
Department of Electrical Engineering and Information Technology
Via Claudio 21, Naples
Italy

To Adriano and Emiliano,
my here and now, my strength.

The ancient covenant is in pieces;
man knows at last that he is alone in the universe's unfeeling immensity,
out of which he emerged only by chance.
His destiny is nowhere spelled out, nor is his duty.
The kingdom above or the darkness below: it is for him to choose.

Jacques Monod

Tu ne quaesieris, scire nefas, quem mihi, quem tibi
finem di dederint, Leuconoe, nec Babylonios
temptaris numeros. Ut melius, quidquid erit, pati,
seu plures hiemes, seu tribuit Iuppiter ultimam,
quae nunc oppositis debilitat pumicibus mare
Tyrrhenum: sapias, vina liques, et spatio brevi
spem longam reseces. Dum loquimur, fugerit invida
aetas: carpe diem, quam minimum credula postero.

Q. Horatius Flaccus

Contents

1

Introduction

1.1 Towards an Holistic View of Biological Systems

The availability of large amounts of genome-scale *omics* data coming from high–throughput sequencing technologies was the driving force behind the rise of *Systems Biology*, a new paradigm that emerged in Life Sciences about fifteen years ago [1].

The need for a more complete and *whole-istic* [2] approach to the interpretation of this huge and complex amount of information drove a shift of perspective: from a traditional, reductionist approach, focused on the single biological components (such as nucleotides, genes, proteins) the attention was moved to an holistic, dynamic view of interacting networks of biological entities [3, 4]. The large collections of detailed omics data produced by molecular studies are essential to build complex hierarchical models: starting from basic elements (such as DNA, mRNA, proteins, and metabolites) several levels of growing complexity may be considered, that range from protein interactions, signaling pathways, and reaction networks to large-scale systems such as cells, tissues or whole organisms [4, 5].

However, the simple organization of data into networks is not enough to gain insight on the features of the system as a whole. As clearly outlined in the seminal paper of Kitano [6], a system-level understanding of an organism should address *i)* the *system structure*, i.e. its components and the way in which they are related, and *ii) the system dynamics*, i.e. how it behaves over time under various conditions. The reconstruction of the **behaviour in time and in space** of the whole system is central in Systems Biology, and is an essential step to unravel the relationships intercurring between the genotype and the phenotype of organisms [7].

To this aim, the biological information should be necessarily complemented by mathematical and/or computational models, either manually or automatically built: these models may be executed or simulated to reveal the adequacy of the biological assumptions and hypotheses on which they were based, in case leading to a refinement of the model and to new biological experiments. This *hypothesis-driven* research approach, and the cyclic process of information integration and *in silico* model building, is typical of the Systems Biology paradigm [4, 6].

1.2 Executable models for Systems Biology

Networks of interacting elements are crucial in Systems Biology, since they permit both an immediate representation of the elements forming the biological system, both an intuitively modeling of their interactions. However, the bare reconstruction of a network and the analysis of its topological features are not enough to understand the whole system behaviour. Moreover, these models are becoming increasingly large and complex, sometimes involving information at the whole genome scale or at multiple levels of detail.

Hence, adequate mathematical and computational tools are required for their analysis and simulation.

Mathematical models can precisely describe quantitative relationships between variables such as metabolite concentrations, or gene expression levels. However quantitative modeling is constrained both by the availability of biological precise data both by the number of variables and equations that may be effectively treated.

Instead, computational models may go beyond this limitation and work also on partial data, and on more complex systems. This qualitative modeling approach relies on the goodness of the underlying abstraction, rather than on the faithfulness of the mathematical implementation. The obtained results, although not precise, may give interesting hints on the features and behaviours of the modeled biological system. The differences between the computational and the biological models may be used to build new hypotheses, to refine the model, and even suggest new experiments that can help in validate or reject the model. This methodology, called by Fisher and Henzinger **executable biology** [8], is based on the close interplay between *in silico* simulations and biological validation of the data.

It should be noticed that in this repeated process of model building and refining, automated formal verification methods, such as the Model Checking, play a fundamental role. Model Checking is a prominent methodology developed for the automatic and exhaustive verification of concurrent hardware and software systems. Its strength relies on the abstract, mathematical approach that is used to address the problem of verifying if a system is compliant with the desired specifications. Both the system and the property to verify are translated into suitable mathematical structures, that are used to determine if the desired property holds in all the possible model computations, i.e. to check if the system is a model of the desired property. If not, a counterexample is generated.

Model checking techniques may be successfully applied to the analysis of biological networks, although a shift in the perspective should be performed [9]. In fact, in the hardware and software domains the verification process is used to ascertain that the system built from the model will have the desired behaviour. If the verification fails, the system is searched against errors (using the counterexamples generated by the checker) and eventually modified. In biology, the process is the opposite: the (biological) system already exists and is correct by default, and the model is built to verify that the hypothesis made on its functioning is able to catch some features of the underlying, partially unknown, mechanisms.

Indeed, Model Checking is a methodology that may fit very well the Systems Biology approach [10]. This is due to several reasons: *i)* it enforces model abstraction. Biological systems are usually very complicated, and abstraction is a crucial strategy to address this complexity; *ii)* it is able to deal with the concept of multi-level functionality that is fundamental in Systems Biology; *iii)* it offers precisely defined formal languages, with a semantic that permits to describe both the constituent components and the way in which they interact. The overall behaviour will emerge from the interplay of the components, a very close view to the Systems Biology paradigm; *iv)* it can infer both qualitative or quantitative properties of *all* possible executions of a model; *v)* it can give partial information through simulations of the model, i.e. a partial exploration of the execution set.

1.3 Motivations

Several successful applications of Model Checking techniques to biological problems have been proposed in literature. They can be classified depending on the different granularity that their semantics are able to capture, as proposed in Brim et al. [9]. The components of a system are typically interpreted as variables, that can be either discrete or continuous depending on the selected level of abstraction, the availability of data, and the model

dimensions. The interactions are interpreted as rules that specify the dynamical changes of the variables. The time in which these interactions take place may be modeled either as a dense, continuous entity or may be represented as a discrete quantity. Lastly, the execution of interactions may be stochastic, i.e. it may proceed with a certain degree of uncertainty that reflects the assumption of a noisy environment. We will describe in detail in chapter 4 a wide range of proposed methodologies encompassing all the possible combinations of the above described features.

In this thesis work we present a non-stochastic, discrete-time and discrete-variables strategy to address the qualitative analysis of metabolic network models. Our strategy is based on the use of the Model Checker Spin to both simulate and verify models of biochemical networks also at the genome-scale size. In particular, we designed a computational workflow that permits the translation of a metabolic network model into an executable model that can be analysed, either with an exhaustive or with an approximated strategy, using simple implicit-time properties.

The motivations underlying this choice are manifold. First of all, metabolic networks models are very large and typically contain thousands of reactions, thus making their analysis a challenging task. Moreover, the network parameters (such as reaction rates) are generally known only for a small subset of them. As a result, quantitative analyses are practically infeasible or reduced to very small models. Hence, only qualitative approaches may permit to obtain some informations on large or incomplete network models.

Moreover, model checking approaches have been mainly focused on signaling and gene networks, essentially because they are amenable to a binary representation that permits *i)* to fit very well the semantics of those checkers expressely designed for the verification and simulation of binary circuits, and *ii)* to reduce the computational burden both in terms of memory management and in terms of processing time, allowing also the elaboration of large models.

Metabolic networks have been studied in more depth by means of the Petri nets formalism, that is very well suited for their representation. However, this approach suffers of several problems, as very thoroughly described in Baldan et al. [11]. First of all, Petri Nets analyses are computationally expensive (the problem of deciding whether a state is part of the reachability set of a net is of intractable complexity, e.g. NP-hard and EXPSPACE-hard), permitting only the modeling of small networks. Moreover, this formalism has some limitations in the management of bidirectional reactions as well as some difficulties in representing the presence of metabolites belonging to different cellular compartments.

The methodology here proposed relies on the well-estabilished automaton-theory model checking approach (described in chapter 3) to tackle the problem of extracting relevant knowledge from a metabolic network model. This method has a computational complexity linear in the size of the model and exponential in the size of the LTL formula. However, we will demonstrate that several interesting biological properties may be extracted relying on reachability properties or on very simple LTL formula, hence preserving the linear complexity of the method. The space complexity is in principle exponential, although this is a worst-case upper bound that is rarely met. Moreover we choose as a tool the prominent model checker Spin [12]. Spin implements state-of-the-art optimization techniques that permit to efficiently explore the space of computations while limiting the issues related to the so called *state-space explosion* problem (i.e. the possible exponential size of the computation space).

Furthermore, we propose an innovative technique that exploits a Flux Balance Analysis step (see chapter 2) to decide in a biologically sound manner the directions of potentially bidirectional reactions. The results of this analysis may be also used to *prune* the model, i.e. to reduce its size by removing zero-flux reactions (see chapter 5). Another key property of the proposed methodology relies on the use of the *swarm* technique proposed by the Spin authors to deal with the verification of large problems: instead of relying on a single, exhaustive search, several searches with distinct strategies are run, with the aim

to explore as many distinct subsets of the state space as possible. Using this approach it is possible to find counterexamples also for usually intractable problems. We exploited this technique both for verification (i.e to search a single counterexample disproving a required property), and when using the model checker as a *behaviour filter*. The latter is based on a non-standard use of the model checker to extract *witnesses* of the ability of a metabolic network model to exhibit certain behaviours. This is obtained by negating the property of interest, as thoroughly explained in chapter 5.

The system model was built considering the reactions as processes that try to access concurrently to the shared variables, which represent concentrations of chemical compounds. A reaction may take place only if all the needed metabolites are available, and when many reactions may proceed at the same time, the possible order of their executions is resolved by non–determinism.

As a result, we are able to extract interesting information on the metabolite concentrations (expressed in generic units) and on the reactions that were used to fulfill the requested behaviour. Moreover, although using an implicit-time logic to define properties, we may easily reconstruct the dynamical information on the system evolution by considering time as a discrete entity inferred by the sequences of transitions along the extracted model executions.

We proved the effectiveness of this methodology by applying it on a genome-scale model of the human hepatocyte, with the aim to gain interesting knowledge on the mechanisms underlying a mendelian disorder called Primary Hyperoxaluria Type I.

1.4 Synopsis

This thesis is divided in two parts. The first one outlines the context in which this work is framed, whereas in the second part the proposed methodology is discussed in great detail, and its application to a genome-scale model of the human hepatocyte is described.

In chapter 2 the focus is on the Systems Biology paradigm. The central role of dynamical models of biological networks is considered, with a particular attention devoted to the subject of this thesis, i.e. the metabolic networks. Their essential features are described, together with their mathematical representation as a stoichiometric matrix and a brief description of one of the most used analysis tools, namely the Flux Balance Analysis method.

In chapter 3 the model checking approach is described, and the formal tools used for the specification of both the model and the property it should satisfy are introduced. More specifically, the Labeled Transition System (LTS) is defined, which is the low level structure used to describe a model. After, two high level languages, Promela and SMV, both having their semantics defined on a LTS, are outlined: the former permits to specify a model by means of an imperative syntax, whereas the latter is based on a descriptive definition of a finite-states automaton. Furthermore, two temporal logics, namely the Linear Temporal Logic (LTL) and the Computation Tree Logic (CTL), are described, together with their formal semantics. Then the problem of choosing a model checking tool is addressed, and a comparison between the two prominent model checkers Spin and NuSmv is performed. Afterwards, the choice of Spin is motivated in detail. Lastly, the automaton-based model checking approach of LTL properties implemented in Spin is outlined, together with some consideration on the complexity issues of this operation.

In chapter 4 a review of the existing formal verification approaches to the study of biological networks is presented, with the aim to correctly contextualize this thesis work. In particular, the classes of properties that may be determined with these kind of approaches are described, since they will be used as a reference in the following chapter.

Chapter 5 opens the second part of the thesis, and may be also considered the most important chapter since it contains several important elements of originality. In particular, the translation of the biological model into the correspondent a Promela program

is presented: it is based on the definition of concurrent processes, representing the reactions, that try to concurrently access to shared resources, representing the metabolite concentration values (in generic units). The benefits of a correct model abstraction are also detailed. Afterwards the set of interesting biological properties that it is possible to define, either as safety properties or as simple LTL formulae, is introduced. Furthermore, the non conventional use of the model checker as a *behaviour filter* is detailed: in this case the computation space is explored to extract *witnesses*, i.e. counterexamples showing the ability of the network to exhibit a certain behaviour. Furthermore, the innovative strategy of using a Flux Balance Analysis step to decide reaction directions is explained, together with the possible exploitation of the obtained data to *prune*, i.e. to also reduce the model in a biologically sound manner. The problem of an approximated search strategy to deal with very large model is then presented, together with a discussion on the strength and limits of such an approach. Lastly, a metalevel of control is introduced, to both help in reducing the portion of the explored space and to permit a more fined-grain modeling of the biological system.

Chapter 6 is focused on the technical details of the software workflow realized to implement the proposed methodology. In particular, input and output format of the various modules are described, with a special focus on how the metabolite concentration and reaction usage information are extracted from the counterexamples files generated by the model checker Spin. Furthermore, the file containing the dynamical information of the metabolites are described: here the time, although not explicitly defined, is considered as a discrete entity inferred by the sequences of transitions along the executions. Lastly, a discussion on the parameter settings of the *swarm* tool, proposed by the Spin authors as an approximated solution to deal with large models, is presented. More specifically, the influence of some parameters on the quality of the behaviour filtering results is addressed.

In Chapter 7 the proposed methodology is tested on a genome scale model of the human hepatocyte, with the aim to verify its correctness and extract interesting information on the mechanisms underlying a mendelian disorder called Primary Hyperoxaluria Type I (PH1). This case study demonstrates the effectiveness of the the implemented workflow on a very large model: proposed methodology allowed both the refinement of the initial model, both the extraction of a large number of interesting qualitative information that were consistent with the known biological evidences of the disease.

Chapter 8 closes the thesis highlighting the obtained results and the future directions of the proposed methodology.

Part I

Background

2

Biological Network Models in the Light of Systems Biology

Systems Biology is a new paradigm that emerged in Biology about fifteen years ago, propelled by the the growing amount of data that high–throughput sequencing technologies were revealing. The need for a more complete and *whole–istic* [2] approach to the interpretation of this huge and complex amount of data drove a shift of perspective: from a traditional, reductionist approach, focused on the single biological components (such as nucleotides, genes, proteins) the attention was moved to an holistic, dynamic view of interacting networks of biological entities. In this new light, genome–scale Biological Network Models play a crucial role: they are the basis for predictive, hypothesis–driven science focused on understanding the structure and the dynamics of a system at the whole genome level.

In the following we will give a brief overview of the Systems Biology approach, and after we will describe different type of Biological Networks. In particular we will focus on Metabolic Networks, since they are the subject of this thesis work.

2.1 The Systems Biology Paradigm

The availability of large amounts of data coming from genome–scale high–throughput sequencing technologies was the driving force behind the rise of the Systems Biology paradigm [1]. In contrast to molecular biology, that is focused on studying the single components of biological entities, Systems Biology aims at understanding life at the whole system level. The large collections of detailed omics data produced by molecular studies are essential to create complex models [3, 13]. This data has a hierarchical nature as shown in figure 2.1: starting from basic elements (such as DNA, mRNA, proteins, and metabolites) several levels of growing complexity may be considered, that range from protein interactions, signaling pathways, and reaction networks to large–scale systems such as cells, tissues or whole organisms [4, 5].

However, the simple organization of data into networks is not enough to gain insight on the features of the system as a whole. As clearly outlined in the seminal paper of Kitano [6], a system–level understanding of an organism should address *i)* the *system structure*, i.e. its components and the way in which they are related, and *ii) the system dynamics*, i.e. how it behaves over time under various conditions. The reconstruction of the **behaviour in time and in space** of the whole system is central in Systems Biology, and is an essential step to unravel the relationships intercurring between the genotype and the phenotype of organisms [7].

In particular, structure and dynamics together may provide a baseline for investigating on an essential, system–level property of biological systems, namely **robustness** [14]. Robustness involves the ability of the system to *i)* constantly adapt to internal or external changes, *ii)* be tolerant to the noise generated by the stochastic signals to which they are exposed, *iii)* exhibit a graceful degradation, i.e. is a slow and gradual degradation of a system functions after damage, rather than catastrophic failure [6]. As Kitano

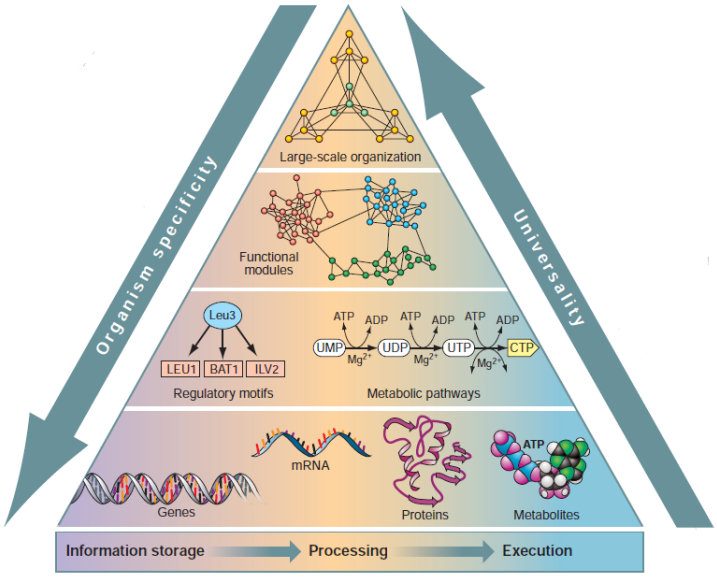

Fig. 2.1. Insights into the logic of cellular organization can be obtained when the cell is viewed as a complex network in which the components are connected by functional links. At the bottom there are the basic elements of life, e.g. genes, proteins and metabolites, that are integrated in a hierarchy of growing complexity. At the lowest level, these components form genetic–regulatory motifs or metabolic pathways (level 2), which in turn are the building blocks of functional modules (level 3). These modules are nested, generating a large–scale architecture (level 4). Taken from [5].

said in [14], *robustness is the fundamental feature that enables diverse species to generate and evolve*. Indeed, this is due to many reasons: *i)* robustness is ubiquitous, in the sense that is observable in a wide range of fundamental biological processes; *ii)* robustness is tightly connected with the ability to evolve facing the environmental and genetic perturbation; *iii)* diseases may be due to a sort of trade–off between robustness and the inevitable fragility of evolvable robust systems. Hence, a deep understanding of robustness mechanisms may give fundamental insights on diseases such as cancer, diabetes, and immunological disorders.

To this aim, the biological information should be necessarily complemented by mathematical and/or computational models, either manually or automatically built [15]: these models may be executed or simulated to reveal the adequacy of the biological assumptions and hypotheses on which they were based, in case leading to a refinement of the model and to new biological (*wet*) experiments. This *hypothesis–driven* research approach, and the cyclic process of information integration and *in silico* model building, is typical of the Systems Biology paradigm [4, 6].

The structures that are most used to represent biological systems are *networks* of elements that interact and evolve over time, thus being responsible of the emerging behaviour of the whole system. Depending on the features of the involved elements and the exchanged information, different biological network may be considered, each offering a distinct view of the biological system under study. We will briefly describe some of them in the next section.

2.2 *In silico* models: Genome–Scale Biological Networks

Several type of genome–scale biological networks may be compiled starting from omics data. A schematic view of the most relevant ones is shown in figure 2.2, where they are classified based on the type of elements and interactions they involve.

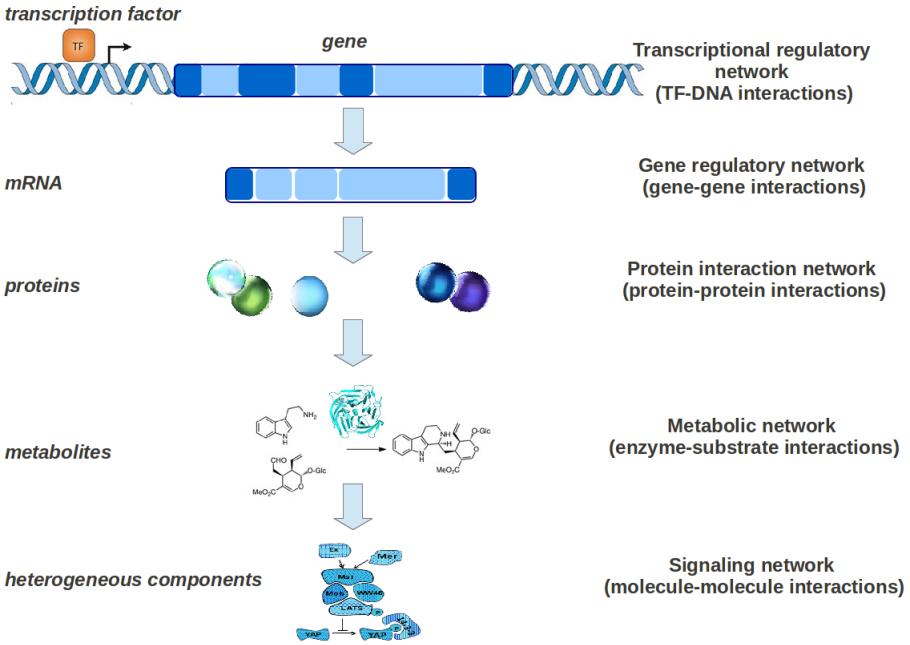

Fig. 2.2. Several hierarchical layers of molecular interactions may be identified in a cell. They involve different kind of relationships between the fundamental elements of life (DNA, RNA, proteins, metabolites, heterogeneous components). Each level may be represented as a biomolecular network, characterized by the kind of interaction that intercurs between pairs of elements.

As already said and shown in figure 2.1, the cell organization may be viewed as a hierarchical combination of layers. The lowest level involve the **molecular interactions** among the fundamental elements of life, e.g. DNA, RNA, protein and metabolite networks [5,16]. It is worth noting that also these interactions are hierarchically organized as shown in figure 2.2, and each level may be represented as a biomolecular network, characterized by the type of interaction that intercurs between pairs of elements. Thus, biomolecular networks can be modeled as graphs, either directed or indirected, where molecules are represented by nodes, and their interactions are represented by edges.

However, it must be reminded that such choice sometimes trades details for simplicity: many different mechanisms of transcription regulation, for example, may be hidden by a single interaction (edge); similarly, enzyme strength or speed are not taken into account when connecting metabolites. Nonetheless, this representation offers indubitable abstraction advantages, since it permits to capture many of the essential characteristics of the underlying biomolecular processes. Seeing a biological system as a graph permits to infer interesting properties (such as robustness [17], or modularity [18]) that have been already established for other systems having the same representation (such engineering ones) [19,20]. At the same time, computational methods expressly developed for graph analyses may be used to extract topological and dynamical features of the modeled systems [18,21–24].

Here, we will give only a brief description and some references related to the network types shown in figure 2.2. In the following section we will describe in great detail the metabolic (reaction) networks.

Transcriptional regulatory networks describe the interactions between proteins (transcription factors) and the DNA that controls gene expression in the cells. These are very complex networks [25], hence a large part of studies focused primarily on their topological features. To better understand them, the basic building blocks (called *motifs* have been extracted and investigated [26].

Gene regulatory networks describe the interactions between cis–regulatory DNAs (e.g., enhancers and silencers) that control gene expression and the targeted genes [27]. Due to the tight connection between the structure of the network and the expressed functions [28] they are widely studied, both in what concerns the network reconstruction phase (see the ARACNE algorithm [29]) both in their structure [30,31], dynamics [32–34], stability [32, 35] features.

Protein interaction networks describe intentional physical contacts established between two (or more) proteins as a result of biochemical events and/or electrostatic forces. They were mainly compiled for small organisms such as yeast (*S. cerevisiae*) [36], or for small portions of higher organisms, and only recently a genome–wide human network has been completed [37, 38]. Several methods to study their dynamics have been proposed, see [39] for a review on them.

Metabolic networks describe the metabolism of cells and living organisms in terms of the reactions that both transport biochemical species across the boundary of the cell, and transform them in the fundamental process of sustaining life [7]. Their reconstruction is really challenging as described in [7, 40, 41]. The most popular approaches to study metabolic networks are the so called Constraint Based Methods [42].

Signaling networks describe the complex interactions that are responsible for the exchange of information within and across the cell boundaries. The signal transduction networks regulate diverse and fundamental aspects of the cellular life [43], with a highly dynamic and extraordinarily complex behaviour [44]. The reconstruction of these network is a very tough task [45], as well as the study of their dynamics over time, that has been addressed in several mathematical and/or computational ways [46–49]

It is worth noting that also Model Checking approaches have been used to address the problem of studying the dynamics of Gene Regulatory, Signaling and Metabolic Networks, as will be described in greater detail in chapter 4.

2.3 Metabolic Networks

Metabolism is the set of molecular transformations and energy transfers which constantly take place in the cells of living organisms to sustain life. It includes degradation processes (catabolism) as well as organic synthesis (anabolism) as shown in figure 2.3. This complex and interwoven system of interacting biochemical transformations, carefully organized in space and time, is described (at the finest level of detail, see Palsson [7]) by networks of biochemical reactions, thus called *Metabolic Networks*.

2.3.1 Network Reconstruction

The process of reconstructing a genome–scale metabolic network, i.e. the identification of the whole set of reactions that are involved in the metabolism of a living organism, is really challenging [7, 40, 41].

Thanks to the high–throughput technologies, complete genomic sequences of organisms are available, together with the corresponding functional annotations. They permit to infer the set of the metabolic reactions which are likely to occur in the organism, i.e. reactions catalysed by the enzymes for which coding genes have been identified. However, this first collection must be necessarily integrated with physiological knowledge and a specific curation effort. Moreover, as new genes are annotated as playing a role in metabolism, or as existing enzymatic annotations are reviewed, the model can be refined.

When the model is complete enough to allow testable predictions about the metabolic behaviour of the whole cell, it can be used to evaluate the goodness of the reconstruction obtained from the functional annotations. Typically, this evaluation is achieved by comparing the model predictions on specific metabolic targets with the known physiological

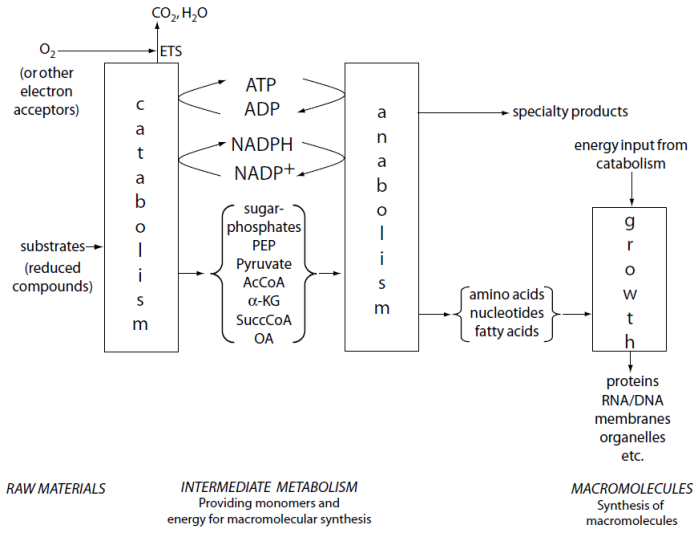

Fig. 2.3. A schematic view of cellular metabolism. It can be roughly divided into a catabolic activity, in which raw materials and substrates are reduced to fundamental species, and an anabolic activity, where the basic elements are used to synthesize more complex molecules such as nucleotides, aminoacids and fatty acids. These, together with the energy obtained in the catalitic activity, are used for the cell growth, which includes the synthesis of different kind of macromolecules. Taken from [7].

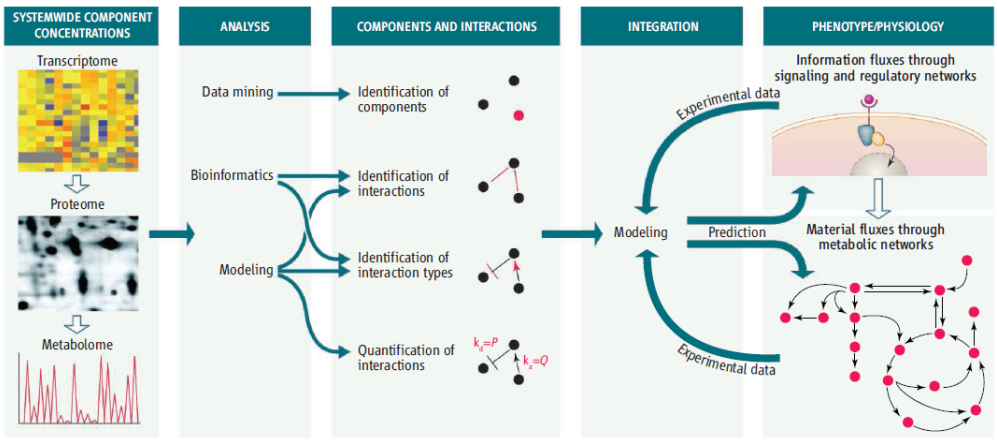

Fig. 2.4. The Systems Biology approach to metabolism. Taken from [41].

information, or with data obtained from new biological experiments. If incoherences are found between the model predictions and the expected experimental results, then the model may be incomplete or erroneous and should be modified. Using the clues suggested by the model predictions, necessarily complemented by a certain degree of functional knowledge, it may be possible to refine the functional annotations and, in case, design new experiments targeted to clarify the unknown or partially explored portions of metabolism.

This Systems Biology approach to the reconstruction of a metabolic network is shown in figure 2.4. Starting from omics data, a system–wide process of component identification and quantification is performed, that involves mRNA, proteins, and small molecular weight metabolites. A subsequent analysis step should lead to the identifications of components and interactions, together with the information on their structure and type. When possible, quantitative measurements of interactions should be extracted. All this

heterogeneous data should undergo a rigorous step of integration, and then, aided by computational or mathematical modeling, the cycle of investigation and refinement that is typical of Systems Biology may take place.

The detailed account of the genome–wide metabolic network reconstruction process is out of the scope of this thesis, but a detailed analysis may be found in Feist et al. [40], where the described methodology is applied to the reconstruction of bacterial metabolic networks. This paper also offers a web resource [50] (maintained by the Systems Biology Research Group at the University of California, San Diego) that trace the current state of development of a large collection of genome–scale metabolic network models for bacteria, archea and eukaryotes. To have a visual perception of the overwhelming complexity of metabolic networks a navigable online map, based on the Roche atlas of metabolic pathways, is maintained online at the EXPASY site [51], and is widely described in the book of Michal [52]. The Kyōto Encyclopedia of Genes and Genomes (KEGG) [53, 54] is the most comprehensive resource for omics studies, available online here [55]. It includes also metabolic pathways of several organisms, that are explorable through a visual map and linked with a wide range of supplementary information on the involved metabolite species, reactions, and enzymes.

2.3.2 Network Analysis

As can be deduced from the above sketched description of the network reconstruction process, the analysis of genome–scale metabolic models is a very complicated and demanding task. First of all, even the simplest organism model contains hundreds of reactions and metabolites (see the the already cited resource at [50] for several examples), and this number grows with the organism complexity, reaching more than three thousand in the *homo sapiens* model. Second, the quantitative parameters, such as reaction fluxes and component concentrations, are not yet available at the whole genome–scale for many of the reconstructed models.

For these reasons, the first works on reconstructed genome–scale metabolic models were mainly exploiting the topological features of networks [56, 57]. With the advances in omics technologies, different approaches have been devised to highlight the temporal dynamics of metabolic networks, each of them having advantages and disadvantages as described in Palsson et al. [58]. Among them, quantitative stochastic or deterministic kinetic approaches have been proposed [59, 60]. However, these models are usually computationally intensive and strongly affected by the lack of precise parameters, and can be better exploited to study small–scale biological processes.

On the contrary, the Constraint-Based modeling framework first proposed by Price et al. [61] has obtained a great success. The already cited work of Palsson et al. [58] provides an online list [62] of more than six hundred papers (dating from 1985 to 2014) that exploited this methodology to investigate on the expressed phenotype of metabolic network models of several organisms. The Constraint–Based Method (CBM) class encompasses a high number of qualitative approaches that may be used for modeling whole-genome scale networks for which quantitative data are not available. Many of them are implemented in widely used software environments such as the COBRA [63], the OptFlux [64], and FASIMU [65] toolboxes. The papers of Dandekar et al. [66] and Lakshmanan et al. [67] offer an exhaustive review of the features and limitations of a wide range of dedicated tools.

In a nutshell, CBMs interpret a metabolic network as a *flow* network. A mathematical representation of the network (the *stoichiometric matrix*) is used to compute a solution space, that is limited by three primary constraints: reaction substrate and enzyme availability, mass and charge conservation, and thermodynamics [42]. Other bounds, derived by a specific knowledge of the system at hand, may be used to reduce the solution space. The obtained solutions define the metabolite fluxes that traverse the reactions while satisfying the given physico–chemical constraints.

A complete description of all the existing CBMs is out of the scope of this thesis, however a very thoroughly review can be found in Lewis et al. [42]. We will focus here only on the Flux Balance Analysis approach, since it is used as a preprocessing step in this thesis work (see subsection 5.3.5).

Although abstract and simplified, the representations obtained with CBM may provide relevant insights on cellular metabolism. In fact, they offer: *i)* a descriptive view of the structured ensemble of reactions that take place in the cell, and *ii)* a predictive power that allows to describe to some extent qualitative and quantitative dynamic features of the cellular metabolic behaviour.

2.3.3 Network Representation

The traditional representation of a metabolic network used by CBMs takes into account the reactions, their direction and the involved metabolites, whereas the enzyme catalysing activity is not explicitly modeled [68]. A biochemical reaction involves amount

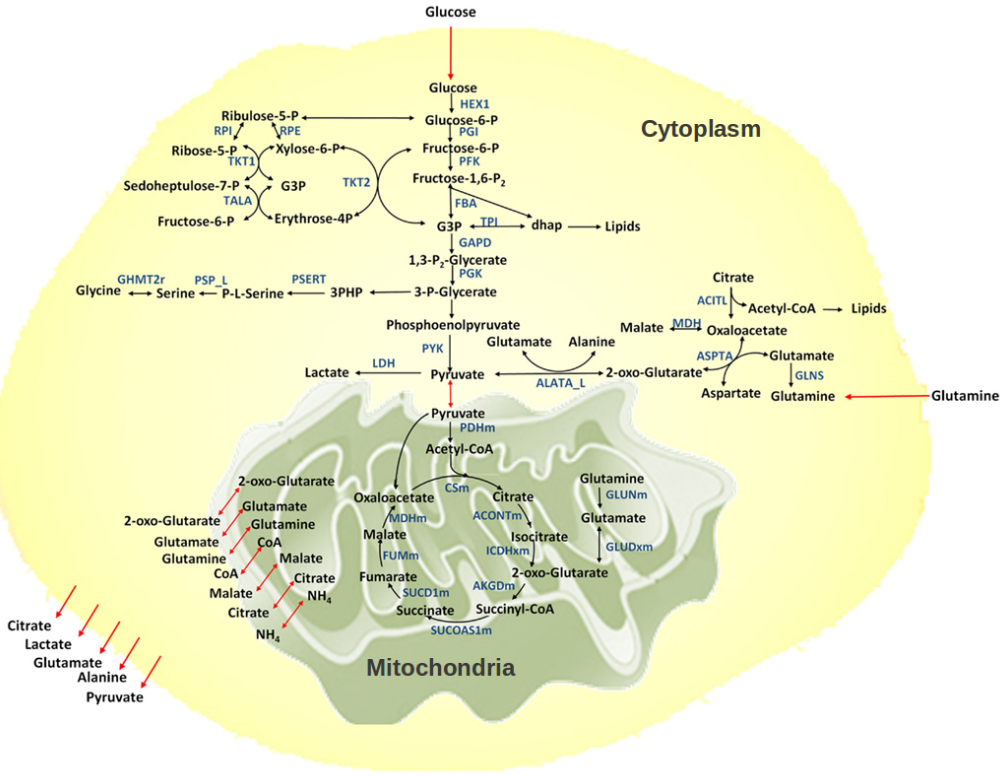

Fig. 2.5. An example of metabolic network model representing a small portion of the human metabolism. Reactions belonging to the cytoplasm and mitochondria compartments are shown. Their preferential directionality is indicated by the arrow, whose color indicates the reaction type: black for the internal, and red for the transport and exchange reactions. The stoichiometry of reactions is also shown, together with the name, in blue, of the catalysing enzyme. Adapted and modified from [69].

of metabolites that are indicated by the corresponding integer stoichiometry. The elements on the left of a reaction are called *reactants* whereas the one on the right part are called *products*. The reactions in a GEnome–scale Model (GEM) may be organized in

groups, corresponding to the cellular compartments (if any) of the modeled biological system. The reactions that move metabolites across compartments or exchange them with the outside environment are indicated as *transport* (or *external*) reactions to distinguish them from the other ones (in turn called *internal*).

The preferential reaction direction is normally indicated by an arrow. However the effective reaction direction is tightly connected to the associated Gibbs free energy under the standard conditions ΔG^0. When ΔG^0 is large and negative, the reaction tends to go in the forward direction; when ΔG^0 is large and positive, the reaction tends to go in the reverse direction; and when $\Delta G^0 \simeq 0$, the system is at equilibrium, and the reactions may proceed in any direction (bi–directional reactions) depending on other parameters such as the reactants/products concentration [70]. When the ΔG^0 value is not known, the corresponding reaction is indicated as being bi–directional, and the effective direction will be established during the analysis.

However, the CBMs are based on a mathematical abstraction of this representation, called the *stoichiometric matrix*. It is defined as the matrix $\mathbf{S} \in \mathbb{R}^{n \times m}$, where n and m indicate respectively the number of metabolites and reactions. An element s_{ij} *in* S is non–zero if the metabolite i is present in the reaction j. Its value is given by the stoichiometric coefficient that the element i has in the reaction j, with a negative sign if it acts as a reactant, and a positive one if it is a final product.

Lastly, a metabolic network may be easily represented and exchanged using the Systems Biology Markup Language (SBML) [71], an XML–based description language that is widely used to publicly distribute reconstructed GEMs.

2.3.4 Flux Balance Analysis

Flux Balance Analysis (FBA) is the most basic and commonly used method for simulating genome–scale metabolism [58]. In FBA, a cellular objective is defined, and a qualitative flux distribution is computed by solving a linear programming problem without knowledge of detailed kinetics of metabolic reactions and of compound concentrations. The solution is found optimizing an objective function that represents the principal biological target (such as the biomass production), subject to the constraints imposed by the metabolic network and metabolite uptake rates [42]. This calculation finds a point in the solution space that should best represent the true cellular phenotype. The solution includes a prediction of the optimal objective magnitude (e.g., biomass yield or growth rate) and potential flux values for each reaction.

We will give same detail only of the FBA approach used in [72], since it was the one used as a preprocessing step in this thesis work (see chapter 5). Here the optimization problem starts from the hypothesis that the cells perform their functions using the minimal possible amount of energy. Furthermore, the flux are weighted using the information on the Gibbs free energy under the standard conditions: this permits to decide the flux directions using biologically sound constraint.

The considered stoichiometric matrix $\mathbf{S} \in \mathbb{R}^{n \times 2m}$ has $2m$ reactions because a metabolic flux across a reaction can be, in principle, positive or negative. Therefore, to ensure the non–negativeness of the variables, the flux of each reaction r_j is decomposed into an irreversible forward one, $v_j^{(+)}$, and an irreversible backward one, $v_j^{(-)}$, as $v_j = v_j^{(+)} - v_j^{(-)}$. Since, by definition, only one of the two components can be different from zero, we have that $v_j^{(+)} = v_j \cdot \chi(v_j)$, and $v_j^{(-)} = v_j \cdot [\chi(v_j) - 1]$, where $\chi(\cdot)$ represent the unit–step function. The flux balance constraint is given by $\mathbf{S} \cdot V = 0$ where $V = (v_1^{(+)}, v_2^{(+)}, \ldots, v_m^{(+)}, v_1^{(-)}, v_2^{(-)}, \ldots, v_m^{(-)}) \in \mathbb{R}^{2m}$ is the vector of fluxes associated with the reactions of the network. Additional constraints, including those related to the maximal fluxes that can be supported by each reaction, are introduced as inequalities. Furthermore, we need to specify the metabolic input(s) and output(s) of the network, i.e. the boundary reactions. By considering k boundary reactions, we obtain an extended

stoichiometric matrix $\tilde{\mathbf{S}} \in \mathbb{R}^{n\times(2m+k)}$ and an extended vector of fluxes $\tilde{V} \in \mathbb{R}^{2m+k}$. To accomplish a particular functional state, the fluxes through a certain number of essential reactions, called target reactions r_j^{tar}, have to be maintained at non–zero values, that is, $v_j^{tar} = \kappa_j > 0$.

In order to optimize biological objective functions, we applied the principle of flux-minimization, which states that, given the value of relevant target fluxes, the most likely distribution of stationary fluxes within the metabolic network is such that the weighted sum of all fluxes is a minimum [73]. Hence, it results in the solution of the following constrained optimization problem for the calculation of steady–state metabolic fluxes:

$$\min_{V\in\mathbb{R}^{2m}} \left(\sum_{j=1}^{m} (w_j \cdot v_j^{(+)} + w_j \cdot K_j^{equ} \cdot v_j^{(-)}) \right) \tag{2.1}$$

where w_j is the weight associated with v_j, while the equilibrium constants K_j^{equ} are introduced to constraint fluxes according to Gibbs free energy calculations. We expressed the equilibrium constant for a reaction r_j through the change of Gibbs free energy under the standard conditions, denoted as $\Delta G_{r_j}^0$:

$$K_j^{equ} = e^{-\frac{\Delta G_{r_j}^0}{R\cdot T}} \tag{2.2}$$

where R is the universal gas constant and T is the absolute temperature. Weighting the backward flux with the thermodynamic equilibrium constants takes into account the thermodynamic effort connected with reversing the natural direction of the reaction. The minimization problem (2.1) is subject to the following constraints:

$$\begin{aligned} \tilde{\mathbb{S}} \cdot \tilde{V} &= 0 \\ 0 \le v_j^{(+)} &\le u_j^{(+)} \\ 0 \le v_j^{(-)} &\le u_j^{(-)} \\ v_j^{tar} &= k_j \end{aligned} \tag{2.3}$$

where $u_j^{(+)}, u_j^{(-)} \in \mathbb{R}^+$ represent the upper bounds of $v_j^{(+)}$ and $v_j^{(-)}$, respectively.

The target fluxes to specify in the minimization problem of equation 2.3 depend from the features of the used model. Since a metabolic network usually encompasses a wide range of very complex functions, its associated network model should be able to satisfy several minimization problems, each one that accomplishes a different metabolic objective (i.e. a different function).

In this case, being l the number of different metabolic objectives, an optimization problem is solved for each of them. This results in a $m \times l$ matrix $\mathbb{V} \in \mathbb{R}^{m\times l}$ that contains the calculated fluxes of the m internal reactions for the l distinct targets. An element $v_{i,j} \in \mathbb{V}$ represents the flux of reaction r_i in the j–th metabolic objective.

It is possible to extract the average flux distribution across all the metabolic objectives as:

$$\bar{v}_i = \frac{1}{l} \sum_{j=1}^{l} v_{i,j} \tag{2.4}$$

.

The average flux is a representation of the average behaviour of the network across all the possible metabolic objectives, and is a fundamental information that will be exploited in this thesis work (to be more precise in section 5.3.5).

2.4 Chapter Summary

In this chapter the Systems Biology paradigm was introduced. In contrast to the traditional reductionist approach of molecular biology, Systems Biology is aimed at interpreting in a dynamic and holistic way the growing amount of data that high–throughput

sequencing technologies are revealing. To get a whole–system picture of the behaviour of an organism, two aspects must be taken into account: *i)* the *system structure*, i.e. its components and the way in which they are related, and *ii) the system dynamics*, i.e. how it behaves over time under various conditions. Hence, the most natural representation of the system structure is a network of interacting elements. However, the bare topological information do not suffice to describe the behaviour of the underlying biological system: instead a dynamical understanding is needed. To this aim, the biological information should be necessarily complemented by mathematical and/or computational models that may be executed or simulated to reveal the adequacy of the biological assumptions and hypotheses on which they were based, in case leading to a refinement of the model and to new biological (*wet*) experiments. To make this possible, mathematical representation of the biological systems are needed. In particular we described the Mathematica representation of metabolic network models, namely the stoichiometric matrix, together with one of the most used methods to study them, namely the Flux Balance Analysis, since they are both essential to this thesis work.

3

Model Checking

It is well known that building flawless computer systems (either hardware or software) is a very tough task, and it is a matter of fact that finding errors and increasing the reliability of those system is, if possible, harder [74]. Although much research is aimed to improve the existing methods [75, 76], the testing approach does not guarantee the complete coverage of all the possible system executions. Moreover, errors in concurrent systems are more difficult to find, since they are often hard to reproduce.

Starting from the intuition that computer systems may be thought as mathematical objects, a new field of study, often referred to as *Formal Methods*, has emerged about forty years ago. The main idea behind this approach was that, for the mathematical representations of computer systems, a specification of the wanted (correct) behaviour could be expressed in terms of some logic. Then, a formal proof could establish if the system was fulfilling the given specification. The first efforts in this sense were proof-theoretic approaches (such as the Floyd-Hoare Logic [77,78]) based on constructing proofs by hand. However, the infeasibility of hand writing proof for complex systems made this technique rapidly outdated.

Moreover, concurrent programs (which are often described as *reactive systems*, [79]), typically show nonterminating behaviour, and are non-deterministic so that their non-repeatable behaviour was not amenable to testing in the Floyd-Hoare Logic.

In the early eighties, Emerson and Clarke [80] and Queille and Sifakis [81], independently defined the foundation of Model Checking (actually Emerson and Clarke also coined the term *Model Checking*). They were inspired by a seminal paper of Pnueli [82], where he suggested the use of Linear Temporal Logic for reasoning about reactive systems (whose semantics can be given as infinite sequences of computation states (*paths*) or as computation *trees* [83]).

Temporal Logics were developed by philosophers to reason about the ordering of events in time, but without the explicit definition of time itself [84]. These logics are often classified based on the assumption made about the underlying time structure, that could be defined either as linear or branching. The meaning of a temporal logic formula is established with respect to a labeled state-transition graph, i.e a *Kripke structure* [85]. Bot linear and branching time logics are characterized by a high degree of expressiveness, that permits to capture a wide range of correctness properties of concurrent programs, and a great deal of flexibility [82].

The brilliant intuition of Emerson and Clarke was to combine the state-exploration approach with Temporal Logic in an efficient manner, showing that the result could be used to solve non-trivial problems. As they state in their first paper [80]: “We argue that proof construction is unnecessary in the case of finite state concurrent systems and can be replaced by a model-theoretic approach which will mechanically determine if the system meets a specification expressed in propositional temporal logic. The global state graph of the concurrent systems can be viewed as a finite Kripke structure and

an efficient algorithm can be given to determine whether a structure is a model of a particular formula (i.e. to determine if the program meets its specification)".

Following these ideas, the Model Checking Problem is stated by Clarke [86] as follows:

Definition 1 (The Model Checking Problem) *Let M be a Kripke structure (i.e. a state-transition graph). Let f be a formula of temporal logic (i.e. the specification). Find all states s of M such that $M, s \models f$.*

This consists in determining if the temporal formula f is true in the Kripke structure M i.e. in **checking** whether the structure M is a **model** for the formula f.

Emerson and Clarke presented a polynomial algorithm for solving the Model Checking Problem for a new (Branching Time) Temporal Logic called Computation Tree Logic (CTL); an improved, linear version of the algorithm was presented in [87], and implemented in the EMC model checker, the first implementing *Fairness Constraints* [86, 88]. The Explicit Model Checker (EMC) evolved rapidly in the MCB model checker, that was able to return a counterexample if the property was not satisfied, introducing a feature that is now a standard for all model checking system.

In the same years, different approaches to the Model Checking of concurrent systems were introduced. The most prominent is due to Vardi and Wolper [89], that used Büchi automata to represent and validate Linear Time Logic (LTL) formulae. The resulting methodology, often referred to as *Automata-based* Model Checking is implemented, for instance, in the popular Model Checker Spin [12].

A detailed account of the birth, the past and the future perspectives of this prominent methodology are given by the main protagonists in a book containing the papers written for the 25th anniversary of Model Checking, [90].

The general structure of a Model Checking system is shown in figure 3.1: the requirements that the system should fulfill are formalized in a suitable property specification language (for instance a Temporal Logic). The system model is fed to the Model Checker together with the property specification. The Model Checker will exhaustively explore all the possible computation of the system model, verifying if the property holds for each of them. If it is not the case, a counterexample is produced that should help in the debugging procedure.

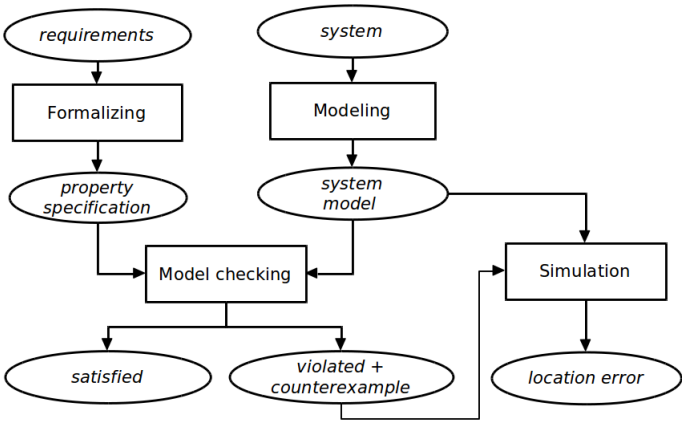

Fig. 3.1. A schematic view of the general structure of a Model Checking system (adapted from [91]).

This methodology has been successfully applied to a large variety of systems, hardware [92, 93], software (ranging from protocol communication [94] to spacecraft controllers [95]), and even biological ones (see chapter 4).

In the following we will give some details on the techniques needed for the Model Checking of linear and branching time properties on parallel and communicating (concurrent) systems. Then, we will motivate our choice of Spin as a model checker and we will briefly describe the techniques on which its based. After we, will address some complexity issues such as the well known *state-space explosion* problem.

3.1 Techniques

The input to the Model Checker (see figure 3.1) are the specifications of the system under study and the property that the system should fulfill. We will give some detail on the low level mathematical structures used for describing system models, and we will provide the description of the high level languages ot two prominent model checking tools, namely Spin and NuSMV. After we will introduce the LTL and CTL logics, describing both their syntax and semantics.

3.1.1 Model specification

The model of the system under study is a prerequisite for Model Checking, and should be specified with a language able to describe its fundamental features, such as:

- the structure, i.e. the component of the system and the way in which they are interconnected. In particular, it must be possible to define parallel and communicating components;
- the state, i.e. a complete *snapshot* of the system. A snapshot is the set of information that describe the system at each moment of its behaviour;
- the behaviours, i.e. the sequence of states that the system may go through, while changing.

Besides having a well defined *syntax*, such a language should possess a *formal semantics* permitting the association of the syntactic description to a mathematical structure, on which the model checking process will be *de facto* performed.

Although several specification formalisms have been defined, like Petri Nets [96] or Process Algebras (such as the π-calculus [97], the Communicating Sequential Process (CSP) [98], or the Performance Evaluation Process Algebra (PEPA) [99]) we will focus only on the State-Transition formalism, since it represents a standard in Model Checking [91] due to its expressive power and its intuitive graphical representation. The State-Transition formalism class contains, for instance, the Finite State Machine (FSM) [100], the Mealy [101] and the Moore [102] Machines, the Büchi [103] and the Timed Automata [104].

Since several automata formalisms are available, the right choice depends from the nature of the problem that should be modeled. The first fundamental issue is related to the importance of time. Indeed, if the system under study is a *time-critical* system with stringent timing constraints, a suitable formalisms is needed that includes an explicit representation of time. To model these systems, a Timed Automata formalism has been successfully implemented in UPPAAL [105] and in Kronos [106]. Both these Model Checking tools provide a high level language to describe the system and avoid the direct declaration of the timed automata.

If time does not need to be explicitly represented, different choices are available that are mainly based on the properties that should be verified (either in linear or in branching time logic) and on the availability of a high level specification language suitable to capture the features of the system under study. Among the existing tools, we will consider only the Spin model checker [12], that implements the Automata-based Model Checking approach for Linear Temporal Logic (LTL) properties, and the NuSMV model

checker [107], that performs Symbolic Model Checking of properties expressed in both Linear and Computation Tree Logic (CTL).

This choice is due to the following considerations: *i)* we are not interested in the time-critical features of the (biological) system under study in this thesis work. *ii)* both Spin and NuSMV have a high level model specification language, called *Promela* and *SMV*, respectively.

It is worth noting that both these languages, despite being remarkably different as will be clarified in section 3.1.1, have their formal semantics defined on the same mathematical structure, namely a Labeled Transition System (LTS), that will be introduced hereafter.

Labeled Transition Systems

A Labeled Transition System (LTS) can be formally defined as follows.

Definition 1 *Labeled Transition System*
Let Σ be an alphabet. A LTS is a triple $\langle S, S_0, \Delta \rangle$ where

- *S = is a set of states*
- *$S_0 \subseteq S$ = is the set of the initial states*
- *$\Delta \subseteq S \times \Sigma \times S$ = is the transition relation between states.*

In this formalism a **state** is a snapshot of the system (i.e. the state of both the control and the data structures), and a **transition** $\langle s, \alpha, s' \rangle \in \Delta$ means that the system may evolve from state s to state s' with an observable action that is labeled as a. A **behaviour** of the system is a sequence of transitions (either finite or infinite) that starts from an initial state and follows a path in the LTS:

Definition 2 *Behaviour of a LTS*
A behaviour is a sequence of states $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \ldots \xrightarrow{a_{i-1}} s_i$ with

- *$s_0 \in S_0$*
- *$\langle s_{j-1}, a_{j-1}, s_j \rangle \in \Delta$, $\forall\ 1 \leq j \leq i$.*

A **trace** is the observable part of a behaviour, and is given by the sequence of the labels i.e. $trace(s_0 \ldots s_i) = a_0, a_1, \ldots a_{i-1}$. A **path** π of the transition system is defined as a behaviour that either ends in a terminal state or is infinite. The set of all the paths starting in a state s are indicated with $Paths(s)$. $Traces(s)$ is the set of the traces of all the paths starting in s, i.e. $Traces(s) = trace(Paths(s))$. Lastly, $Traces(LTS) = \bigcup_{s \in S_0} Traces(s)$ is the set of all traces starting from an initial state of the LTS.

It is worth noting that, although it shares the same graphic structure of a State-Transition systems, the LTS is a lower level formalism with respect to the correspondent specification language. Since the LTS is the same mathematical structure in which different languages are translated (via the formal semantics), the definition of an equivalence criterion may permit the comparison between different syntactic specification. This is of great interest in the process of translating the system into its formal specification: the initial model may be very close to the original system, and several subsequent step of *abstraction* will eventually transform the initial specification into a reduced model that captures only the details relevant to the checking process. Hence, the possibility of controlling the correctness of the abstraction process through the comparison of the obtained LTS is of capital relevance. To this aim, several equivalence relationships have been defined, such as the strong and the weak bisimulation [91].

High Level Specification Languages

Although the LTS is the common mathematical structure used for model specification, usually model checking tools provide high level languages that offer a great deal of flexibility and easiness of specification when compared to the manual specification of a LTS model. We will briefly describe the Spin language, Promela, and the NuSMV one, called SMV. When choosing a tool, besides considering the type of logic it supports, also the available modeling language should be carefully evaluated.

Promela

Promela is an imperative language, especially tailored for the purpose of writing high-level specification models of concurrent systems [12]. It allows for the declaration of three type of objects:

- *Processes*. They describe the structural components of the system. A maximum number of 255 process may be declared;
- *Data Objects*. These are the variables, either local to processes or global. A global declaration allows for shared variable communication. The variable size must be finite;
- *Message Channels*. They model the exchange of data between processes, allowing both for handshaking (i.e. synchronized communication) or for asynchronous send/receive operations. A set of side-effect free operations permits to check if the queue is empty or full, to verify its length or the presence of a value on the top.

The semantics of the Promela instructions is based on the concept of executability: each statement is either executable or blocked depending on the system state. The selection among multiple executable statements is non-deterministic: instructions, either belonging to the same or to different processes, may be arbitrarily interleaved. However, it is possible to explicitly define blocks of uninterruptible sequences of instructions using the keyword *atomic*. The Promela language provides also flow control instructions, such as the *if-else* selection construct and the *do* repetition statement. However, their semantics is different from similar statements of other imperative languages: indeed, if more then one instruction is executable, then the choice is purely non deterministic.

SMV

SMV is a descriptive language, originally created by McMillan [108] and extended in the NuSMV tool [109] to include invariants and LTL formulae specifications. It allows to declare:

- *Modules*. Each module is a Finite State Machine. A complex system can be divided into modules, each instantiable many times. Modules can be composed either synchronously or asynchronously (using interleaving). They can also be hierarchically nested;
- *Signals*. These are the variables, and are associated with finite data types. It is possible to assign to a signal an enumeration of values, among which a value will be non–deterministically chosen at runtime. Variables in a module are visible inside the same module and inside all the including ones. The variables outside a module can be passed as parameters;
- *Assignments*. These permit to model the behaviour of the components defining the way in which the values of the variables change. It is possible to: assign a value in the initial state (with the keyword *init*), specify the transition relation to the next state (with the keyword *next*), specify a state invariant (with a simple assignment). Of course, it is not possible to specify for the same variable both an invariant and a changing behaviour. All the assignments of the systems are viewed as parallel, and are simultaneously executed. When modules are synchronously connected, a single step

corresponds to a single step in each of the components. In asynchronous connection a single step corresponds to a single step performed by exactly one component.

It is clear from this succinct description that these two modeling languages are rather different: in fact, SMV is more suitable for the description of hardware, synchronous systems, whereas Promela is tailored for describing asynchronous, complex systems like software ones.

3.1.2 Property Specification

Together with model specification, also the property specification should be provided to the model checker. We will briefly describe the syntax and semantics of a linear and a branching time logic. The linear logic assumes that time is linear, thus its semantics is defined on paths, i.e. on sequences of states. Instead, the branching time logic has a semantics defined on trees. However both of them allow for the specification of properties that regard the relative *ordering* of events, but without any explicit reference to their precise timing. The time is assumed to be discrete, with transitions corresponding to the advance of a single time unit. Hence, the system is observable only at fixed time points.

We will focus in particular on the Linear Time Logic (LTL) introduced by Pnueli [82] and on the Computation Tree Logic (CTL) proposed by [87]. Both have their semantics defined on a *Kripke structure*, i.e. a LTS extended with a set of atomic propositions AP and a labeling function L that associates to each state of the LTS the subset of atomic propositions holding true in that state.

Definition 1 *Kripke structure*
Let $LTS = (S, S_0, \Delta)$ a labeled transition system and AP a set of atomic proposition. The associated Kripke structure is

$$K = (S, S_0, \Delta, AP, L)$$

where $L : S \to 2^{AP}$

Hence, if $a \in L(s)$ then the atomic proposition a is true in the state s.

Linear Temporal Logic

LTL formulae over a transition systems are built with the atomic propositions $a \in AP$, the boolean connectors $\neg$ and $\wedge$ and two temporal modalities, e.g. $\bigcirc$(meaning "next") and $\mathbf{U}$ (meaning "until"). The formula $\bigcirc\phi$ holds in the current moment if ϕ holds in the next one. The formula $\phi_1 \mathbf{U} \phi_2$ holds in the current moment if there is some moment in the future in which ϕ_2 holds and ϕ_1 holds in all the moments before.

Definition 1 *LTL Syntax*
Let $a \in AP$ be an atomic proposition. LTL formulae are defined according to the following grammar:

$$\phi ::= True \mid a \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 \mathbf{U} \phi_2$$

Of course, it is possible to derive the other propositional logic operators from the $\neg$ and $\wedge$. More important, it is possible to derive the temporal modalities $\Diamond$ ("eventually") and $\Box$ ("always") as follows:

- eventually: $\Diamond\phi \stackrel{def}{=}$ True $\mathbf{U}\phi$
- always: $\Box\phi \stackrel{def}{=} \neg\Diamond\neg\phi$

From these it is possible to derive other temporal modalities such as

- "infinitely often ϕ": $\Box\Diamond\phi$

- "eventually forever ϕ": $\Diamond\Box\phi$

We will now define the satisfaction relation $\models$ for the infinite words over 2^{AP}, i.e. to the sequence (sets) of labels associated to the states that belongs to a path π in the Kripke structure K.

Definition 2 *Satisfaction Relation for LTL*
Let $\sigma = A_0A_1A_2\cdots \in (2^{AP})^\omega$ be a sequence of words over 2^{AP}, and $\sigma[j...] = A_jA_{j+1}A_{j+2}...$ is the suffix of σ starting in the $(j+1)$-st symbol A_j.
The satisfaction relation $\models$ is defined by

$$\begin{array}{ll}
\sigma \models \mathit{True} & \\
\sigma \models a & \mathit{iff}\ \ a \in A_0 \\
\sigma \models \phi_1 \wedge \phi_2 & \mathit{iff}\ \ (\sigma \models \phi_1) \wedge (\sigma \models \phi_2) \\
\sigma \models \neg\phi & \mathit{iff}\ \ \sigma \not\models \phi \\
\sigma \models \bigcirc\phi & \mathit{iff}\ \ \sigma[1...] = A_1A_2\ldots \models \phi \\
\sigma \models \phi_1 \mathbf{U} \phi_2 & \mathit{iff}\ \ \exists\, j \geq 0 : ((\sigma[j...] \models \phi_2) \wedge (\sigma[i...] \models \phi_1\ \forall\, 0 \leq i < j))
\end{array}$$

For the derived operators the satisfaction relation is:

$$\begin{array}{ll}
\sigma \models \Diamond\phi & \mathit{iff}\ \ \exists\, j \geq 0 : \sigma[j...] \models \phi \\
\sigma \models \Box\phi & \mathit{iff}\ \ \sigma[j...] \models \phi\ \forall\, j \geq 0 \\
\sigma \models \Box\Diamond\phi & \mathit{iff}\ \ \overset{\infty}{\exists} j : \sigma[j...] \models \phi \\
\sigma \models \Diamond\Box\phi & \mathit{iff}\ \ \overset{\infty}{\forall} j \sigma[j...] \models \phi
\end{array}$$

Where $\overset{\infty}{\exists}$ means $\forall i \geq 0 : \exists j \geq i$, i.e. "for infinitely many j $\in \mathbb{N}$". Similarly $\overset{\infty}{\forall}$ means $\exists i \geq 0 : \forall j \geq i$, i.e. "for almost all j $\in \mathbb{N}$" The satisfaction relation permits to define the semantics of a LTL property ϕ as the language $Words(\phi)$ that contains all the infinite words over the alphabet 2^{AP} that satisfy ϕ:

Definition 3 *LTL Semantics over Words*
Let ϕ be an LTL formula over AP.

$$Words(\phi) = \{\sigma \in (2^{AP})^\omega : \sigma \models \phi\}$$

Then it is possible to define the semantics of LTL w.r.t a Transition System. This is assumed, for simplicity, to be without terminal states.

Definition 4 *LTL Semantics over a Transition System*
Let K be a Transition System defined by the Kripke structure $K = (S, S_0, \Delta, AP, L)$ and Φ a LTL-formula over AP.

- *For a path π of K the satisfaction relation is defined as*

$$\pi \models \phi \quad \text{iff} \quad \mathit{trace}(\pi) \models \phi.$$

- *For a state $s \in S$ the satisfaction relation is defined as*

$$s \models \phi \quad \text{iff} \quad \forall \mathit{paths}\sigma \in \mathit{Paths}(s)\ \sigma \models \phi.$$

- *For the transition system K the satisfaction relation is defined as*

$$K \models \phi \text{ iff Traces(K)} \subseteq Words(\phi).$$

From these definition follows that a transition system K satisfies ϕ iff $\pi \models \phi\ \forall \pi \in Paths(K)$. Thus,

$$K \models \phi \text{ iff } s_0 \models \phi \forall\, s_0 \in S_0.$$

Computation Tree Logic

CTL has a syntax that distinguishes between state and path formulae. The former involve properties of a single state, such as assertion over the atomic propositions or the state branching structure. The latter involve features of a path (i.e. an infinite sequence of states), such as temporal properties.

Definition 1 *CTL Syntax*
Let $a \in AP$ be an atomic proposition, and ϕ a path formula.
State formulae are defined according to the following grammar:

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid \exists \phi \mid \forall \phi$$

Path formulae are formed as follows:

$$\phi ::= \bigcirc \Phi \mid \Phi_1 \mathbf{U} \Phi_2$$

The temporal operators $\bigcirc$(Next) and **U** (Until) have the same meaning as in LTL. However to obtain a legal state formula they should be preceded by a universal ($\forall$) or an existential ($\exists$) path operator. The boolean operators are defined in the usual way. It is possible to derive other temporal modalities such as "eventually" or "always" as follows:

- eventually: $\exists \Diamond \Phi = \exists(true \mathbf{U} \Phi)$
  $\forall \Diamond \Phi = \forall(true \mathbf{U} \Phi)$
- always: $\exists \Box \Phi = \neg\forall\Diamond\neg\Phi$
  $\forall \Box \Phi = \neg\exists\Diamond\neg\Phi$

The CTL formulae are interpreted over the states and paths of a transition system described by a Kripke structure. The semantics of CTL is defined by both a satisfaction relation for the state formulae and a satisfaction relation for the path formulae. Writing $s \models \Phi$ means that s satisfies Φ if and only if the state formula Φ holds in state s. Similarly, $\pi \models \phi$ means that the path π satisfies the path formula ϕ. $Paths(s)$ represent the set of all the paths starting from state s.

Definition 2 *Satisfaction Relation for CTL*
Let $a \in AP$ be an atomic proposition, $K = (S, S_0, \Delta, AP, L)$ a Kripke structure, and s a state of S.
Let Φ and Ψ be two CTL state formulae, and ϕ be a CTL path formula.
The satisfaction relation $\models$ is defined for state formulae by

$$\begin{aligned}
&s \models a && \text{iff } a \in L(s)\\
&s \models \neg \Phi && \text{iff } \neg(s \models \Phi)\\
&s \models \Phi \wedge \Psi && \text{iff } (s \models \Phi) \wedge (s \models \Psi)\\
&s \models \exists \Phi && \text{iff } \exists \pi \in Paths(s) : \pi \models \Phi\\
&s \models \forall \Phi && \text{iff } \pi \models \Phi \ \forall \pi \in Paths(s)
\end{aligned}$$

For a path π, the satisfaction relation $\models$ is defined for a path formula by

$$\begin{aligned}
&\pi \models \bigcirc \Psi && \text{iff } \pi[1] \models \Psi\\
&\pi \models \Phi \mathbf{U} \Psi && \text{iff } \exists j \geq 0 : (\pi[j] \models \Psi \wedge (\pi[k] \models \Phi \, \forall 0 \leq k < j))
\end{aligned}$$

where for path $\pi = s_0 s_1 s_2 \ldots$ and integer $i \geq 0$, $\pi[i]$ denotes the $(i+1)$-th state of π, i.e. $\pi[i] = s_i$.

Then it is possible to define the semantics of CTL on a Transition System.

Definition 3 *CTL Semantics*
Let K be a Transition System defined by the Kripke structure $K = (S, S_0, \Delta, AP, L)$. *The satisfaction set* $Sat_K(\Phi)$ *for the CTL-state formula* Φ *is defined by:*

$$Sat_K(\Phi) = \{s \in S | s \models \Phi\}$$

The transitions system K satisfies the CTL formula Φ *if and only if* Φ *holds in all initial states of K:*

$$K \models \Phi \text{ iff } s_0 \models \Phi \ \forall\, s_0 \in S_0$$

.

3.2 Choosing the Model Checker

CTL and LTL not only differ in the assumption made on the underlying time structure, but also on the expressive power they have. It is worth noting that their expressiveness is sometimes not comparable, in the sense that some properties can be defined only in one of them [110, 111]. For instance, *fairness* properties (see section 3.2.1) are easily expressed in LTL, and can be checked using the same algorithm used for checking other LTL properties. This does not apply to CTL. Conversely, in CTL it is possible to easily define the *restart* property (i.e. "From any state, it is possible to get to the reset state") that in LTL is not expressible. The diversity of their inner structure is reflected also in the difference between the algorithms used to model check these properties. We will consider the ones implemented in the most prominent tools developed for these tasks, e.g. Spin [12] and NuSMV [107]. In brief, the first is an explicit-state checker based on the transformation of the (negated) LTL formula in a Büchi automaton that is synchronously executed with the system automaton. On the resulting automaton the problem of the acceptance of the empty language is resolved (see section 3.2.1). NuSMV is a symbolic model checker based on the construction of Ordered Binary Decision Diagrams (OBDD). The model checking problem is solved using a fixed-point based, recursive traversal of the parse tree of the state formula. Additional algorithms are needed for fairness checking and counterexample building (see [91] for details).

A great debate is still open on which of the two approaches is the best, both in theory and in practice. But until now, no clear response is available [112, 113]. From a theoretical point of view, the complexity in time and space should be compared. The LTL model has a time complexity linear in the dimension of the model but exponential in the length of the formula. Instead, CTL model checking has a time complexity that is linear both in the length of the model and in the formula. However, in practice, the running time of LTL model checkers is comparable to the CTL checkers over a wide range of problems [114, 115]. This is due to several reasons, including that useful LTL formula are generally short, and that very rarely the conversion of a LTL formula shows a worst-case (exponential) behaviour.

The principal differences between linear time and branching time logics are summarized in table 3.1, adapted from [91]

Similar consideration can be made for the space complexity. Indeed, the LTL model checking is affected by the state-space explosion problem (see Section 3.3) whereas symbolic model checking offers a very effective strategy to deal with this problem. However, several techniques like partial order reduction and on-the-fly space construction methods mitigate this effect in the LTL checking [116, 117]. Moreover, comparison on the effective amounts of memory (bytes) and time used to solve similar problems with both the techniques show often similar results for both [118]. It should be noted also that, if more than a counterexample is needed, with LTL checking no additional searches are needed (counterexamples are the traces generated during verification). On the contrary,

Table 3.1. The principal differences between linear time and branching time approaches. Adapted from [91].

	Linear time	Branching time
"Behaviour" in a state s is	path-based: trace(s)	state-based: computation tree of s
Temporal logic	LTL: path formulae ϕ $s \models \phi$ iff $\forall \pi \in Paths(s), \pi \models \phi$	CTL : state formulae plus existential path quantification $\exists$ universal path quantification $\forall$
Complexity of the model checking problems	$\mathcal{O}(\lvert K\rvert \cdot 2^{\lvert\phi\rvert})$	$\mathcal{O}(\lvert K\rvert \cdot \lvert\Phi\rvert)$
Counterexample generation and added complexity	During verification none	Additional search needed $\mathcal{O}(\lvert K\rvert)$
fairness	supported by the LTL language	special techniques needed
expressiveness	Not Comparable	

with CTL the generation of a counterexample is not part of the verification process, and requires an additional exploration of the whole space. Also, the structure of the CTL counterexamples is linear only for a small subset of universally quantified formulae, otherwise it has much more complex structures that make more difficult their exploitation w.r.t the linear trace obtained with the LTL checking [119].

Concluding, a clear answer is not available, but in general CTL symbolic verification tends to be adopted in the hardware verification domain, where variables are often very simple (bit, boolean or bit-arrays) and more suitable for OBDD representation. On the contrary, LTL model checking is more used in the software domain, where the variable structures are generally more complex. Moreover, asynchronous processes tend to be dominant in software applications, condition that is very well exploited by the partial reduction order techniques, whereas hardware systems tend to be synchronized and clocked.

Table 3.2. The principal differences between the Spin and NuSMV model checker.

	Spin	NuSMV
Model specification language	Imperative: Promela	Descriptive: SMV
Property specification languages	LTL ω-properties	CTL LTL
Verification	Explicit: Automaton-theoretic	Symbolic: OBDD
Number of counterexamples	≥ 1	1
Type	Trace	Tree
Fairness support	yes	yes

In this thesis we decided to use the Spin model checker. The reasons for this choice are based on the following considerations:

1. the biological system we want to model does not need an explicit time representation, and is inherently asynchronous;
2. the properties to express are very simple, thus not pushing toward a worst-case behaviour if expressed with LTL;
3. the LTL automaton-theoretic approach implemented in Spin permits to generate more than a single counterexample, whereas NuSMV CTL based approach allows to generate only a single counterexample;
4. the specification language of Spin is more suitable for the description of this system: the representation in the SMV language would have been clumsy and rather difficult to obtain, whereas the Promela translation was immediate and straightforward;
5. Spin comes with many facilities that allows to fine tune the search for counterexamples in the state-space, such as the possibility to decide the number of the extracted counterexamples, or the amount of memory needed for the structures;
6. Spin can be configured to perform partial explorations of the state-space, that is often the only solution to tackle very large problems, such as the biological ones;
7. the Promela language offers support for the specification of simple state properties, that can be easily and directly included into the system model with the *assert* instruction.

3.2.1 Model Checking of LTL Properties

Since this thesis is based on the use of the model checker Spin, we will focus only on the model checking methodology that Spin implements, i.e. the Automata-based approach.

We will briefly describe how model checking of LTL properties is performed using an Automata-based approach. Recall that the LTL semantics defines for a formula ϕ a language $Words(\phi) \subseteq (2^{AP})^{\omega}$ (see definition 3). It is possible to demonstrate that the set $Words(\phi)$ is an $\omega-regular$ language, hence it may be represented by a Non–deterministic Büchi Automaton (NBA) [91]. Instead of proving that $K \models \phi$, the complement of ϕ is considered, e.g. $\neg\phi$ and a path π such that $\pi \models \neg\phi$ is searched in K. If this path is found, then $TS \not\models \phi$ and a prefix of π is returned as a counterexample. More precisely:

$$\begin{aligned} K \models \phi \;\; & \text{iff} \;\; Traces(K) \subseteq Words(\phi) \\ & \text{iff} \;\; Traces(K) \cap ((2^{AP})^{\omega} \backslash Words(\phi)) = \emptyset \\ & \text{iff} \;\; Traces(K) \cap Words(\neg\phi) = \emptyset. \end{aligned}$$

Let $\mathcal{B}_{\neg\phi}$ be the NBA with accepted language $\mathcal{L}_{\omega} = Words(\neg\phi)$, then it follows

$$K \models \phi \text{ if and only if } Traces(K) \cap \mathcal{L}_{\omega}(\mathcal{B}_{\neg\phi}) = \emptyset.$$

If K has a finite number of states, then it can be transformed into an equivalent Büchi automaton $\mathcal{A}_K$. Its synchronous product with the NBA for the LTL formula is indicated as $\mathcal{A}_K \oplus \mathcal{B}_{\neg\phi}$.

This process shown in figure 3.2: the model, specified in a high level language, is translated into the corresponding LTS and eventually into the corresponding Büchi automaton $\mathcal{A}_K$. Similarly, the specification ϕ is negated and then translated into the corresponding NBA $\mathcal{B}_{\neg\phi}$. Then the synchronous product is calculated and the corresponding automaton is searched for an acceptance path π. If this path is not found, it is proved that the system model satisfies the requested property. Otherwise a counterexample (e.g. a prefix of π) is returned.

Hence, the problem of model checking a LTL formula is transformed into the problem of verifying if the product automaton accepts the empty language. In the NBA this corresponds to find a path from the initial state to a cycle containing a final state. This

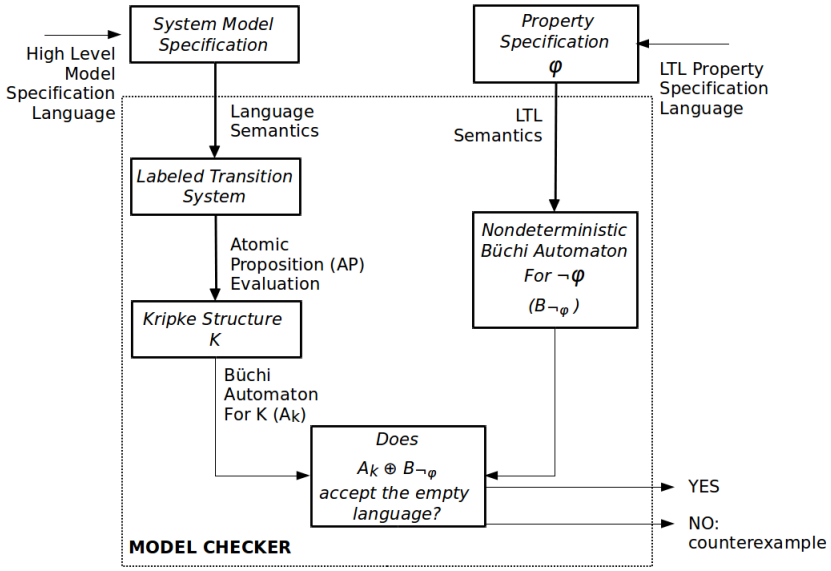

Fig. 3.2. A schematic view of the Automata Based Model Checking approach

is a reachability problem on a finite state/transition graph, and can be efficiently solved with a linear (in the size of the graph) procedure.

However, the translation of a LTL formula into the corresponding NBA has time and space complexity $\mathcal{O}(2^{|\phi|})$. This affects the overall complexity, that is linear in the size of the model and exponential in the length of the formula, i.e. $\mathcal{O}(|\mathcal{A}_k| \cdot 2^{|\phi|})$ (see [91] for further details).

An efficient algorithm for the LTL model checking problem is the Nested Depth First Search, that has been implemented in Spin [12].

Correctness claims

We will briefly review the categories of claims that are relevant for the verification of concurrent systems, and how they can be specified in Spin. Interesting properties might be classified as follows:

- *Safety.* They require, informally, that "nothing bad should happen". To this class belong both *state properties* and *path properties*. The firsts, also called *invariants*, are linear time properties given by a condition ϕ that should hold **in all** the reachable states. They are the simplest (yet powerful) type of properties, and can be verified in a time that is linear in the length of both the transition system and the formula ϕ. Path properties are a generalization of invariants, since they express conditions on finite path fragments rather then on single states;
- *Liveness.* These properties constrain the infinite system behaviour. A typical liveness property is the starvation freedom requirement (each waiting process will eventually be scheduled);
- *Fairness.* This class is very important in reactive system, since it helps in ruling out unrealistic behaviours. It is possible to define three types of fairness assumption, based on the strength of the requirement, i.e. Unconditional, Strong and Weak Fairness. Fairness claims constrain actions along infinite behaviours.

These classes may be specified in Spin using different combinations of the tools hereafter described.

- *The assert instruction.* An *assert* instruction is always executable, and permits to specify state invariants that can be easily verified with a reachability check;

- *LTL formulae.* It is possible to express a wide range of correctness claims, including fairness, using a LTL syntax that comprise operators such as Always [], Eventually <>, and Until **U**. The set of operators may be extended with the Next operator (X), but this should be used with caution and it is by default disabled since it may conflict with the Partial Order Reduction algorithm (see [12] for details). The LTL formula is converted into an automaton (using the *never* instruction) and the synchronous product of the system model and the property automaton is checked as described in section 3.2.1;
- *The never claim.* This instruction permits to directly specify the NB automaton that will be synchronized with the system model to perform the checking. The *never* claim may be used to specify all those ω-properties that are not expressible with the LTL syntax. *Never* claims are intended to express a behaviour that should never happen in the system, and Spin flags as errors the executions that exactly matches the behaviour specified by the claim;
- *Meta labels.* Spin permits to use special meta labels to mark states of interest, thus allowing the detection of the presence/absence of infinite, cyclic behaviours for liveness properties checking. In particular it is possible to specify:
  - *accept labels.* Usually, these labels are used to formalize Büchi acceptance conditions: they are generated automatically by the LTL translator and placed into the corresponding *never* claim. However they can be used elsewhere to prove, for instance, LTL liveness properties;
  - *end labels.* These are used to identify acceptable (i.e. non-deadlock) end states;
  - *progress labels.* These are used to identify infinite cycles that are allowed in the model. This can used to detect starvation cycles.

Lastly, it should be noted that in Spin there is support only for the weak fairness assumption.

3.3 Complexity Issues

Here we will briefly address the nemesis of model checking, i.e. the so called state-space explosion problem, and the techniques implemented in Spin to contain the exponential blow–up of memory and time that are connected with this issue.

The state-space of a system is the set of all the possible states of the associated Kripke structure. If we consider a finite system with variables defined on a finite domain, we might have a state for each possible value combination of all the variables: for N variables having a domain of k values the number of states is k^N. This exponential blow-up is exacerbated by: *i)* the presence of parallel components, for which the resulting state-space is given by the product of the dimensions of the state-space of each component, and *ii)* the presence of shared channels, that further increase the number of possible states. Thus, it is evident that even very small systems may require a huge amount of memory to be exhaustively verified.

As already said, Spin implements a Nested Depth First Search, that in its original version was first described in [120]. The basic algorithm is modified to perform on-the-fly verification. In this approach, the construction of the automaton for the LTL formula is performed simultaneously with, and guided by, the generation of the model automaton. Thus, it is possible to find that a property does not hold by constructing only a fraction of the model.

Anyway, this is not enough to deal with real problems. Hence, several other methods have been implemented in Spin for tackling the exponential explosion of the states. They rely on optimization techniques aimed at reducing *i)* the number of reachable states *ii)* the memory required for storing each of them. The partial order reduction algorithm and the statement merging technique fall in the first category, whereas the

lossless memory reduction strategies (such as the minimal automata representation and the collapse compression) fall in the second one.

In addiction to the aforementioned techniques, Spin offers also the possibility to perform approximated searches on the state space. Indeed, real problems may have a number of states so high that an exhaustive search is practically infeasible (see [121]). The *bit-state hashing* strategy permits to partially explore the state space: if no counterexample is found, nothing can be stated on the property satisfaction. However, if a counterexample is found, then we can surely assert that the property is not satisfied. Recently, Holzmann et al. have developed and tested a multi-cpu bitstate hashing approach that, together with a random transition ordering (to solve non–deterministic choices) and other tuning strategies, permits to effectively explore different parts of a very large state space. The *swarm* technique is described in full detail in [122], and has been largely exploited in this thesis when dealing with very large biological models. A complete description of all these methodologies is outside the scope of this thesis, for all the technical details please refer to the Spin Primer and Reference Manual [12].

3.4 Chapter Summary

In this chapter we introduced the prominent Model Checking methodology. After a brief description of its historical roots, we addressed the problem of specifying the system model and the property that it should fulfill. We started describing a low-level mathematical formalism used for the model specification, i.e. the Labeled Transition System (LTS), and after we outlined the features of two high-level languages, Promela and SMV, that both have their semantics defined on a LTS.

Afterwards we described two property definition languages, i.e. the Linear Temporal Logic (LTL) and the Computation Tree Logic (CTL). For both of them we gave a precise definition of their semantics.

Then, we addressed the delicate issue of the choice of a model checker, and since we decided to not model time explicitly, we restricted our possibilities only to two prominent tools, Spin and NuSMV (whose languages are respectively Promela and SMV). The former is able to verify ω-regular properties (which include LTL ones) with an automaton-theoretical approach, whereas the latter performs symbolic verification of CTL properties using binary decision diagrams (BDD). Strengths and limits of both the strategies are discussed, and the choice of the model checker Spin is motivated. After, we outlined the Spin automaton-theoretical approach to verification, together with the Promela instructions that supports the properties specification. Lastly, we discussed the computational issues related to the so-called state-space explosion problem, and we briefly outlined the optimization strategies implemented by Spin.

4

Model Checking in the Systems Biology Era

Systems Biology can be described as an approach to the understanding of life through the study of how the properties of biological systems arise through the multi-level interactions between their components. In particular, the focus is on the construction of models of the biological systems that can help in understanding the mechanisms suggested by the experimental data. These models are becoming increasingly large and complex, sometimes involving information at the whole genome scale or at multiple levels of detail. Hence, adequate mathematical and computational tools are required for their analysis and simulation.

However, mathematical and computational approaches provide techniques for modeling these systems that are widely different and often complementary. Indeed, mathematical models are focused on a precise (denotational) description of the system, based on equations that may simulated and possibly solved. On the contrary, computational models are based on an operational semantics that captures and mimics (through the execution) the model behaviour.The differences between these two strategies lie both in the size of the systems that can be modeled, both in the quality of the obtained results.

Mathematical models can precisely describe quantitative relationships between variables such as metabolite concentrations, or gene expression levels. However **quantitative modeling** is constrained both by the availability of biological precise data both by the number of variables and equations that may be effectively treated.

Instead, computational models may go beyond this limitation and work also on partial data, and on more complex systems. This **qualitative modeling** approach relies on the goodness of the underlying abstraction, rather than on the faithfulness of the mathematical implementation. The obtained results, although not precise, may give interesting hints on the features and behaviours of the modeled biological system. The differences between the computational and the biological models may be used to build new hypotheses, to refine the model, and even suggest new experiments that can help in validate or reject the model. This methodology, called by Fisher and Henzinger **executable biology** [8], is based on the close interplay between *in silico* simulations and biological validation of the data, as shown in figure 4.1.

It should be noticed that in this repeated process of model building and refining, automated formal verification methods, such as the Model Checking, play a fundamental role, although a shift in the perspective should be performed [9]. In fact, in the hardware and software domains the verification process is used to ascertain that the system built from the model will have the desired behaviour. If the verification fails, the system is searched against errors (using the counterexamples generated by the checker) and eventually modified. In biology, the process is the opposite: the (biological) system already exists and is correct by default, and the model is built to verify that the hypothesis made on its functioning is able to catch some features of the underlying, partially unknown, mechanisms.

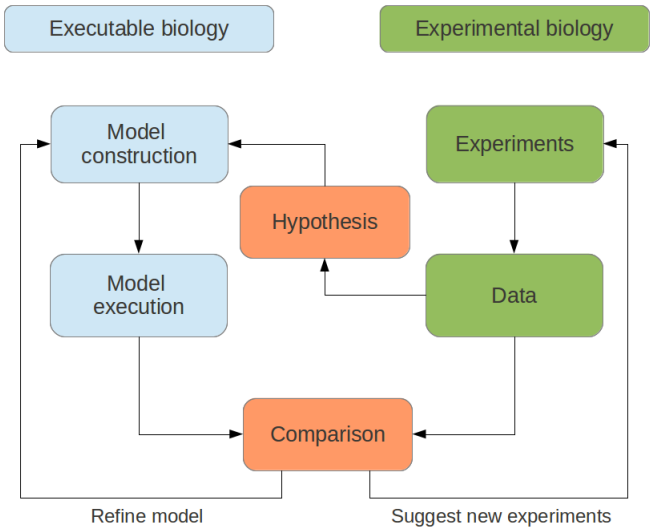

Fig. 4.1. The executable biology framework: biological knowledge and experimental data are used to draw an hypothesis, from which a computational model is built. The model is executed and the results are compared to the data, possibly resulting in a model refinement or in suggestions for new experiments (adapted from [8]).

Indeed, Model Checking is a paradigm that may fit very well the Systems Biology approach [10]. This is due to several reasons:

- It enforces model abstraction. Biological systems are usually very complicated, and abstraction is a crucial strategy to address this complexity.
- It is able to deal with the concept of multi-level functionality that is fundamental in systems biology [123].
- It offers precisely defined formal languages, with a semantic that permits to describe both the constituent components and the way in which they interact. The overall behaviour will emerge from the interplay of the components, a very close view to the systems biology paradigm.
- It can infer both qualitative or quantitative properties of *all* possible executions of a model.
- It can give partial information through simulations of the model, i.e. a partial exploration of the execution set.

We will now review some successful applications of Model Checking techniques to biological problems, with the aim to better describe the context in which this thesis is placed.

4.1 Modeling Biological Systems

As described in chapter 2, the typical objects to model in the framework of Systems Biology are **networks** of interacting elements, that evolve in time. Depending on the features of the network and on the desired properties, various approaches may be used. They might be classified according to the different granularity that their semantics is able to capture, as proposed in Brim et al. [9] and hereafter described.

A modeling semantics should be able to describe the dynamics of the system taking in consideration:

- *the components*. They are interpreted as variables, that can be either *discrete* or *continuous*, depending on the selected level of abstraction, the availability of data and the model dimensions;

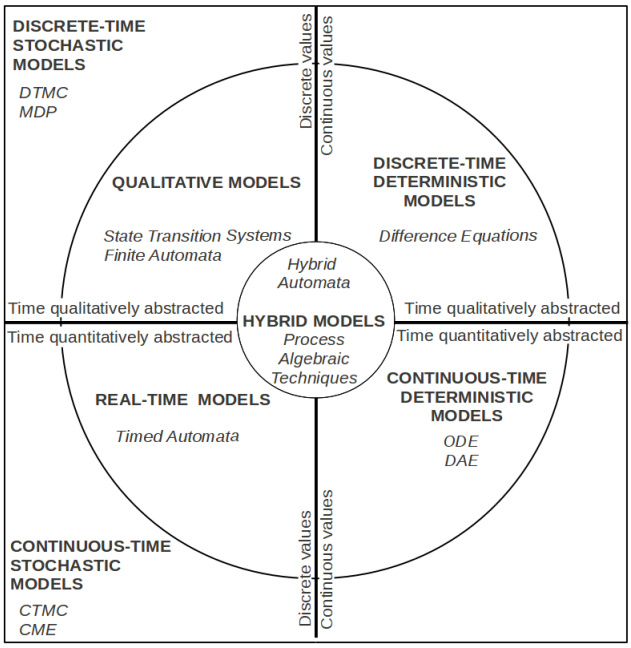

Fig. 4.2. Model types sorted according to different level of details captured in their semantics (adapted from [9]). DTMC: Discrete-Time Markov Chains, MDP: Markov Decision Processes, CME: Chemical Master Equation, CTMC: Continuous-Time Markov Chains, ODE: Ordinary Differential Equations, DAE: Differential Algebraic Equations

- *the interactions*. They are interpreted as rules that specify the dynamical changes of the variables. The time in which these interactions take place may be modeled either as a dense, *continuous* entity or may be represented as a *discrete* quantity. Lastly, the execution of interactions may be *stochastic*, i.e. it may proceed with a certain degree of uncertainty that reflects the assumption of a noisy environment;

According to these possible semantics, the various modeling approaches may be classified as shown in figure 4.2. On the right side of this scheme, there are models involving continuous component quantities and deterministic interactions. These models are mathematical and inherently quantitative, since are based on a purely denotational semantics [8]. They are generally based on ordinary differential equations (ODEs) and/or differential algebraic equations (DAEs). On the left side of the scheme the discrete-value models can be found. If time is treated as a discrete quantity, we have either qualitative executable models based on Finite State Machine representations or stochastic models such as Discrete-Time Markov Chains (DTMC) or Markov Decision Processes (MDP). For time quantitatively abstracted, we have executable models based on Timed Automata representations or, if probability is considered, stochastic approaches based on the Chemical Master Equation (CME). This can be converted into an executable model using Continuous-Time Markov Chains (CTMC). Lastly, Hybrid Models, such Hybrid Automata or Process Algebraic Techniques, mix discrete and continuous representation for both variables and time dynamics.

4.2 Specification of Biological Properties

The specification of interesting biological properties is of course dependent on the chosen modeling approach. It is possible to distinguish between qualitative and quantitative properties specification.

4.2.1 Qualitative Properties

These are properties for which the information on time is considered implicitly. We have described in chapter 3 two of the most prominent temporal logic, i.e. Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). LTL and CTL suffice to express the major part of the interesting biological property, that can be classified as follows (see [9]):

1. *reachability properties*. They express reachability of specified concentration levels in given model variables, and can be used for expressing global bounds of attainable concentrations;
2. *temporal ordering of the events*. These properties may be used to capture the qualitative temporal patterns in the dynamics of the inspected variables;
3. *variable correlations*. These properties are intended to catch cooperation and dependencies in biological processes, such as co-expression of certain genes.
4. *stability properties*. They can be used to analyse the presence of stable concentration levels for a certain species.
5. *monotonous trends*. These are very important properties in biology, since monotonicity captures strong increasing or decreasing dynamics of individual species.
6. *oscillation*. These properties capture behaviours that are typical of biological systems (such as circadian or ultradian clocks).

It is worth to note that the majority of these properties may be expressed by means of very simple logic formulae, often involving only state invariants, as described, for instance, in [9, 124, 125].

4.2.2 Quantitative Properties

Quantitative properties involve reasoning on the dynamics of the system over time, such as the

1. *energy consumption rates* of the involved reactions;
2. *metabolite dynamics*, i.e. production/degradation rates of the involved metabolites;
3. *equilibrium states* of the whole system.

To this aim several formalisms have been proposed that allow to deal with these quantitative aspects of biological systems. They usually extend the LTL or CTL logics, and can be roughly divided into deterministic and stochastic logics. The formers are characterized by a quantitative notion of time, such as the time extension of CTL called Timed Computational Tree Logic (TCTL) [126] and its simplified version used in the tool UPPAAL [127].

Stochastic logics allows to specify the probability and performance measures on Markov Chains. In the case of DTMCs the PCTL logics, a Probabilistic extension of CTL, can be used [128] It is worth noting that PCTL is a discrete-time logic and thus the path formulae are interpreted over discrete time steps. To formalize properties of CTMC, Continuous Stochastic Logic (CSL) [129] has been introduced.

4.3 Related works: an overview

A broad range of different modeling approaches has been successfully applied to the verification and simulation of biological networks. Table 4.1 shows a comparison between a significant subset of relevant examples. The description of the property and the model types are based on the classifications given in sections 4.1 and 4.2.

Table 4.1. A comparative view of various modeling approaches applied to different Biological Systems. The size of the obtained model depends both from the biological system and from the modeling technique. Where not otherwise stated, size is specified in terms of number of components/number of interactions. Their type is immediately deduced from the associated biological network type; a indicates the number of equations used to describe the components of the model; b indicates the number of cells.

Biological System		Properties		Model			Tool	Reference Number
Description	Network Type	Type	Specification Language	Type	Specification Language	Size		
Flower morphogenesis in *A. Thaliana*	Gene Regulatory Network	stability	LTL	Qualitative	LTS	6/34	Spin	[130]
Mammalian cell cycle	Gene Regulatory Network	stability	CSL	Continuous-Time Stochastic Model	CTMC	2	PRISM	[9]
Mammalian cell cycle	Gene Regulatory Network	reachability, stability, temporal ordering	CTL	Qualitative	CTS	532/732	NuSMV	[124]
Nutritional stress response in *Escherichia coli*	Gene Regulatory Network	reachability, stability	CTL	Qualitative	QTS	7	GNA NuSMV	[131]
Tryptophan biosynthesis in *Escherichia coli*	Metabolic Pathways Gene Regulatory Network	reachability, variable correlation		Qualitative	Petri Nets	10/6		[132]
Ammonium Transport in *Escherichia coli*	Reaction Network	reachability, stability	LTL	Qualitative	RATS	9/5	BioDiVinE	[9]
λ-repressor protein CI in *Escherichia coli*	Reaction Network	metabolite dynamics	Process Algebra	Continuous-Time Stochastic Model	CTMC	5/3	Bio-PEPA PRISM	[133]
Toy model	Reaction Network	oscillation	PCTL	Qualitative	DTMC	6/6	PRISM	[134]
Delta-Notch protein signaling	Signaling Network	equilibrium states		Qualitative	Hybrid Models	4^b		[135]
Fibroblast Growth Factor	Signaling Network	metabolite dynamics	CSL	Continuous-Time Stochastic Model	CTMC	10/38	PRISM	[136]
Mitogen-activated protein kinase (MAPK) cascade	Signaling Network	reachability, stability, temporal ordering	CTL	Qualitative	LTS	9	BIOCHAM	[125]
Signal transduction in primary human chondrocytes	Signaling Network	metabolite dynamics	TCTL	Quantitative	Timed Automata	10/13	ANIMO UPPAAL	[137, 138]

Gene Regulatory Networks (GRNs) have been very well studied in the model checking context because the interaction between components may be easily represented by their presence/absence, i.e components are represented by a boolean variable and the interactions are represented by logical rules on their values. Following this approach Chabrier et al. [124] successfully modeled a very large network, involving more than 500 genes. They defined the Concurrent Transition Systems (CTS), a modeling syntax that permits to define modular systems, and is after translated in the NuSMV language. They checked using CTL reachability, stability and temporal ordering properties.

A similar study on a much smaller (although real) biological system has been performed by Bošnački et al. [130] and uses the LTL specification syntax and the Spin model checker to verify stability properties.

Lastly, a Petri Net approach that integers metabolic pathways knowledge and GRNs has been proposed by Simao et al. [132], and reachability and variable correlation properties have been explored. However, also in this case the size of the model is small.

When a quantitative approach is chosen, the model dimensions drop drastically. This is essentially due to lack of knowledge on the parameter values for all the interactions, and to the increased computational complexity deriving from a large model. Brim et al. [9] propose a stochastic, continuous-time model of a small regulatory network described by two equations.

They use CTMC as a modeling language and verify stability properties specified in CSL using the prominent tool PRISM [139]. An alternative approach is proposed by Batt et al. [131]: they start from a continuous-variable model described by seven equation and discretize it using Qualitative Transition System (QTS), a syntax that permits to assign to each variable a set of inequalities describing its values range. The model is then translated into the language of the Gene Network Analyzer tool [140] and again into a NuSMV model. Reachability and stability properties are verified using the CTL syntax.

Reaction Networks (RNs) are maybe the most difficult systems to study since they are not amenable to a boolean representation as the GRNs. In fact, the information on the production/consumption rates are essential to their modeling and understanding. Moreover, the real parameters governing the reactions are often unknown. A qualitative approach has been proposed in [9], where a continuous variable model of a RN is discretized using Rectangular Abstraction Transition System (RATS), a syntax based on an approach similar to the one described by Batt et al. for the GRN. The model is then converted into a Kripke structure within the tool BioDiVinE [141], and reachability and stability properties are expressed in LTL.

Bio-PEPA, a new modeling language based on Process Algebra, is proposed by Ciocchetta and Hillston [133] to describe biological networks. They use it to quantitatively model a small reaction network. The Bio-PEPA model is converted into a CTMC, for which the metabolite dynamics are analysed using the PRISM tool.

The last example involving a RN is described by Ballarini et al. [134], where a small toy model described by three equations is investigated to detect the presence of oscillation behaviour. The system is modeled in the PRISM tool using DTMCs and the oscillation properties are specified using the PCTL language.

The last type of biological system we take into account are the Signaling Networks (SNs). Also in this case, many approaches have been used. Ghosh et al [135] qualitatively investigate the equilibrium states of the Notch-Delta signaling system of four interacting cells using Hybrid Models.

Also Fages et al. [125] qualitatively describe a SN using the BIOCHAM [142] tool. They investigate reachability, stability and temporal ordering properties of a SN. The properties are expressed in CTL and the network is modeled as a LTS.

A quantitative approach is described in Heath et al. [136], where they use the tool PRISM to obtain a quantitative description on a not-too-small SN. They use CTCM to

model the network and the CSL syntax to investigate on metabolite dynamics. Lastly, Schivo et al. [137] propose a quantitative approach based on a Timed Automata (TA) representation. The system may be visually modeled and simulated using the ANIMO interface [138], and is then translated into a TA that is analysed using UPPAAL. Metabolite dynamics properties are specified using TCTL.

A complete review of all the application of Model Checking to the modeling, verification and simulation of Biological Networks is out of the scope of this thesis. The already cited papers of Brim et al. [9], Fisher et al. [8], and Melham [10] offer a comprehensive view of the state-of-the-art methodologies and techniques. The paper of Chaouiya et al. [143] describe a the Petri net approach to the analyses of different kind of networks, whereas the review of Baldan et al. [11] is focused on Petri net application to metabolic networks. Carrillo et al. [144] review the existing modeling tools.

Nonetheless, also a small subset of examples permits to draw some interesting considerations:

- Gene Regulatory Networks and Signaling Networks are widely studied, whereas Reaction Networks are less explored;
- model checking is a widely used and invaluable tool for biological systems modeling;
- both quantitative and qualitative approaches are needed to investigate a biological system, since the type of modeled properties are quite different;
- the size of the modeled systems is generally very small, and a great degree of abstraction and suitable tools are needed to deal with large models;
- qualitative approaches are generally enough to analyse a large variety of interesting biological properties;
- interesting qualitative properties may be very simply expressed both in LTL and CTL.

4.4 Chapter summary

In this Chapter we introduced the concept of executable biology as defined by Fisher in [8], and we highlighted how the model checking approach is very well suited to fit this paradigm, both for quantitative and for qualitative analyses. After, sticking to the classification proposed in [9] we described different classes of strategies, that can be categorized, based on the granularity of their semantic expressivity in discretecontinous time, discretereal variable value, and stochasticnon-stochastic representations. Furthermore, we presented both qualitative and qualitative properties that can be expressed using these methodologies. Lastly, we presented a brief review of related efforts, describing for each of the them the methodology type, the extracted properties the model dimension.

Part II

Main Theme: Model Cheking of Metabolic Networks

5

Model Checking of Metabolic Network Models

In the first part of this thesis work the Systems Biology paradigm (chapter 2) and the Model Checking approach (chapter 3) have been described, and the reasons that make the choice of model checking a proper methodology to investigate on the genotype-phenotype relationships in Genome-scale Models have been elucidated (chapter 4).

We will describe in this and in the following chapter a framework for the **verification and simulation of Genome-scale Metabolic Network models by means of Model Checking techniques**. The basic idea relies on considering the reactions of a metabolic network as processes that try to access concurrently to the shared resources represented by metabolites. Interesting properties are expressed as safety properties on states, i.e as invariants or simple Linear Time Logic (LTL) *always* formulae, and checked or simulated using the Spin tool. A preprocessing step based on a Flux Balance Analysis (FBA) approach is described, that is essential to decide the reaction directions and/or to reduce the model size in a biologically coherent and sound manner. A complete pipeline for the conversion into a Promela model of a metabolic network, either provided in SBML format or as a stoichiometric matrix, is built using the Perl language, together with tools for extracting and analysing in a parallel environment the data coming from the Spin generated counterexamples. It is described in detail in chapter 6.

The proposed methodology is then successfully applied to the study of a Mendelian metabolic disorder, namely Primary Hyperoxaluria type I, using a genome-scale metabolic network model of the human hepatocyte cell. The obtained results are described in depth together with some considerations on the complexity issues in chapter 7.

To our best knowledge, this approach is novel and original, although perfectly in context with the Systems Biology paradigm and the executable model philosophy that have been described in chapter 4. Furthermore, we will show that the use of the Model Checking techniques for Metabolic Network analyses is worthwhile and effective, and permits to obtain invaluable qualitative information on the dynamics of large-scale models for which the detailed kinetics and dynamic information are still unavailable.

This chapter is organized as follows: in section 5.1 a toy model is described, to give a first idea of how Model Checking is used to analyse a metabolic network. Section 5.2 addresses the difference between traditional model checking and the behaviour filtering approach, and their strength and limits. In section 5.3 the translation of a network into an executable model is described in detail, together with the strategies used for its abstraction and the reduction of the overall complexity. A Flux Balance Analysis preprocessing step is also introduced and discussed. In section 5.4 the biological properties of interest are described, together with the associated invariant definitions. In section two 5.5 control tools, one based on coupling two processes that communicates through synchronized channels, one based on defining priorities among reactions, are presented, and their use to constrain the state space search is explained.

5.1 Model Checking of Metabolic Networks at a Glance

We start giving an intuitive idea of how model checking works when applied to the qualitative study of the dynamics of a metabolic network.

To model check a metabolic network, the first step needed is to build an executable model that preserves the features of the system. A metabolic network may be viewed as made up by metabolites (the components of the system) whose concentrations is changed according to the rules represented by the biochemical reactions (the dynamics of the system). The time is not explicitly modeled, but flows in a stepwise fashion: a reaction takes place in a single atomic step, if and only if all the metabolites are available. Since we deal with finite variables, we may assign a value to all the metabolite and let the network evolve, and we are guaranteed that the associated Labeled Transition System (LTS) is finite. Hence, we may observe the overall emerging behaviour by tracing the changes of the variable values in response to the performed reactions.

This idea is better understood with an example. A toy model made of only four

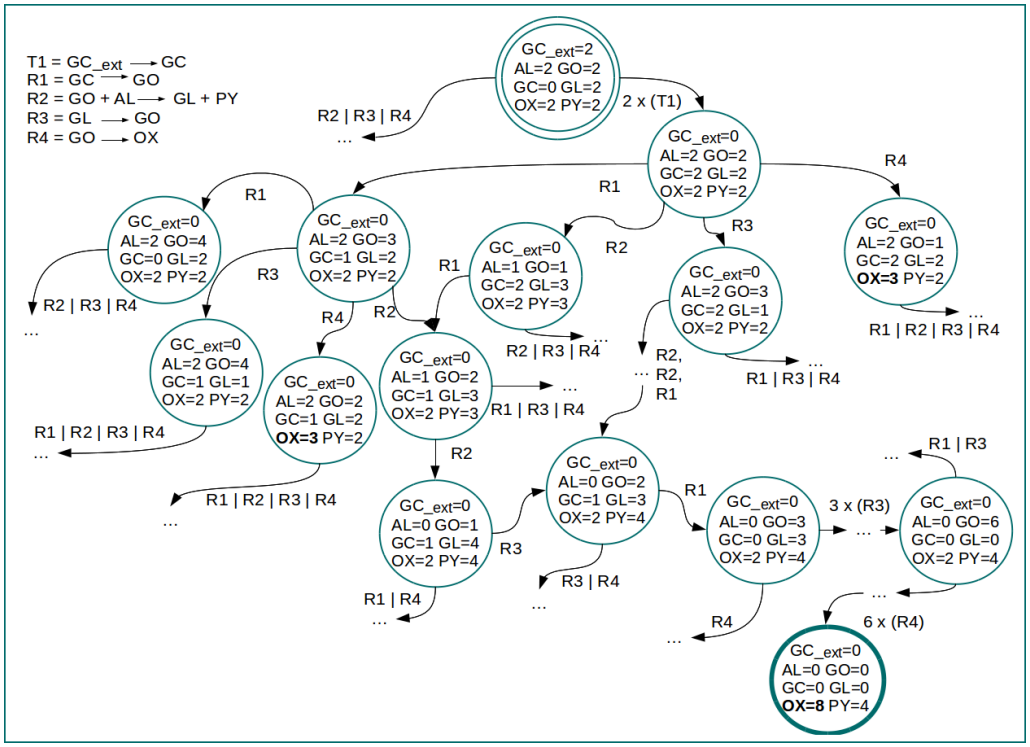

Fig. 5.1. A graphical representation of (a portion of) a LTS modeling a set of four internal ($R1$ to $R4$) and one transport ($T1$) reactions. The involved metabolite concentrations are represented by the variables: GC_{ext} (external glycolate), GC (glycolate), GO (glyoxylate), AL (alanine), GL (glycine), PY (pyruvate), OX (oxalate). Each state is represented by a circle and identified by a snapshot, i.e. the set of the variable values. The double circled state is the initial state. Arrows represent transitions between states and are labeled with the actions (i.e. the reactions) that trigger the state change. The | (*or*) operation symbol means that different actions may be executable. Sequence of actions are sometimes represented by comma-separated label sequences or by a multiplication symbol (representing the same action being executed in all the intermediate states). A terminal state, i.e. a state without a leaving transition, is indicated by a thick-bordered circle. The Oxalate (OX) concentration is indicated in bold in all those states where it increases its value. This can occur both in intermediate or in terminal states.

internal and one transport reactions is shown in figure 5.1, together with a graphical

representation of a part of the associated LTS. A state in the transition system is given by a snapshot of the variable values, each representing a concentration (in generic units) of a metabolite. The initial state is the double circled node: all the variables have been assigned the same initial value. Each path in the graph starting from the initial state is a different computation, identified by the associated trace. For instance, a computation leading to the production of eight (generic) units of oxalate is given by the trace $\pi = T1, T1, R1, R2, R2, R3, R1, R3, R3, R3, R4, R4, R4, R4, R4, R4$. Different computations may lead to the same state and different computations may start from the same state, being the model intrinsically non-deterministic. However, each computation has a different associated trace (i.e. a different sequence of actions). Please do note that it is very straightforward to determine the dynamic behaviour of the system along a computation: the sequence of labels corresponds to the (ordered) set of reactions that took place, whereas the sequence of states follows at each step the change in metabolite concentrations. When no reaction can take place due to the lack of the needed metabolites, a final state is reached (in figure shown as a thick circled node).

Once the model is defined, we must specify the properties we want to check. What are the biologically relevant properties that may be interesting to check or deduce from the model? Referring to the classification that we presented in section 4.2 we may be interested in the following qualitative properties:

- *reachability*: does the model produces Oxalate? In case, what is the maximum Oxalate production and what are the values of the other metabolites?
- *temporal ordering*: what is the ordered sequence of reactions (if any) that corresponds to the maximum produced Oxalate?
- *monotonous trend*: what are the metabolites that are only produced by the network? And what are the ones that are only consumed?

Furthermore, we may imagine to remove reaction $R2$ from the network, obtaining a so-called Knock-Out (KO) model, and then compare the results for the two networks. We can ask for instance, if the maximum Oxalate is the same or what are the metabolites that change the most, and which is the sign of this variation.

As will shown in section 5.4 all these information may be extracted using properties expressed as simple invariants.

5.2 Model Verification and Behaviour Filtering

As outlined in the previous section, there are several interesting properties that can be defined for a Genome-scale Metabolic Network model. However, an important topic should be immediately addressed, that is concerned with working on biological models.

First of all, is often important to assess if a biological network is able to **satisfy a property**. This is in contrast with the standard automaton-theoretic approach that we have described in subsection 3.2.1, in the sense that we do not want to ascertain that the model is free from errors, rather we want to prove that the model is able to **express a required property**. In this sense, we are following the concept (widely used in the context of SAT-based or BDD model checking [91, 145]) of generating a *witness* for a property that has been proved to hold in the system. Indeed, if the model is able to satisfy the required property then an example of a successful computation will be generated.

The use of a simple trick makes this approach immediately amenable also to the automaton-theoretic methodology that we have described: since in its standard use the Non–deterministic Büchi Automaton (NBA) is specified for the negation of the property that should hold (see figure 3.2), to extract a witness it suffice to negate the property of interest. In fact, the first negation will be annealed by the innermost negation applied when building the NBA. In other words, if Φ is the property for which we require a

witness, we will submit to the model checker the property $\Psi = \neg\Phi$. The checker will build the automaton for $\neg\Psi = \neg\neg\Phi = \Phi$. In practice, the extracted counterexample (if any) will be a witness of the ability of the system to satisfy the property Φ.

The second issue is more delicate. Indeed, for many of the biological properties above described a single witness is not enough: due to their intrinsic robustness (see chapter 2), biological systems are usually able to accomplish a function using different paths in the network. It would be of great interest being able **to extract a whole set of witnesses** describing a complete **behaviour of the network**. This is not a problem, since the Spin model checker permits to generate all the counterexamples that it is able to find. Indeed, we are using the model checker to *query the behaviour of the model* [146]. Since we are extracting a set of interesting computations from the whole set of the possible ones, we refer to this operation as **behaviour filtering**. The information gained from the filtered behaviours may help both in improving the understanding of the model both in inferring new properties, thus remaining in the context of the hypothesis-driven approach (see chapter 2) that is a distinctive feature of the Systems Biology paradigm.

However, this approach, when applied to large models, may cause some problems: filtering out the whole set of counterexamples that represent a behaviour may be practically infeasible. This is due both to the computational issues related to the state-space explosion problem (see section 3.3) both to the intrinsic difficulty in elaborating a very large number of computations. The state-space explosion issue is common also to the standard Model Checking approach. Apart from being tackled with dedicated algorithms, this problem may be circumvented moving from an exhaustive to a **monitoring approach** [9]. That is, instead of perlustrating the whole computation space a sort of "*sampling*" is performed that extracts only a subset of the computations of interest. This is a wide used approach to deal with large models, and sometimes the only feasible one to have some information on their correctness.

Of course the application of sampling-like procedure raises the problem of the representativity of the extracted computations, but some considerations should be done. First, we are using Model Checking in a different perspective, i.e. as a tool to extract qualitative information from a model that is representing an already-existing system, i.e. the biological one (see chapter 4). Hence the extracted information, although not exhaustive, may be of great importance because: i) they still permit to describe the dynamic behaviour of the system and ii) they may be compared with the existing knowledge either confirming it or, if different, suggesting new hypotheses about the system (and hence the model) behaviour.

Moreover, both the sampling problem and the issue of dealing with a large number of counterexamples may be mitigated using the swarm tool proposed by Holzmann et al. in [122] to tackle the problem of verifying a model too large to be exhaustively analysed. As will be explained in detail in chapter 6 it is possible to tailor the perlustration of the computation space by sampling at different depths, and in different portions of the computation tree. Furthermore, it is possible to specify the maximum number of extracted counterexamples in each sampling zone. Indeed, a well tuned strategy makes it possible to extract a great amount of very interesting information from model on which any other exhaustive approach would have been impossible.

5.3 From a biological model to an executable model

As already described in chapters 2 and 4, a metabolic network model can be considered as a dynamic system based on the interplay between the contained reactions, which consume/produce metabolites in a mainly concurrent fashion. The relative amounts of the produced/consumed metabolites depend on the reactions specific stoichiometry. The system keeps changing until there are reactions that can take place i.e. until there are reactants that can be transformed in products by some reactions. Indeed, the reactions

activity is limited only from the availability of the stoichiometric quantity of the involved reactants.

The approach that we propose captures all these biological relevant features, and is based on considering each chemical reaction as a **process that tries to access concurrently to the shared resources represented by the metabolite concentrations**. Intuitively, when a reaction occurs the reactant concentrations should diminish whereas the product ones must increase, both according to the reaction stoichiometry. Furthermore, we require that the reactions should take place until there are reactants to use. This perspective is described in detail in the following sections.

5.3.1 Metabolites

Each chemical compound is represented by a numeric (global) variable, storing the metabolite concentration (in a generic unit measure). Metabolites that can move between cellular compartments or can be exchanged with the environment (i.e. involved in transport reaction) will have a distinct variable for each of the compartment in which they appear. Metabolite concentrations may assume only integer values, since we are dealing with integer multiples of the stoichiometric coefficients (that are defined as integers).

5.3.2 Reactions

As described in chapter 2, a metabolic network may contain two kinds of reaction: **transport reactions** involve the transfer of (usually) a single chemical species to a different compartment, or its exchange with the external environment. The usual chemical reactions are called **internal reactions** because they involve only compounds that belong to the same compartment. Each reaction type should be modeled in a different way:

- *transport reactions*: each reaction is modeled as a process that tests if the metabolite variable corresponding to the source compartment is greater than zero and, in this case, will decrement of one unit its value and increment of one unit the metabolite variable of the destination compartment. To ensure that this sequence of operations is executed without any interruption, it must be implemented as an *atomic* instruction (see section 3.1.1).
- *internal reactions*: each reaction is a process that tests the value of the reactants variables and, if all of them are available in the quantity defined by the reaction stoichiometry, updates the values of the reactant and product variables. Also in this case, an atomic clause should be used to ensure the exclusive access of the process to the needed variables.

5.3.3 Translating the biological model to Promela

With the ideas of section 5.3 in mind, we designed first a naïve translation that captures in a direct and smooth way the essential features of the model. In the following paragraph, we will address some drawback of this solution, and give details on the improved, final implementation that was developed and effectively used on two biological models (see chapter 7).

The Promela translation has a *proctype* declared for each reaction and a *do* statement to model its repeatability. The test-and-set operation on the variables is obtained with an *atomic* clause surrounding the guard and the variable update instructions. An example, shown in the Promela code snippet 1, involves the reactions of the LTS of figure 5.1.

```
/* THE METABOLITE VARIABLES */
 byte GC_ext = 2;
 byte GC = 0;
 byte GO = 2;
 byte AL= 2;
 byte GL=2;
 byte PY=2;
 byte OX=2;

/* INTERNAL REACTION N. 1: GC -> GO */
   proctype R1(){
     do :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1;}
     od
    }

/* INTERNAL REACTION N. 2:  GO + AL  →  GL + PY */
   proctype R2() {
     do :: atomic{((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1}
     od
   }

/* INTERNAL REACTION N. 3: GL -> GO */
   proctype R3(){
     do :: atomic{(GL > 0)  →  GL=GL-1; GO=GO+1;}
     od
    }

/* INTERNAL REACTION N. 4: GO -> OX */
   proctype R4(){
     do :: atomic{(GO > 0)  →  GO=GO-1; OX=OX+1;}
     od
    }

/* TRANSPORT REACTION N. 1: GC_ext -> GC*/
   proctype TRN_R1(){
     do :: atomic{(GC_ext > 0)  →  GC_ext=GC_ext-1; GC=GC+1;}
     od
    }

/* START THE PROCESSES  */
      init { atomic{run R1(); run R2(); run R3(); run R4();
                                          run TRN_R1(); } }
```

Promela code snippet 1: The translation of a metabolic network made of four internal and one transport reactions. Each reaction is mapped on a distinct process; an additional process is used for the timeout invariant property specification.

5.3.4 Model abstraction: reducing the complexity

As clearly pointed out in the Spin Manual [12], a poorly designed model may have a deep impact on the computational complexity. Unwanted processes or variables may deeply affect the size of the state space, increasing the time and memory required for its exploration. On the contrary, abstracting the model and keeping it as simple and small as possible (building the "smallest sufficient model", as called by Holzmann) has a deep impact on the feasibility of the verification/exploration steps. The Promela implementation of code snippet 1 relies on defining a process for every reaction. But this choice may have sense only for very small models, for two reasons: first, Spin has a limitation on the max number of processes that may be run at the same time (i.e. 255, see [12]). This may result in a severe constraint on the number of representable reactions, and this is not acceptable since metabolic network models typically include a number of reactions larger than 255. Second, and most important, the presence of a large number of processes results in an unwanted increment of the computational burden, condition that is never desirable when working with large models.

Hence, we decided to adopt another approach, building a new, more abstract, model that relies on the definition of a single process containing a single *do* statement, with each branch corresponding to a distinct reaction. The reaction itself is modeled as before, i.e. as an atomic test-and-set of the variables. The main difference with the previous model is that reactions are not modeled as single processes, but as single instructions, each

one competing for (not-deterministic) execution inside the *do* statement (see subsection 3.1.1). The same principle also holds when modeling the network compartments, if any. The first idea was to design each compartment as a distinct process. However, this choice, although reflecting the original model structure, introduces only additional complexity while not improving the functionalities. Hence, it was rapidly abandoned.

To reduce complexity, a second relevant issue must be addressed: the choice of the initial value of the metabolite variables. The model itself is finite, and it will evolve until there are metabolites to transform. But the number and type of possible executions clearly depends on the available metabolites: indeed, higher the initial values higher the number of possible different states to be explored. Since a genome scale metabolic network usually contains a large number of metabolites, often in different compartments, the number of variable to define may be very high (see chapter 2). Keeping the number of different states as low as possible, while guaranteeing the full functionality of the model, is mandatory. The approach we decided to use is based on a fundamental assumption: we are mainly interested in exploring the behaviour of the system in terms of the executed reactions and of the relative metabolite consumption/production rather than reasoning on the values themselves. The initial values are then chosen taking into account the minimal amount that is needed to make each reaction of the model executable at least once. This minimal amount is simply the maximum (w.r.t. all the reaction) stoichiometric coefficient of each metabolite. This is a good choice because: *i)* the reaction stoichiometry usually involves only small integer numbers; *ii)* it assures that in principle every reaction can be used at least once.

Another way to reduce the model size takes into account the presence of transport reactions that move metabolite from or to the outside environment, although it requires some care and an *a priori* knowledge of both the model at hand and the properties that will be defined. Indeed, if the external metabolites are not included in the properties to check, then we are not interested in modeling the transport to/from the environment. Since a Promela model is a finite and closed one (i.e. all the sources of input and all behaviours are completely specified [12]), the net effect of the transport reactions is only to sum/subtract the total values of the involved external metabolites to/from the corresponding internal variables. Hence, unless we need to specify conditions on the external metabolites, the corresponding transport reaction may be removed (together with the external metabolites) thus lowering the overall complexity of the model. The initial value of metabolites that are transferred from the external environment into the system, might be directly assigned as the initial value of the internal corresponding variable (as in the Promela code snippet 2).

The last topic concerns the definition of properties, in particular of the state invariants. Indeed, for small models is easy to define a monitoring process containing an *assert* clause. Due to the inner non–deterministic choice of the instructions and to the exhaustive approach, the monitoring process will be interleaved in all the possible combination with the main process. However, this choice can in principle double the state space (see page 386 of the Spin Manual [12]). This is not an issue in small models, but may be relevant for larger ones. Hence, a clever implementation of the property checking code may significantly change the overall verification process, as described in Section 5.4.6.

Concluding, the modeling strategy here described is biologically plausible, and captures the key properties of the metabolic networks structure and dynamics:

1. the production/consumption of the network metabolites is represented by the increment/decrement of the corresponding variables;
2. the reaction stoichiometry is guaranteed by the exclusive access to the metabolite variables;
3. the reaction dependence is modeled by the presence/absence of the needed metabolites;
4. the concurrency of reactions is guaranteed by non-determinism.

```
/* THE METABOLITE VARIABLES */
 byte GC = 2;
 byte GO = 2;
 byte AL= 2;
 byte GL=2;
 byte PY=2;
 byte OX=2;


/* INTERNAL REACTION N. 1: GC -> GO */
/* INTERNAL REACTION N. 2:  GO + AL  →  GL + PY */
/* INTERNAL REACTION N. 3: GL -> GO */
/* INTERNAL REACTION N. 4: GO -> OX */

   proctype REACT_process() {
     do
        :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1;}
        :: atomic{((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1 }
:: atomic{(GL > 0)  →  GL=GL-1; GO=GO+1;}
:: atomic{(GO > 0)  →  GO=GO-1; OX=OX+1;}
     od
   }

/* START THE PROCESS  */
      init{run REACT_process()}
```

Promela code snippet 2: The same example metabolic network of snippet 1, but translated using a single do-statement rather than multiple processes

In the Promela code snippet 2 the same internal reactions contained in the example code of the snippet 1 are shown, but modeled this time as a single process. The transport reaction moving GC_{ext} from the outside environment and the GC_{ext} variable itself have been removed, whereas the value of GC_{ext} was directly assigned to the GC variable.

Please note that the techniques here described have been used for building the executable models used in this thesis work (chapter 7).

5.3.5 The importance of a Flux Balance Analysis step

We have implicitly assumed right now that all the reactions in the network have a unique, defined direction. However, this is not always the case: first, in large-scale models, information on the Gibbs free energy associated to a reaction (and hence to its direction, see subsection 2.3.3) is often missing. Second, some reactions are inherently bi-directional. Of course, bidirectional reactions may add a great load of unwanted computational burden, since each of them may result in a cycle in the corresponding LTS. This is a very delicate issue, that must be carefully addressed.

To this aim, we have devised a novel approach that exploit the powerful Flux Balance Analysis (FBA) method to decide the directions of potentially bi-directional reactions. As explained in subsection 2.3.4, the Flux Balance Analysis approach used in [72] determines the reaction directions in a way that has a strong biological meaning. Indeed, the minimization process is based on the assumption that the modeled organism should accomplish its biological functions with the minimum waste of energy, and the steady-state flux distribution is calculate for a precise metabolic object. When more than a metabolic function exists, the mean of the distributions of the fluxes across the l metabolic objectives may be considered, as already described in subsection 2.3.4.

Thus, the direction of each reaction may be decided taking into account the sign of the corresponding flux, i.e. for each reaction r_i, with associated average flux $\bar{v}_i$:

- if $\bar{v}_i > 0$, the reaction will have its normal direction (i.e. the one which is contained in the stoichiometric matrix).
- if $\bar{v}_i < 0$, the direction will be the inverse one, i.e. products and reactants will be exchanged (i.e. the $i-th$ row of the stoichiometric matrix will be multiplied for -1).
- if $\bar{v}_i = 0$, different strategies may be considered. A zero-flux indicates that the corresponding reaction is never used, hence it is possible to remove it from the network.

Otherwise, the choice may be guided by some biological knowledge. Lastly, it is possible to decide randomly its direction.

We call *network pruning* this combined process of direction decision and reaction removal. It is worth noting that the pruning step is of capital importance when dealing with large models: in fact *i)* pruning permits to avoid the cycles in the LTS induced by the presence in the network of bi-directional reactions, and ii) it has the secondary but fundamental effect of reducing the model size in a biologically consistent manner. Using this novel, combined approach of FBA and model checking also very large metabolic network models are amenable to qualitative analyses.

In this thesis work, we decided to perform a network pruning in which the unused reactions were removed. This choice was made for all the tests we performed on real problems (chapter 7).

5.4 From biological features to formal properties

Once introduced the model specification, we have to define the formal properties (i.e. the behaviours) we want to check or extract. As already outlined in the example of section 5.1, several interesting questions about metabolic network models may be asked. We will now describe in depth the process of translation of the biological properties into the formal tools we have introduced in chapter 3. It is worth noting that a wide range of interesting biological features may be expressed using the Promela assert instructions (i.e. state invariant specifications) that can be verified with a simple reachability analysis. Only in one case a LTL formula, although very simple, might be used to speed the verification process (see subsection 5.4.6).

We will follow the property classification proposed in section 4.2 when describing the properties that we implement in our framework. Please do note that in this classification the set of the *reachability* properties refers to the ones expressing the possibility for the network to reach a fixed concentrations of a species. This does not necessarily correspond to the concept of reachability of a state into the associated LTS. To avoid confusion we call the former *Concentration reachability properties*, since they are mainly concerned with the ability of the network to produce particular metabolite concentrations (such as the minimum or the maximum one). Before detailing the set of the expressible properties and the detail of their implementation in Promela or LTL, we introduce an important instruction, namely the *timeout*, that permits to defer the verification of an assert when a deadlock state is reached.

5.4.1 The *timeout* clause

Once an invariant property is specified (using the *assert* instruction) and the verification is started, the model checker Spin will verify whether the specified invariant holds in each of the states. That means that a counterexample is generated as soon as a state not fulfilling the specified property is found. But in the metabolic network context, it is often more interesting to investigate the concentration values obtained when the final, "steady-state" condition of the network is reached after the initial perturbation caused by the input concentration (i.e. the initial values) of the metabolites. To this aim, the invariant check can be performed when a **deadlock state is reached by the model**. A deadlock state corresponds to a situation in which all the reactions that may occur have already took place, and nothing more can happen due to the lack of the needed metabolites.

This idea is implemented using the Promela *timeout* instruction. This instruction becomes executable only when no other instruction in the whole Promela program is executable, i.e. when a deadlock state is reached. Hence it is possible to trigger the invariant check only when the deadlock condition is reached. An example of timeout

```
/* THE METABOLITE VARIABLES */
 byte GC = 2;
 byte GO = 2;
 byte AL= 2;
 byte GL=2;
 byte PY=2;
 byte OX=2;


/* INTERNAL REACTION N. 1: GC  →  GO */
/* INTERNAL REACTION N. 2:  GO + AL  →  GL + PY */
/* INTERNAL REACTION N. 3: GL  →  GO */
/* INTERNAL REACTION N. 4: GO  →  OX */

   proctype REACT_process() {
     do
        :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1;}
        :: atomic{((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1 }
        :: atomic{(GL > 0)  →  GL=GL-1; GO=GO+1;}
:: atomic{(GO > 0)  →  GO=GO-1; OX=OX+1;}
:: timeout  →   assert(OX <= 2);
     od
   }

/* START THE PROCESS  */
      init{run REACT_process()}
```

Promela code snippet 3: The same code of snippet 2, here including an *assert* clause in combination with a *timeout* instruction. The *timeout* instruction is used to trigger the *assert* only when all the other instructions are blocked.

invariant specification is given in the Promela code snippet 3. Its application to the LTS of figure 5.1 would result in the extraction, from the computation space, of only those states in which: *i)* the oxalate is above the initial value and *ii)* no reaction can take place (the bold bordered state of figure 5.1 is one of them). Without the timeout clause, instead, the filtering process would generate a counterexample for each of the intermediate states in which the oxalate concentration increases (indicated in Fig. 5.1 with the OX state variable written in bold). Of course, the timeout (deadlock) condition forces the metabolic network to consume all the available metabolites. This may correspond, in a biological context, to the waste of some energies in the so called *futile cycles*. However, this effect is only marginal, and can be easily deduced from the computation analysis.

5.4.2 On limits and strengths of an approximated verification strategy

As already discussed in section 5.2, working with genome-scale models may be very challenging, due to their size and intricacy, and approximated strategies are sometimes the only possible approach to have useful insights on their dynamics.

When an approximate strategy is used, two different issues must be took into account, i.e. the representativity of the extracted behaviours and, at the opposite, the meaning of a verification step that does not find a counterexample.

For what concerns the sampling process, by no means it is possible to precisely know how many states might be generated for a model. This is essentially due to the techniques used for the verification process, i.e. the on-the-fly generation of the LTS, the partial order reduction and the nested search algorithms [12]. Hence, it is impossible to quantify the explored portion of the space and assess the representativity of a set of counterexamples. However, tests on small and *ad hoc* built models [122] have shown that the approximated *swarm* approach supported by Spin performs very well in the verification context. Since we are interested in finding more than one counterexample, an intelligent extraction strategy should be carefully devised as will be described in chapter 6. With some precautions, a filtering procedure may be finely tuned to gather a large number of interesting counterexamples within the limits of a reasonable trade-off between the computational expense and the obtained results.

On the other hand, when an exhaustive search ends without finding counterexamples we are assured that the specified property holds in the model. If an approximate strategy is used, it is not possible to draw a sharp conclusion. In the software and hardware verification context, it would be of course a big issue, since nothing could be stated about the correctness of the model. However, in the biological context we have to take into account other conditions. In fact, as described in chapter 2, biological systems are robust. But a system would not be so robust if relying on a combination of events that has a very low chance of occurring. Hence, if a counterexample is not found at the end of an approximated search, it is very unlikely that an exhaustive one will find it.

As introduced in chapter 4, the verification of biological models requires a shift of perspective from the traditional model checking view. Indeed, in the hardware and software verification context, the lack of an exhaustive answer can invalidate the whole search. Instead, in the biological context even an approximated result may be of great interest. This inversion is rooted in the crucial difference between the biological system and the hardware or software ones: the latter may contain design errors that should be exhaustively detected, whereas a biological system already exists and is correct by default. The biological model is built to verify that the hypotheses made on its functioning are able to catch some features of the underlying, partially unknown, mechanisms. Hence also partial, qualitative information may be essential in elucidating the possible dynamics of the modeled system.

In conclusion, we are trading a sure - but impossible to obtain - answer with a less accurate, but available one.

5.4.3 Concentration reachability properties

We will describe now the properties that may be used to extract interesting information about the metabolite concentrations. All of them are specified using the invariant construct provided by the Promela language, i.e. the *assert* instruction. Apart from knowing if the network is able to produce a metabolite, is usually very important to have the details of how this target is reached. Hence, the list of the performed reactions, as well as the final concentration of the other metabolites should be extracted: luckily, these information are very easily obtained from the counterexample files that Spin generates (see chapter 6).

Production or consumption of a given metabolite

The simplest property that may be defined involves the ability of a network to produce/-consume a metabolite. This is readily obtained specifying as invariant the negation of the desired behaviour (see section 5.2): in fact, if we want to know if a given metabolite is produced, we should specify as an invariant that the value of the metabolite concentration should be always not greater than the initial value. That is:

Property Definition 1 *Production of a metabolite*
Let M_0 be the initial value of the concentration of a metabolite M. We want to check the property:

$$\Phi = \Diamond(M > M_0).$$

The property passed to the checker is:

$$\Psi = \neg\Phi = \neg(\Diamond(M > M_0)) = \Box(M \leq M_0).$$

The extracted counterexample (if any) will be a witness of the ability of the system to satisfy the property Φ, i.e. to produce an amount of metabolite M greater than the initial value M_0.

A similar property may defined for the consumption of a metabolite: we require that the value should be always not lower than the initial value. That is:

Property Definition 2 *Consumption of a metabolite*
Let M_0 be the initial value of the concentration of a metabolite M. We want to check the property:

$$\Phi = \Diamond(M < M_0).$$

The property passed to the checker is:

$$\Psi = \neg\Phi = \neg(\Diamond(M < M_0)) = \Box(M \geq M_0).$$

The production/consumption properties can also be used with a timeout clause. However, this corresponds to a more stringent request on the network behaviour, since it is stating that the *final* value of the metabolite should be greater (for the production) than the initial one. Of course the satisfiability of the property in an intermediate state (i.e. without the timeout clause) does not imply the satisfiability of the property in the final state, since a metabolite that can be both produced and consumed by the network does not necessarily accumulate in the final states. The same holds for the consumption of a metabolite.

This strategy may of course be used both to extract a single counterexample, i.e. a witness of the ability of the network to express the required property, or to filter out a behaviour, i.e. a set of counterexamples describing in more detail how the network is able to fulfill it.

Maximum or minimum concentration of a given metabolite

A more interesting feature is the ability of a network to reach a maximum or minimum concentration of a metabolite. We will describe the first the strategy to obtain maxima. Since an *a priori* knowledge of a possible maximum value is not available, we can not directly use the approach described in Definition 1. Rather we implement an iterative extraction strategy (similar to the one described in [9]). At first, a behaviour filtering step is performed as described in Definition 1. The metabolite maximum value is extracted from the obtained counterexamples. This value is used to start another extraction step, with the same strategy, and the whole process is performed until no counterexamples are found.

Property Definition 3 *Maximum of a metabolite*
Let M_0 be the initial concentration value of the metabolite M, and M_{i-1} the maximum reachable concentration of M obtained at the $(i-1)$-th step. At the i-th step the required property is:

$$\Box(M \leq M_{i-1})$$

If a counterexample is found, then extract the maximum concentration value M_i and start again the process.
Else the maximum value is given by

$$M_{max} = M_{i-1}.$$

Extracting more than a counterexample at each step obviously speeds up the whole process. Since the initial values of the model are finite and the model itself is finite, a maximum will be obtained in a finite number of steps. Moreover, in the biological context it is often not so relevant to get the actual absolute maximum, but also a near-to value will be of the same interest.

The maximum search may be also performed using the timeout clause, although it should be noticed that this will represent a more stringent request on the network behaviour. When searching for a maximum concentration of a metabolite at timeout (i.e. at the final state reached from the network) an a priori knowledge of the metabolite behaviour may be useful. Indeed, for metabolites that can be only accumulated by the network, the maximum search strategy above described may be directly applied, and the

timeout value is guaranteed to be an absolute maximum. On the contrary, if a metabolite is both produced and consumed by the network (i.e. its dynamics is not monotonically increasing) the timeout search may result in a local maximum value, since the timeout favours metabolite consumption (when possible). Hence, if the metabolite is both produced and consumed by the network, or if an a priori knowledge of its behaviour is not available, a two-step approach may be used: the first with the timeout, to establish the (maybe local) maximum and a subsequent one without the timeout either to confirm it, or to get the absolute one.

The minimum search is based on the same techniques above detailed. Hence, a generic extraction step is described as follows:

Property Definition 4 *Minimum of a metabolite*
Let M_0 be the initial concentration value of the metabolite M, and M_{i-1} the minimum reachable concentration of M obtained at the $(i-1)$-th step. At the i-th step the required property is:

$$\Box(M \geq M_{i-1})$$

If a counterexample is found, then extract the minimum concentration value M_i and start again the process.
Else the minimum value is given by

$$M_{min} = M_{i-1}.$$

Also in this case, the whole process may benefit from a behaviour filtering approach to speed up the search and lower the number of required iterations. In contrast to the maximum search, using the timeout clause may sometimes be mandatory: in fact for metabolites that are only produced by the network a simple minimum search will find only the initial states (i.e. the initial values), that are not really informative. Instead, with a timeout is possible to find the more interesting, final-state minima.

Satisfaction of a Metabolic Objective

Another set of interesting behaviours involves the ability of the network to simultaneously satisfy a set of concentration constraints, i.e. the so-called Metabolic Objectives. As said in section 2.3.4, they represent metabolic functions that the network should be able to perform. A Metabolic Objective is usually expressed as a constraint on the simultaneous production and/or consumption of a set of metabolites. The corresponding property may be easily obtained combining the conditions on the production/consumption of the single metabolites (expressed as described in the Property Definition 1 and 2) with the boolean logic operator $\wedge$ (*and*). We describe hereafter this approach (similar to the one used in Chabrier et al. [124]).

Property Definition 5 *Satisfaction of a Metabolic Objective*
Let m be the total number of metabolites in the network, M_{j_0} the initial concentration of the metabolite j, and M_j its generic concentration value, with $\{1,..,j,..,l\} \subseteq \{1,..,m\}$ is the subset of the involved metabolites. The property for the j-th metabolite is defined either as $p_j = \Diamond(M_j < M_{j_0})$ or $p_j = \Diamond(M_j > M_{j_0})$
We want to check the property

$$\phi_{MO} = (p_1 \wedge ... \wedge p_j \wedge ... \wedge p_l)$$

The property passed to the checker is:

$$\Psi_{MO} = \neg\Phi_{MO} = \neg(p_1 \wedge ... \wedge p_j \wedge ... \wedge p_l).$$

That is

$$((\neg p_1) \vee ... \vee (\neg p_j) \vee ... \vee (\neg p_l))$$

and

$$\neg p_j = \begin{cases} \Box(M_j \geq M_{j_0}), & \text{if } p_j = \Diamond(M_j < M_{j_0}) \\ \Box(M_j \leq M_{j_0}), & \text{if } p_j = \Diamond(M_j > M_{j_0}) \end{cases} \quad \forall j \in \{1, .., l\}$$

Also for the validation of a Metabolic Objective it is possible to extract a single counterexample, i.e. a witness of the ability of the network to fulfill the required property, or to filter out a whole behaviour describing the different ways in which the property is satisfied. The timeout clause, as usual, expresses a tighter constraint on the network behaviour, since it requires the network to satisfy the metabolic objective in the final states.

5.4.4 Monotonous trend properties

Another interesting feature of a network is the ability to monotonically consume or produce a metabolite. This is of course a more stringent constraint than the simple production/consumption above described. Since the presence into the network of reactions that both consume and produce a metabolite does not imply the ability of the network to eventually use them, a formal verification might me be used. This property may be easily implemented introducing a boolean variable, that is used to flag the inversion in trend of a metabolite value. The instruction that updates the flag value is placed inside the atomic clause surrounding each reaction changing the metabolite trend in the inverse direction with respect to the desired one. If we are interested in verifying the monotonically increasing (decreasing) production of a metabolite, we will place the flag update instruction only in the reactions that consume (produce) the metabolite.

```
/* THE METABOLITE VARIABLES */
....

/* THE FLAG VARIABLE */
bool flag_monotonic = true;

/* THE PROPERTY TO CHECK */
#define GO_monotonic (flag_monotonic==true)

   proctype REACT_process() {
     do
        :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1;}
         /* THE REACTION DECREASING THE GO METABOLITE CHANGES THE FLAG VALUE */
        :: atomic{((GO > 0) && (AL > 0)) → GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1; flag_monotonic=false;}
        :: atomic{(GL > 0)  →  GL=GL-1; GO=GO+1;}
        :: atomic{(GO > 0)  →  GO=GO-1; OX=OX+1;}
     od
   }

/* START THE PROCESS  */
      init{run REACT_process()}

/* THE LTL FORMULA */
ltl Monotony_check { always GO_monotonic}
```

Promela code snippet 4: The implementation of the increasing monotonicity check for the GO metabolite. The assert is expressed as an LTL *always* formula.

This property can be implemented as a simple assert stating that the flag value should remain unchanged in all the reachable states. However, if we are not interested in the generated counterexample, we may use a more efficient approach (as explained in section 5.4.6) and use an *always* LTL formula, as shown in Promela code snippet 4. The monotonicity check property may be defined as follows:

Property Definition 6 *Monotonicity*
Let flag_monotonic be a boolean variable, placed in the atomic clause surrounding the

reactions that decrease (respectively: increase) the concentration of a metabolite M. The check for increasing (decreasing) monotonicity is given by the property:

$$\Phi = \Box(flag\_monotonic == true).$$

It is worth noting that in this case we are using the checker in a traditional way, i.e. simply specifying the property that we want to be verified by the model. Lastly, in the monotonicity check the timeout clause has no sense, and its use in this context has not been further investigated.

5.4.5 Temporal ordering properties

Linear Time Logic does not account for an explicit description of time. Hence it is not possible to express properties that directly address the relative timing of the events. However, it is still possible to extract relevant biological information on the temporal behaviour of the system by extracting the ordered sequences of reactions from the counterexamples. Assuming that each reaction takes places into a generic unit of time, the sequences of the performed instructions may give information on the relative ordering that the reactions should have to reach a specific network condition (see subsection 6.1.5). Furthermore, as described in section 5.5 it is possible to define some tools that constrain the ordering of a subset of transitions while executing the model.

5.4.6 Reducing complexity with clever choices

As already introduced in subsection 5.3.4 a great effort must be spent in the model abstraction process, since a good model can drastically reduce the effort needed for verification. Among the different aspects that should be considered, a clever choice of how coding the property to check may be essential. We use essentially safety properties that can be expressed as state invariants. However, the different ways in which invariants are declared may strongly affect the overall computation. We take into account three different strategies to express the same property, and we well briefly review their strengths and limits:

1. *a monitoring process.* This is the simplest way to check an invariant: an assert is placed in a process that is run in parallel with the system one (see Promela code snippet 5). Due to the inner non–deterministic choice of the instructions and to the exhaustive approach, the monitoring process will be interleaved in all the possible combinations with the main process. However, this choice can in principle double the state space (see the Spin Manual [12]). This is not an issue in small models, but may be relevant for larger ones. Moreover, when an approximated search is used, the explored paths may not contain the monitoring process, with two net effects: *i)* potentially more time is needed for the verification, since the assert will not be triggered until the monitoring process is executed; *ii)* it may become difficult to ascertain if an empty search is due to the property effectively holding or to an unlucky choice of the process interleaving. The only strength of this approach, in our context, relies in the easiness of its use, since there is no need to put the assert instruction in a special place.
2. *an inline assert.* Here we may distinguish two cases. If the invariant verification is performed at timeout, then it can be placed in the *do* loop, as shown in the Promela code snippet 2. Indeed, the timeout clause ensures that the assert will be triggered only when no other reaction in the do cycle is executable. Otherwise, the assert can be placed into the atomic clauses of the reactions of interest, i.e. the reactions that involve the metabolites contained in the assert itself, as shown in 6. Although this approach requires the knowledge of the right place for the assert, it is the safest one: in fact, the atomic clause ensures that the assert is triggered as soon as the reaction has took place. Since the assert refers to the metabolites involved in the reaction, we are sure that we are checking their value at the right moment.

3. *a LTL formula.* A third possible approach is shown in 7. Here, instead of using an assert clause, we specify an LTL formula stating that the property of interest should be always true. To this aim, we use the *always* operator defined as in subsection 3.1.2. The LTL formula is translated into a Promela *never* claim, i.e. an automaton that is executed in synchronous product with the model LTS (as explained in subsection 3.2.1). In this case the property is checked before and after the execution of each instruction. A *never* claim constrains the generation of the state space to only those executions that may violate the property (see [12]). Hence it may detect more rapidly a property violation, but does not generate all the witnesses that are generated with the other strategies above described.

Hence, we rely on the inline assert strategy whenever we want to extract a witness or a whole behaviour, since we are interested in the generated counterexamples. Instead, we use the LTL formula strategy when we want only to verify if a property holds, but we do not care about counterexamples (as for the case of the monotony check see section 5.4.4). Please do note that when using the inline assert strategy without the timeout, only a subset of all the possible counterexamples is generated: it contains the executions in which a combination of the instruction *without* the inline assert eventually leads to the instruction *with* the assert. To obtain a larger set of behaviours including, for instance, more than one use of the instruction with the assert interleaved by various combinations of the other instruction without the assert, another strategy may be used: in this case the assert should be specified inline also *in all other instructions*, event though they do not alter the variable monitored by the assert.

```
/* THE METABOLITE VARIABLES */
 ....
 byte OX=2;
 ....

   proctype REACT_process() {
     do
        :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1;}
        :: atomic{((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1; }
        :: atomic{(GL > 0)  →  GL=GL-1; GO=GO+1;}
        :: atomic{(GO > 0)  →  GO=GO-1; OX=OX+1;}
     od
   }

proctype MONITOR_process() {
     assert(OX <= 2);
   }

/* START THE PROCESSES  */
      init{atomic { run REACT_process(); run MONITOR_process } }
```

Promela code snippet 5: Invariant checking strategy n. 1. The assert is placed into a monitoring process, that is interleaved with the one containing the reactions.

5.5 Control tools: working on transitions

In this section we describe the technique we have designed to constrain the computation space exploration to only some interesting portions. These strategies are based on controlling the transitions that the checker will generate and explore during verification. These tools may be very useful, for different reasons: *i)* they permit to generate models that are more similar to their biological counterpart, and *ii)* they allow to perturb the system, simulating conditions that are different from the normal ones. The model checker we decided to use in this thesis work, e.g. Spin, and the formal techniques on which it is based (e.g. the automaton-theoretic approach described in subsection 3.2.1) do not

```
/* THE METABOLITE VARIABLES */
  ....
 byte OX=2;
 ...

   proctype REACT_process() {
     do
        :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1;}
        :: atomic{((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1; }
        :: atomic{(GL > 0)  →  GL=GL-1; GO=GO+1;}
        :: atomic{(GO > 0)  →  GO=GO-1; OX=OX+1;  assert(OX <= 2);}
     od
   }

/* START THE PROCESS  */
      init{run REACT_process()}
```

Promela code snippet 6: Invariant checking strategy n. 2. The assert on the OX concentration is placed inline, in the atomic clause surrounding the reaction involving the OX metabolite

```
/* THE METABOLITE VARIABLES */
 ....
 byte OX=2;
 ....

/* THE PROPERTY TO CHECK */
#define OX_increase ((OX <= 2))

   proctype REACT_process() {
     do
        :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1;}
        :: atomic{((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1;}
        :: atomic{(GL > 0)  →  GL=GL-1; GO=GO+1;}
        :: atomic{(GO > 0)  →  GO=GO-1; OX=OX+1;}
     od
   }
/* START THE PROCESS  */
      init{run REACT_process()}

/* THE LTL DECLARATION */
ltl Invariant_check { always OX_increase}
```

Promela code snippet 7: Invariant checking strategy n. 3. The property involving the OX concentration is defined as a constant with a preprocessor directive. The *always* LTL formula, named Invariant_check, is specified at the end of the Promela program.

permit to directly express properties on the transitions between states. That means, for instance, that is not possible to remove from the state space all those computations that not involve reactions in which we are interested. However some "tricks" may be used to force the executions along interesting paths: we implemented two approaches, one based on the definition of a synchronized control process running in parallel with the principal one, and a second that is based on defining subsets of reactions with different priorities. Of course, both these techniques may be used together with the property verification and behaviour extraction strategies that have been previously described.

5.5.1 Synchronized processes

Sometimes may be crucial that a specific reaction (or a set of reactions) should be executed as soon as possible. This may be due to some biological knowledge stating, for instance, that a reaction is catalyzed by a very strong enzyme and therefore should take place as soon as all the needed metabolites are present. Since we have no direct control on the choice of the next transition that the model checker will explore, we decided to partially circumvent this problem coupling the execution of the main process with another one running synchronously with it. This "control" process will direct the execution of the non–deterministic one, checking at each step if the reaction of interest may

take place. If this is the case, the reaction is performed, and after the control returns to the main process where one of the possible other reactions is non-deterministically chosen. This strategy is implemented using synchronization channels, i.e. a handshaking communication between the two processes. The reaction of interest is contained in the control process, and is executed only if all the metabolites are available. At each step the control process will try to execute the reaction, and after it will return the control to the main process. The details of the implementation are given in the Promela code

```
/* THE METABOLITE VARIABLES */
 byte GC = 2;
 byte GO = 2;
 byte AL= 2;
 byte GL=2;
 byte PY=2;
 byte OX=2;

/* CHANNELS AND MSG FOR THE SYNCHRONIZATION PART*/
 chan R_goes_chan[2] = [0] of mtype; // An array of two channels.
 chan S_goes_chan = [0] of mtype; // A channel
 mtype token; // the token passed on the channels.

/* THE REACTION PROCESS */
   proctype R_process() {
     do
      /* INTERNAL REACTION N. 1: GC  → GO */
      :: atomic{ R_goes_chan[((GC > 0) -> 1 : 0)]?token  →  GC=GC-1; GO=GO+1; S_goes_chan!token}
      /* INTERNAL REACTION N. 3: GL  →  GO */
      :: atomic{ R_goes_chan[((GL > 0) -> 1 : 0)]?token  →  GL=GL-1; GO=GO+1; S_goes_chan!token}
      /* INTERNAL REACTION N. 4: GO  →  OX */
      :: atomic{ R_goes_chan[((GO > 0) -> 1 : 0)]?token  →  GO=GO-1; OX=OX+1; S_goes_chan!token}
     od
   }

/* THE CONTROL PROCESS */
 proctype S_process(){
  atomic{
    do
      ::S_goes_chan?token; if
                           /* INTERNAL REACTION N. 2:  GO + AL  →  GL + PY */
                           :: ((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1
                           :: else;
                           fi;
                           // THE CONTROL IS PASSED BACK TO THE REACTION PROCESS
                           R_goes_chan[1]!token;
    od
}
}


/*ANTI-DEADLOCK PROCESS, ACTIVATES S_PROC*/
 proctype Once_process(){
         S_goes_chan!token;
         }

/* START THE PROCESSES  */
        init{
              atomic{
                      run Once_process(); // runs first the Once_process to avoid deadlock
                      run S_process();
                      run R_process();
              }
         }
```

Promela code snippet 8:

snippet 8. Three process are now present: the principal one, called "R_process", containing all the reactions that should be non-deterministically chosen; the control one, called "S_process", containing only the reaction of interest; and the "Once_process" containing the first activation of the channel. This last process is essential to avoid an initial deadlock state. In fact, the principal and control processes may communicate only through a synchronous handshake: this means that one process should be sending and the other one

receiving, or vice-versa. However, in the first state they are both in a receiving state. The Once_process passes the token to the control process, and then terminates. The control process contains the reaction of interest, that is executed if all the needed metabolites are present. After, the control is passed to the main process. As before, this process contains a *do* statement, expressing the non-deterministic choice between the contained reactions. However, it may happens that the selected instruction is not executable due to the lack of some metabolites. To avoid deadlocks, the receiving instruction may be executed only if all the metabolites are available: in fact, only in in this case the logical expression in the receiving clause will evaluate to one, thus matching the channel array id of the send. In all other cases it will evaluate to zero: this index corresponds to a dummy channel expressly built to deflect handshake attempts that should fail.

Of course, this approach may be used together with the verification/filtering techniques described in section 5.4. However, the property instructions must be placed with some care: to be sure of their executability they should be placed before a synchronization point (i.e. before a write instruction). Hence, the best place for an assert clause is in the S_process, right before it passes back the control to the main process: this guarantee that it is evaluated at each cycle of execution. The flags for the monotonicity check should be placed as usual in the atomic clause surrounding the reaction of interest, but before the token passing, that should be the last instruction. Instead, a timeout clause may be safely placed as before at the end of the do cycle in the main process: if no reaction is executable when the control passes back to the main process, then the system is in a deadlock state that will trigger the timeout.

This strategy can be used both to simply constrain the execution space, filtering out the more meaningful behaviours, both to **perturb** the model. For instance it can be used to simulate the expression, suppression or over-expression of a gene involved in the production of an enzyme, that in turn controls presence, absence or over-usage of a reaction, as briefly described in the following sections.

Switching Reactions on/off

The handshaking strategy above described may be used to specify a reaction that should always take place in a normal model (usually called *Wild Type* model). Hence, the presence of the control process assures that the reaction is performed each time the needed metabolites are available, thus simulating the preference of the Wild Type (WT) model for the executions in which the reaction of interest works well. A simple change in the code of the Snippet 8 makes it possible to switch off a reaction: it suffices to put a *skip* instruction in place of the reaction execution to simulate a diseased model (called also *Loss of Function* or *Knock-Out* model).

Multiple Execution of a Reaction (Gene Overexpression)

Another biological feature that may be simulated using the handshaking approach above described is the over–usage of a reaction. This can be obtained altering the ”speed of the execution“: normally a reaction takes place only once in a single ”time unit“ (i.e. an execution step). Instead, to represent over-expression, a reaction is performed as many times as possible in a single time unit, and this process is limited only by the availability of the right amounts of the needed metabolites. This is obtained simply substituting a *do* cycle to the *if* block in the controller process, as shown in the Promela code snippet 9. With this construct, the essential behaviour of an enzyme over-expression is modeled: the correspondent reaction takes place more rapidly than in a normal system, provided that the needed metabolites are present.

```
/* THE CONTROL PROCESS */

proctype S_process(){
  atomic{
    do
      ::S_goes_chan?token; do
                           /* OVEREXPRESSION OF REACTION N. 2:  GO + AL  →  GL + PY */
                           :: ((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1
                           :: else-> break;
                           od;
                           // THE CONTROL IS PASSED BACK TO THE REACTION PROCESS
                           R_goes_chan[1]!token;
    od
}
}
```

Promela code snippet 9: This code fragment shows how the control process is changed to implement the multiple execution of a reaction. All other parts remained unchanged (see Promela code snippet 8)

5.5.2 Reactions with Priorities

The second control technique we have designed deals with the possibility of specifying sets of privileged reactions, whose execution is favourite with respect to the others. The reactions in a biochemical model are usually considered as being all equally executable, and are essentially constrained only by the availability of the needed substrates. However, this assumption may lead to neglect interesting features of the biological systems. In fact,

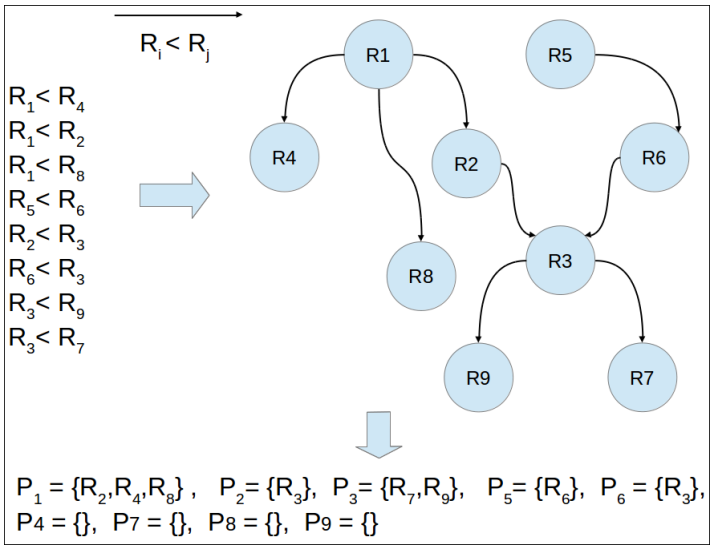

Fig. 5.2. Building priority sets. Priorities are usually defined between pair of reactions. Starting from the pairwise relationships a directed graph is build. A priority set for a node (i.e. a reaction) contains all the direct successor of the node itself. Nodes without successors have an empty priority set. A reaction with an empty priority set may be always executed. A reaction with a not-empty priority set may be executed only if *all* the reactions in its priority set can not take place.

sometimes may be of interest to model *priorities* between reactions that compete for the same substrate (such as reactions $R2$ and $R4$ in the example of figure 5.1), or that are tightly coupled in production/consumption cycles of the same metabolites (such as reactions $R2$ and $R3$ in the same example). In those cases, establishing priorities among the involved reactions may help in building more effective and realistic models. The

priorities may be established taking into account biological informations, such as the relative strength of the enzymes regulating the reactions.

When a priority relationship is defined among two reactions, the one with the lower priority is blocked until the other one can be executed. Referring again to the example of figure 5.1, if $R2 > R4$ (i.e. $R2$ has greater priority than $R4$) then if $R2$ may take place, $R4$ is skipped. Hence $R4$ may be executed only if $R2$ can not *and* all the metabolites needed for $R4$ are available We will now describe in more detail this strategy.

Complete inhibition

As shown in the examples above, typically priorities are defined among reaction pairs. We can write $r_i < r_j$ if the reaction i has a priority lower than reaction j. Starting from these pairwise relationships, a priority set may be build for each reaction with a simple graph-based exploration procedure, as shown in figure 5.2. A directed graph is built representing the priority relationships between the reactions. Then the priority sets are built exploring the graph. For each node representing the reaction r_i, the priority set P_i contains all the direct successors of r_i. Hence, nodes without direct successors have an empty priority set. A reaction r_i can take place if and only if all the reactions contained in its priority set P_i can not be executed. Of course, reactions with an empty priority set can be executed at any time, since they do not depend from any other reaction.

Let be E_i the executability condition of a reaction r_i , $i \in (1..m)$, where m is the total number of reactions in the system. Each E_i is a test expression that is true if and only if all the metabolites needed to the reaction i are available, and is false otherwise (see section 5.3.2).

The priority set associated to the reaction r_i is $P_i = r_l, ...r_k$ with $(l, ..., k) \subset (1..m)$ and $(l, ..., k) \cap (i) = \emptyset$.

Lastly, we indicate with "$\neg$" the boolean connector *not* and with "$\wedge$" the boolean connector *and*.

A reaction r_i can take place iff $(\neg ((E_l) \wedge \cdots \wedge (E_k)) \wedge (E_i))$.

That is, a reaction r_i with priority set P_i can take place if and only if no reaction in set P_i can take place *and* all the metabolites needed to r_i are available. For a reaction r_j with set $P_j = \emptyset$ nothing changes: as before, it can be executed whenever its executability condition E_j becomes true.

The Promela snippet 10 shows the Promela implementation for the example of figure 5.1. Reactions R3 and R4, beyond having their own executability clause, also have the negation of the executability clause of R2. It should be noted that in the case of R4 the overall executability clause may also be rewritten as follows: $E_4 = (\neg((GO > 0) \wedge (AL > 0))) \wedge (GO > 0)) \Rightarrow E_4 = (((GO <= 0) \vee (AL <= 0)) \wedge (GO > 0))$. Since concentrations may not be negative we have: $E_4 = (((GO = 0) \vee (AL = 0)) \wedge (GO > 0))$. However, to be executed reaction R4 needs in input GO, hence it will be executable only when $((AL = 0) \wedge (GO > 0))$, i.e. only when the AL metabolite, essential for R2 to take place, is not available *and* the GO concentration is greater than zero (i.e. GO is still available). For reaction R3 a similar reasoning leads to the following final executability clause: $E_3 = ((GO = 0) \vee (AL = 0)) \wedge (GL > 0))$. In this case it suffice that only one of the metabolites needed from R2 is not available to make R3 executable. When R2 is not more executable, then R3 and R4 become both executable (provided that GO is still available, of course): in this case the choice of the reaction to perform will be purely non–deterministic.

In this approach, a reaction is completely blocked until the reactions from which it depends (i.e. the ones with higher priority) remain executable. Please do note that we decided to consider only the direct successors of each node. Another possible choice was to extend the dependency of a reaction also to all successor nodes. However, this would be a very stringent and unlikely biological constraint, and was not further considered.

Also, the effect of the priorities on the reactions may be modeled in a different way: instead of completely blocking the reactions with lower priority, a fixed rate of executions

```
/* REACTION PRIORITIES: */
/* R4 < R2 ; R3 < R2 */


/* PRIORITY SETS: */
/* P1= {};  P2= {} ; P3= {R2}; P4= {R2} */


/* INTERNAL REACTION N. 1: GC  →  GO */
/* INTERNAL REACTION N. 2:  GO + AL  →  GL + PY */
/* INTERNAL REACTION N. 3: GL  →  GO */
/* INTERNAL REACTION N. 4: GO  →  OX */

   proctype REACT_process() {
     do
        :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1;}
        :: atomic{((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1 }
        :: atomic{(!((GO > 0) && (AL > 0))) && (GL > 0))  →  GL=GL-1; GO=GO+1;}
        :: atomic{(!((GO > 0) && (AL > 0))) &&(GO > 0))  →  GO=GO-1; OX=OX+1;}
     od
   }

/* START THE PROCESS  */
      init{run REACT_process()}
```

Promela code snippet 10: This code fragment shows how priorities between reactions are implemented. Reaction R4 and R3 have a lower priority with respect to reaction R2. Hence they can be executed only if R2 is blocked. This is obtained coupling the executability clauses of R3 and R4 with the negation of the R2 clause.

may be established. Using again the same example, if $R2 > R4$ (i.e. $R2$ has greater priority than $R4$) then the resulting effect of this new policy is that if $R2$ is executable, $R4$ is skipped as before. However, after a fixed number of times, a possibility to execute is given to $R4$: as soon as the needed metabolites are available, $R4$ may take place, and then the cycle starts again. We decided to explore this possibility in our future works (see section 8.1).

Concluding, from these examples it clearly emerges that the set of possible computations is drastically reduced by the introduction of the priority constraints. Hence, priority definition has a double beneficial effect: it reduces the state space to explore (without increasing the model complexity), and permits to refine the model adding meaningful biological informations.

5.6 Comparing models

At this point, we have described in detail all the building blocks needed to successfully adapt the model checking approach to the metabolic network context. All these pieces may be assembled in different fashions to obtain executable models of biological system with varying phenotypes. It is worth noting that the final products of the verification/simulation of different models are counterexamples, which are always made of metabolite concentrations and reaction sequences. This is an important feature, since it permits to readily compare biologically different models. We will exploit this feature when working on a real model, as described in chapter 7.

5.7 Chapter Summary

This chapter is the core of whole thesis, since it describes in details the essence of the proposed methodology. We started introducing a small example to permit an easier understanding of the relationships between the network model and the executable model. Then we present the fundamental idea of using the model checker as a *behaviour filter* to

extract witnesses of the ability of the model to satisfy a property. Limits and strengths of such an approach are discussed. The remainder of the chapter is mainly organized in three parts: the first describes in detail how a Promela model of a biological network is realized, with a section entirely dedicated to the model abstraction techniques that permitted do contain the computational complexity of the verification/behaviour filtering processes. The second part of the chapter presents in detail the type and the formal definition of interesting biological properties, such as the concentration reachability or the monotonous trend properties. A section address the issues related to the use of approximated strategies to explore the network model, and the strengths and limits of such a strategy are thoroughly discussed. Furthermore, the use of a timeout clause to investigate on the final states of the network is also presented, and a final section describes the implementation tricks that may give the best results in terms of complexity. The third, and last, part of the chapter is devoted to the presentation of the control elements that were designed to address the problem of reducing the search for counterexamples only to some portions of the computation space by selecting interesting transitions or defining priorities among them.

6

An Analysis Workflow

In this chapter we will describe in detail how the verification and behaviour filtering strategies presented in the previous chapter have been implemented. In section 6.1 we outline the elaboration workflow that has been designed to analyse a metabolic network model. Each step of this pipeline will be illustrated in detail in a dedicated subsection, with a particular attention on the final outcome of the whole pipeline. Furthermore, in section 6.2 we will address the issue of a satisfactory tuning of the approximated search strategies when dealing with timeout/non timeout behaviour filtering processes. In this chapter we will describe the details of the implementation and the general parameter settings used to obtain the results shown in chapter 7, where the proposed workflow is applied to a real-sized case study. However, this is a very general framework, that can be adapted to the study of any biological system that may be described through a metabolic network.

6.1 The Elaboration Pipeline

A schematic view of the proposed pipeline is shown in figure 6.1. The input is a metabolic network model, expressed either as a stoichiometric matrix or as a SBML file (see subsection 2.3.3). The input file is passed to the *Promela Translation Module* that converts it into an executable Promela model. If the network contains bi-directional reactions, or a pruning step is needed, then a *Flux Balance Analysis* step is performed and the resulting pruning and/or reaction directions are passed to the *Promela Translation Module* together with the network model. The translation module expects in input also a file containing the initial variable values, and a file with the property specification. At this stage, also a control strategy may be defined as described in section 5.5. The output of the *Promela Translation Module* is a Promela program containing both the model and the property specification. The Promela program is verified and/or simulated using the Spin Model Checker. Approximated searches are defined and fine-tuned used the Swarm tool. The counterexamples found by the model checker are saved in special-format files called *trails*. Trail files are translated in textual format and information are extracted using the *Extraction Module*. The Promela Translation Module and the Extraction Module are both written in the Perl programming language [147]. This choice it is due to several reasons, listed hereafter.

- Perl is a language explicitly designed to be easy to use and efficient;
- Perl has a powerful built-in support for text processing, an essential feature when working with thousands of counterexamples as we do;
- Perl has one of the world's most impressive collections of third-party modules, that permit to easily solve a wide range of common problems;

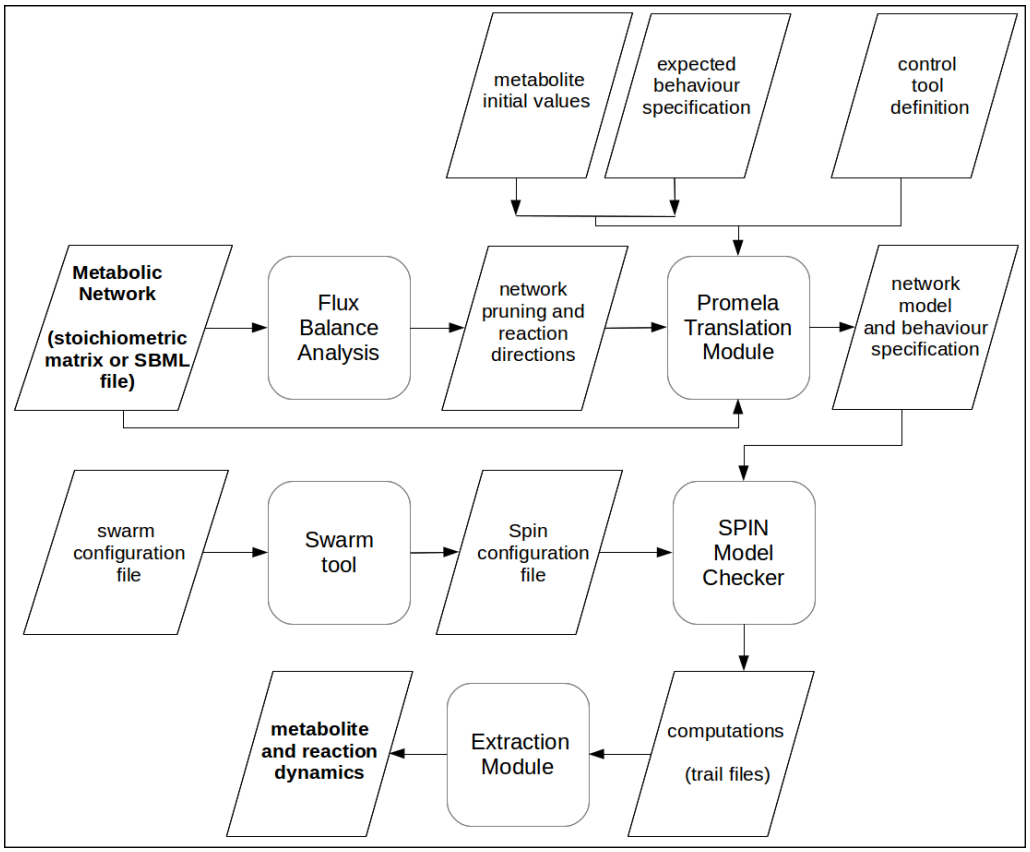

Fig. 6.1. The elaboration workflow. The input is a Metabolic Network, either represented as a SBML file or as a stoichiometric matrix. If the network contains bi-directional reactions or if a pruning is needed to reduce its size, a Flux Balance Analysis step is performed. The resulting information are passed to the *Promela Translation Module* together with the network. This module expects in input also the initial values of the metabolite variables and the property specification. If needed, a control tool may be defined according to the strategies described in section 5.5. The generated Promela program is passed to the model checker Spin. The *swarm* tool is used to fine-tune the configuration parameters when an approximated search strategy is needed. The counterexample computations generated by the checker are saved in the *trail* files. These special-format files are translated in a textual format by the *Extraction Module*, whose final results are the information on the performed reactions and the metabolite values.

- Perl has a community, grouped under the *BioPerl* project [148], that develops and maintains code expressly designed to tackle issues related to the bioinformatics and computational biology fields;
- Perl is free and open access.

We will now describe in depth the proposed pipeline, detailing the input, output and inner mechanisms of each module. The general settings here described have been tested on a real, large-sized model of an human hepatocyte, as described in chapter 7.

6.1.1 The Flux Balance Analysis Step

The Flux Balance Analysis (FBA) step has been performed as outlined in subsection 2.3.4 using the Cobra extension of Matlab [63]. This part has been developed by R. Pagliarini at the Telethon Institute of Genetics and Medicine (TIGEM) of Naples in the context of a genome-wide study of the human hepatocyte, described in Pagliarini and di Bernardo [72].

However, thanks to the modularity of the whole pipeline, in principle any sound FBA strategy could be used in place of the above cited approach, provided that the final results are the fluxes traversing the reactions contained into the network. The output of the FBA step should be a textual file containing m lines, e.g. one entry for each reaction in the network.

6.1.2 The Promela Translation Module

The Promela Translation Module contains a set of programs designed for the specific purpose of converting a metabolic network into a Promela program. Each of the developed Perl programs deals with two network input format (i.e. SBML or stoichiometric matrix), or with the different kind of structures needed to verify several types of properties. Moreover a completely different Promela program should be created if one of the control structures described in chapter section 5.5 is specified.

In input the module needs several files: the network definition, the property specification, the initial values of the variables, and, optionally, the pruning/direction files. In output it generates a Promela program. The format of the various files are described in more detail in the following sections. Among the files that are needed in input, only one is optional, namely the one containing the reaction directions or the network pruning. If it is not specified, the whole network is loaded from the network input file, and the reaction directions are established accordingly to the information there contained.

The metabolic network

The metabolic network in input may be given in input to the module either as a SBML file or as a stoichiometric matrix. In the former case, a specific BioPerl module is used to load the unique associated files, containing all the information about the network. Otherwise, three textual files are needed: one containing the stoichiometric matrix with n rows (the metabolites) and m columns (the reactions); a second file should contain n lines with identifiers and extended names of the metabolites (tab-separated); the third one should contain m lines, with identifiers and extended string of the reactions (tab-separated).

The identifiers will be used in place of the compound names and of the reactions string to have a more compact and safe generation of the Promela code.

The initial values

The initial value file should contain the type of each variable, and its initial value. It can contain a line with the keyword “ALL”, whose settings of type and value will be applied to all the variables, or n lines with the specification for different variables, or a mix of both the specifications. The variables can be declared using either the name or the identifier.

The property specification

The property specification file is a single line of text containing the Promela code for the assert that should be coupled with a specified Promela program. The timeout clause should be explicitly declared before the assert.

The pruning/direction file

The pruning file should have m columns. Each line contains a number $r_i \in \mathbb{R},\ i = 1, ..m$ which represent the mean flux through the corresponding reaction. Depending on the value of the flux, we have the following possibilities:

1. $r_i > 0$. The i-th reaction should be considered with its original direction, i.e. the one specified in the stoichiometric matrix or in the associated SBML file.
2. $r_i < 0$. The i-th reaction should be reversed, i.e. the reactants should be treated as products and viceversa.
3. $r_i = 0$. In this case the choice may be different depending on the needed strategy:
   a) the i-th reaction is ignored, since is carrying a zero flux, and is pruned i.e. cut away from the resulting network.
   b) the i-th reaction direction is randomly chosen.
   c) the i-th reaction direction is chosen according to the original direction in the network.

The default is to prune the network (since this operation reduces its size), however the other behaviours may specified via an input flag to the model.

Output

The output of the module is a Promela program, as the one presented in the Promela code snippet 11. Another file, called the *description* file, is generated together with the Promela program, and will be used by the Extraction Module to link the compounds and reaction identifiers with their descriptions.

6.1.3 The model checker Spin

We used the version 6.2.2 of the model checker Spin. When launching exhaustive searches, we have used it from command line or within a *bash* script, as in the example that follows:

```
#!/bin/bash
cd /user/sangiovanni/test/
‘spin -a  LTS_mini.pml‘
‘gcc -DMEMLIM=4096 -O2 -DSAFETY -DNOFAIR -DNOCLAIM -w -o pan pan.c‘
./pan -m10000 -E -n -c1
```

Here we first moved in the directory containing the Promela file (a necessary step to make the overall process work), and then we call Spin specifying the name of the Promela file to use. The compilation of the Promela program generates a C program, usually called *pan.c*. The *pan.c* is compiled into the corresponding executable, and in this phase the compile-time parameter should be specified. They specify how the model should be built, and the total memory it can use. In the above example we specified 4 GB of RAM (DMEMLIM=4096), to verify only safety properties (-DSAFETY) without fairness enforcement (-DNOFAIR) and without using a never claim automaton (-DNOCLAIM). The -O2 parameter is for C optimization.

After, the *pan* is executed passing the run-time parameters that specify details on how the search should be performed: the max reachable depth (-m10000), if the search should stop after the first error and how many counterexamples should be saved (-c1). The flag -E suppresses the report of deadlock states, the -n flag suppresses the report of unreached states. See the Spin Manual [12] for all other details on parameters. The output of a verification step are files called *trails*, containing the information on the visited states and the executed transitions.

6.1.4 The Swarm Tool

The swarm tool has been thoroughly described in the paper of Holtzmann et. al [122]. It is used to launch in parallel several approximated (i.e. bitstate) perlustrations of the state space, each one using a (possibly different) strategy, with the aim to maximize the explored portions exploiting multiple and different randomized searches. Here we will describe the parameter settings that we have used to run Spin via the swarm tool. The first step to use swarm is to define the compile-time and run-time parameters for the pan executable as global shell variables with a fixed name, as in the following example:

```
#!/bin/bash
export CCOMMON="-O2 -DSAFETY -DNOFAIR -DVECTORSZ=3072"
export RCOMMON="-e -c5 -E"
```

After, a configuration file must be created following the template that is given together with the swarm tool. All the details on its inner functioning and on the meaning of the parameters may be found in the already cited Spin Manual and swarm paper [12,122]. In

Listing 6.1. An example swarm configuration file.

```
## Swarm Version 3.1 -- 9 April 2011
# range
k 3 8  # min and max nr of hash functions
# limits
d       10000 # optional: to restrict the max search depth
cpus    2     # nr available cpus (exact)
memory  2G    # max memory (M=megabytes, G=gigabytes)
time    15s   # max time (h=hours, m=min, s=sec, d=days)
hash    1.5   # hash-factor (estimate)
vector  512   # nr of bytes per state (estimate)
speed   25000 # nr states explored per second (estimate)
file    LTS_mini.pml # file with the spin model to be verified

# each line defines one complete search mode
-DBITSTATE -DPUTPID -DP_RAND -DT_RAND
-DBITSTATE -DPUTPID -DP_RAND -DT_RAND -DT_REVERSE
-DBITSTATE -DPUTPID -DP_RAND -DT_RAND -DREVERSE
-DBITSTATE -DPUTPID -DP_RAND -DT_RAND -DREVERSE -DT_REVERSE
```

particular, the swarm configuration file contains the specification of the *search modes*, i.e. the strategies used to explore the state space. Since in Spin the verification is performed executing the instructions in the order in which they are declared in the Promela program, an unlucky ordering may result in an infeasible search. Spin permits the random ordering of states (-DP_RAND) and transitions (-DT_RAND) to overcome this issue. Clearly, this is is an essential feature of the swarm search since it allows the exploration of different portions of the state space, and will be exploited on a real-sized problem as the case study of chapter 7.

However, the swarm tool uses the same file with some fixed and predefined seeds to launch all the different searches that include a random ordering. This means that the *same* random seeds are always used in the same order, also if the swarm file is compiled again. Hence, simply compiling two different swarm script does not imply using different seeds, i.e. a different ordering of states or transitions. This corresponds in exploring always the same portion of the space, that in some cases may not be useful. The above example of variable setting and the swarm configuration file of listing 6.1 were used to extract the data of subsection 6.2.

6.1.5 The Extraction Module

We will now describe in detail an essential step of the pipeline, i.e. the Extraction Module. The purpose of this module is the translation of the results coming from the Spin model checker into useful knowledge about the used metabolites, the performed reactions and their dynamics in time. This target is reached with the combination of two distinct steps: a translation phase, in which the information coming from the trail files is converted and stored into binary files, and a subsequent one in which the binary files are processed to collect structured information about the system. We will describe them in more detail

in the following subsection. It is worth noting that this pipeline has been conceived essentially to deal with the behaviour filtering analyses. However, other kind of processing on the results may be performed as briefly outlined in subsection 6.1.6.

The Translation Phase

The result of a verification/extraction step performed by Spin is a set of files with extension *.trail*. Each of these files records the steps that lead from the initial state to the one in which the assert has been triggered. They are not immediately exploitable, since they use an internal representation of states and transitions. However they are easily translated in a textual file calling Spin with the following instruction

```
spin -g  -p  -k $trail -c $promelapgm > $transtrail
```

where the input variable should respectively specify the generated trail file, the Promela program used to generate the trail and the destination file that will contain the textual translation. The textual file that is obtained looks like the one of listing 6.2. This listing contains all the information needed to reconstruct the metabolite dynamics and the used reactions. In fact on each step is listed the line of the Promela program that was executed, and, immediately after, the list of the changed variables together with their new values. At the end of the file the final values of all the variables are listed. It is only a matter of extracting, storing and processing all the information.

However, this can be a very though task when the program contains thousands of reactions and metabolites, and the generated trails are in the order of hundred of thousands. To address this problem in the most efficient way, some *Perl* programs have been developed to rapidly parse the textual trail files and convert them into the small possible pieces of binary information. To speed this process, we designed a *bash* pipeline to run in parallel as many possible jobs, each converting and writing a subset of all the generated trails.

Two separate programs were designed to convert the trail files, each one that works on a single file at a time: one extracts from a trail file the overall information on the final compound values and on the total reaction usages, the other extracts from the trail file detailed information on the metabolite dynamics, saving the time steps and the reaction that changed the metabolite values. They are described in more detail in the following subsections.

The overall information

For each trail file the the following two files are generated:

- a compound file: it contains as many lines as the used metabolites. Each line is made up of two fields, a fixed length char for the metabolite code (i.e. the identifier specified in the metabolic network model, see subsection 6.1.2), and a long unsigned integer for the final metabolite value;
- a reaction file: it contains as many lines as the used reactions. Each line is made up of two long unsigned integer fields, one for the Promela program line number code (that corresponds to the performed reaction), the second for the usage value, i.e. for the total number of times the reaction has been used.

The dynamics information

For each counterexample the following two files are generated:

- a compound file: it contains as many lines as the number of steps in which some metabolite has changed its value. Each line is made up of three fields, a long unsigned integer that specifies the step number, a fixed length char for the metabolite code (i.e. the identifier specified in the metabolic network model, see subsection 6.1.2), and a long unsigned integer for the metabolite value at the end of the step;

Listing 6.2. An excerpt of a translated trail file

```
proc 0 = :init:
using statement merging
Starting R_proc with pid 1
proc 1 = R_proc
  1:    proc  0 (:init:) LTS_mini.pml:29 (state 1) [(run R_proc())]
  2:    proc  1 (R_proc) LTS_mini.pml:17 (state 1) [((Gc_ext>0))]
  2:    proc  1 (R_proc) LTS_mini.pml:17 (state 2) [Gc_ext =
           (Gc_ext-1)]
                Gc_ext = 1
  2:    proc  1 (R_proc) LTS_mini.pml:17 (state 3) [GC = (GC+1)]
                Gc_ext = 1
                GC = 1
                ....
                ....
 17:    proc  1 (R_proc) LTS_mini.pml:21 (state 19) [((GO>0))]
 17:    proc  1 (R_proc) LTS_mini.pml:21 (state 20) [GO = (GO-1)]
                GO = 0
 17:    proc  1 (R_proc) LTS_mini.pml:21 (state 21) [OX = (OX+1)]
                GO = 0
                OX = 8
 18:    proc  1 (R_proc) LTS_mini.pml:22 (state 23) [(timeout)]
spin: LTS_mini.pml:22, Error: assertion violated
spin: text of failed assertion: assert((OX<=2))
 19:    proc  1 (R_proc) LTS_mini.pml:22 (state 24)
          [assert((OX<=2))]
spin: trail ends after 19 steps
-------------
final state:
-------------
#processes: 2
                Gc_ext = 0
                GC = 0
                GO = 0
                AL = 0
                GC = 0
                PY = 4
                OX = 8
 19:    proc  1 (R_proc) LTS_mini.pml:16 (state 25)
 19:    proc  0 (:init:) LTS_mini.pml:30 (state 2) <valid end
          state>
2 processes created
```

- a reaction file: it contains as many lines as the used reactions. Each line is made up of two long unsigned integer fields, one for the step number, the other for the Promela program line number code (that corresponds to the performed reaction).

Hence two separate procedures are run, one to collect the overall compound and reaction information, the other to extract the dynamics. Both will work on the complete set of trail files that were generated by Spin, launching in parallel as many jobs as possible. At the end of the conversion phase, the first procedure has created two directories storing the overall information, with each directory containing as many files as the number of extracted trail files. The second procedure has created a directory containing two files for each of the processed trail files.

The Extraction Phase

After, the binary files are parsed to extract a more compact view of the system behaviour. This task is accomplished by three distinct procedures, that can be launched in parallel and independently from each other: the first collects the information on the final metabolite values across the different counterexamples; the second collects the information on the total *reaction usage*, i.e. the total number of times a certain reaction is performed across the different counterexamples; and the third translates the dynamics of a metabolite in a more human-readable format. Each of them extracts the data stored in the binary files described in the previous subsection, elaborates them and, at the end of the process, creates one or more textual files. The data contained in these files are described in more detail in the next subsections. Since the extraction of the dynamic information is a very computational demanding process, only a subset of metabolites (specified in a file given in input to the procedure) is extracted.

Output

The output of the Extraction Module are *i)* the file containing the metabolite final values, *ii)* the file containing the reaction usage values, and *iii)* the directories containing the dynamics of the specified metabolites. The first two files offers already a structured knowledge of the data collected from the counterexamples, whereas the dynamic data might be further processed to extract more compact information. All of them will be described in more detail in the following subsections. We will use as an example the results obtained for the program of the Promela code snippet 3. The timeout behaviour filtering results in three trail files (see subsection 6.2 for more details).

The compound file

The compound file is a tab-separated textual file, that contains the information on the value of the variables across all the extracted counterexamples as the one shown in listing 6.3. The columns contain on each row, i.e. for each compound:

Listing 6.3. The compound usage values file

```
COMPOUND  COMPOUND  INITIAL  USAGE   MIN    MAX     USAGE MEAN AND
ID        NAME      VALUE    VALUES  VALUE  VALUE   DISPERSION
LC00001   Gc_ext     2       0;      0      0        0 + - 0
LC00002   GC         0       0;      0      0        0 + - 0
LC00003   GO         2       0;      0      0        0 + - 0
LC00004   AL         2       0-2;    0      2        1 + - 1
LC00005   GL         2       0;      0      0        0 + - 0
LC00006   PY         2       2-4;    2      4        3 + - 1
LC00007   OX         2       8;      8      8        8 + - 0
```

1. the unique identifier of the compound (see subsection 6.1.2);
2. the name;
3. the initial value: it is the concentration value when the verification process was started, i.e. the one indicated to the Promela Translation Module (see 6.1.2)
4. the range of final values. They are called final since they indicate the metabolite concentration in the last state of each extracted counterexample. It can be a unique value, thus meaning that the metabolite has reached always the same concentration in all the different counterexample (such as the OX or the GC_ext compounds in

listing 6.3). Otherwise a metabolite concentration may be different across the counterexamples: in this case they are indicated by ranges, i.e. by a pair of numbers separated by a dash symbol (–), meaning that the whole range of values, included the extremes, have been reached by the metabolite in different counterexamples.
5. minimum and maximum final value;
6. mean and dispersion of the final value, calculated across all the counterexamples.

The reaction file

Similarly, the reaction file is a tab-separated textual file, that contains the information on the usage values of the reactions across all the extracted counterexamples. An example is shown in listing 6.4. The columns contain for each row, i.e. for each reaction:

Listing 6.4. The reaction usage values file

REACT. ID	REACTION STRING	USAGE VALUES	MIN VALUE	MAX VALUE	USAGE MEAN AND DISPERS.
1	1 Gc_ext-->1 GC	2;	2	2	2 +- 0
2	1 GC-->1 GO	2;	2	2	2 +- 0
3	1 GO + 1 AL-->1 GL + 1 PY	1-2;	1	2	1.33 +- 0.58
4	1 GL-->1 GO	2-4;	2	4	3 +- 1
5	1 GO-->1 OX	6;	6	6	6 +- 0

1. the unique identifier of the reaction (see subsection 6.1.2);
2. the string describing the reaction;
3. the usage value: each reaction can be used a certain amount of times between the first and the last state of the counterexample. The usage value of a reaction is exactly the number of times the reaction has been used. It may be the same across all the counterexamples, as happens for the reaction 1Gc_ext $\longrightarrow$ 1GC or may vary as for the reaction 1GO + 1AL−¿$_1$GL + 1PY in listing 6.4. In the latter case they are indicated by ranges, i.e. by a pair of numbers separated by a dash symbol (–), meaning that the whole range of values, included the extremes, have been reached by the usage value in different counterexamples;
4. the minimum and maximum final value;
5. the mean and dispersion of the final value calculated across all the counterexamples.

The compound dynamics

The dynamics extraction procedure creates a directory containing as many directories as the generated trail files. Each of these subdirectory contains a distinct files for each of the compound specified in the input file. The single dynamics file looks like the one in listing 6.5 For the compound indicated on the first line, one or more lines are present, depending on the number of step (if any) in which the metabolite was changed. Each line contains:

1. the step id. This can be considered as the time in which the metabolite was changed. By convention the initial state is considered as the step 0;
2. the value of the compound concentration at the end of the time step;
3. the reaction string, i.e. the event that changed the compound value. The initial value has no reaction associated, of course;
4. the reaction unique identifier.

Listing 6.5. The dynamics file for the GC compound.

```
DYNAMICS FOR COMPOUND ID LC00005 ( GC ):
STEP N. VALUE    REACTION STRING             REACTION ID
0       2        initial value
7       3        1 GO + 1 AL-->1 GC + 1 PY   3
8       4        1 GO + 1 AL-->1 GC + 1 PY   3
9       3        1 GC-->1 GO                 4
10      2        1 GC-->1 GO                 4
11      1        1 GC-->1 GO                 4
12      0        1 GC-->1 GO                 4
```

The sequence of steps together with the metabolite values may be regarded as a temporal series describing the dynamics of the metabolite concentration over the time, where time itself is considered a discrete quantity that flows in a stepwise manner.

The information contained in these file may be exploited as a whole by simply collecting the dynamics across the different counterexamples as done on a real case study in section 7.4

6.1.6 Other Processing

All the data extraction process above described has been expressly tailored for the behaviour filtering process. However, sometimes the only information needed is the simple presence or absence of a counterexample after a verification step, such as in the the Metabolic Objectives Satisfaction or the Monotony Check verifications (both described in chapter 5). For these purpose some *ad hoc* bash shell scripts were designed to travel across the directories and extract the needed information on the presence of counterexamples.

6.2 Search Tuning With the Swarm Tool

We would like now to investigate more deeply the effects of the swarm search strategies on the representativity of the extracted counterexamples, when using the system as a behaviour filter. The aim of a swarm approximated search is to gain as much information as possible, while relying on a partial exploration of the state space. The essence of this approach is the use of multiple different searches to expand the coverage of the state space. However, when using this technique in combination with the behaviour filtering strategy, a careful tuning of the search parameters may have beneficial effects on the quality of the extracted counterexamples.

In fact, for a specified property, the search tuning may deeply affect the number and the structure of the extracted counterexamples depending on the presence or absence of the timeout clause. When an exhaustive search is performed, of course, we have not these issues, since the state space is completely explored. Instead, if an approximated search is used, we shall distinguish between behaviours that are extracted using the timeout and behaviours that results from the information collected in intermediate states. We use the example of section 5.1 to show the differences among the results, in both timeout and non timeout searches, obtained with different search strategies related to the specification of the number of extracted counterexamples and of the number of search modalities.

We want to filter out the behaviours in which the compound OX is produced, and to this aim we define the assert as detailed in Property 1, i.e. assert(OX $\leq$ 2). Moreover, we want to extract all the possible counterexamples that Spin would generate, hence we will use the inline assert strategy (see subsection 5.4.6), placing the assert instruction after all the reactions. The resulting program is shown in the Promela code snippet 11.

```
/* THE METABOLITE VARIABLES */
  ....
 byte OX=2;
 ...

   proctype REACT_process() {
     do
        :: atomic{(GC > 0)  →  GC=GC-1; GO=GO+1; assert(OX <= 2);}
        :: atomic{((GO > 0) && (AL > 0))  →  GO=GO-1; AL=AL-1; GL=GL+1; PY=PY+1; assert(OX <= 2); }
        :: atomic{(GL > 0)  →  GL=GL-1; GO=GO+1; assert(OX <= 2);}
        :: atomic{(GO > 0)  →  GO=GO-1; OX=OX+1;  assert(OX <= 2);}
     od
   }

/* START THE PROCESS  */
      init{run REACT_process()}
```

Promela code snippet 11: The Promela program used for the exhaustive search of all the possible counterexamples in which the OX compound is produced

When we restrict the search only at the final states, we will use the timeout instruction, and the resulting Promela program will be the one of the code snippet 3. The results of the exhaustive search for the two cases are shown in table 6.1. These results and all the the other presented in this subsection have been collected as described in subsection 6.1.5. Of course the concentration ranges are quite different among the two strategies,

Table 6.1. Metabolite concentration values (in generic units) after two different exhaustive searches involving respectively a non timeout and a timeout strategy. The number of extracted counterexamples for each of them is shown. The concentrations are reported as ranges. Numbers separated by a dash symbol (–) means that the whole range of values, included the extremes, have been reached by the metabolite. A single number indicates that the metabolite concentration has the same value among all the counterexamples.

	No timeout	Timeout
Gc_ext	0–2	0
GC	0–2	0
GO	0–5	0
AL	0–2	0–2
GC	0–4	0
PY	2–4	2–4
OX	3-8	8
Counterexamples	658	3

since the timeout generates only three counterexamples that corresponds to the three different final states of the LTS. These states are characterized by the different values of the compounds PY and AL. It is worth noting that the non-timeout concentrations include also the timeout ones.

Now we will address the case in which approximate searches are performed. Several parameters may affect the obtained results: among them there are some more connected with the availability of needed resources such as time, memory and number of CPUs. Of course, for them the more, the better. Here we will take into account other settings, namely the number of extracted counterexamples per job, i.e. what we call the *max trail* parameter, and the number of distinct search strategies to use, i.e. the *search modes*, while maintaining unchanged the maximum elaboration time, set to 15 seconds (a very large value since the exhaustive problem is solved in about 2 seconds), the number of CPUs, set to 2, and the amount of available RAM, set to 2 GB. More specifically, the *max trail* parameter is important when extracting information using a non-timeout search. In this case, several counterexamples may be found as soon as a state s not satisfying the assert

is met: these are the counterexamples corresponding to nearby states, in which the value of the variable of interest is the same of s and the other state variables undergo some small fluctuations. That means that only the topmost part of the LTS is explored, and the range of the extracted counterexamples may be not so much representative, as happens for the results shown in table for the *max trail* parameter set to 5, as shown in table 6.2 In

Table 6.2. Metabolite concentration values (in generic units) for three different swarm runs of a non-timeout search. The value used for the *max trail* parameter is shown, together with the number of extracted counterexamples. Numbers separated by dash symbols (–) indicate range of values, including the extremes.

	Max trail parameter value		
	5	10	20
Gc_ext	0–2	0–2	0–2
GC	0–2	0–2	0–2
GO	0–3	0–5	0–5
AL	0–2	0–2	0–2
GC	0–4	0–4	0–4
PY	2–4	2–4	2–4
OX	3–5	3-7	3-8
Counterexamples	25	50	100

this case, the search extracts a very small number of counterexamples, too few to obtain a decent coverage of the results: the maximum OX value extracted is only 5 generic units. The second one, in which the value of the input parameter is doubled, results in a doubled number of extracted counterexamples. In this case the OX value is 7, almost near to the final value, and with only a small portion of the all possible counterexamples. Doubling again the parameter value (*Max trail*=20), we obtain the complete range of values for the OX compounds, but extracting only the 15% of the all possible counterexamples. A similar effect, of course, is found for the reaction usage values. Please do note that these results were all obtained using the same 4 distinct *search modes*, a high number for a very small problem. This was intentional, to demonstrate that the *search modes* setting does not suffice alone to extract a relevant subset of counterexamples in the case of a non-timeout search strategy. When using a timeout strategy, on the contrary, the

Table 6.3. Metabolite concentration values (in generic units) for two different swarm runs of a timeout search. The number of different *search modes* used is shown, together with the number of extracted counterexamples. Numbers separated by dash symbols (–) indicate range of values, including the extremes. A single number indicates that the metabolite concentration has the same value among all the counterexamples.

	Search modes number	
	2	4
Gc_ext	0	0
GC	0	0
GO	0	0
AL	0–1	0–2
GC	0	0
PY	3–4	2–4
OX	8	8
Counterexamples	4	10

different number of *search modes* may be of great importance. In fact each *search mode*

corresponds to the exploration of a different portion of the state space, hence widening the possibilities to find timeout counterexamples. The model we are using as example is really small, hence here the difference is not so striking. However on large models adopting different search strategies may be essential, as highlighted also by Holtzmann et al. in their paper describing the swarm strategy [122].

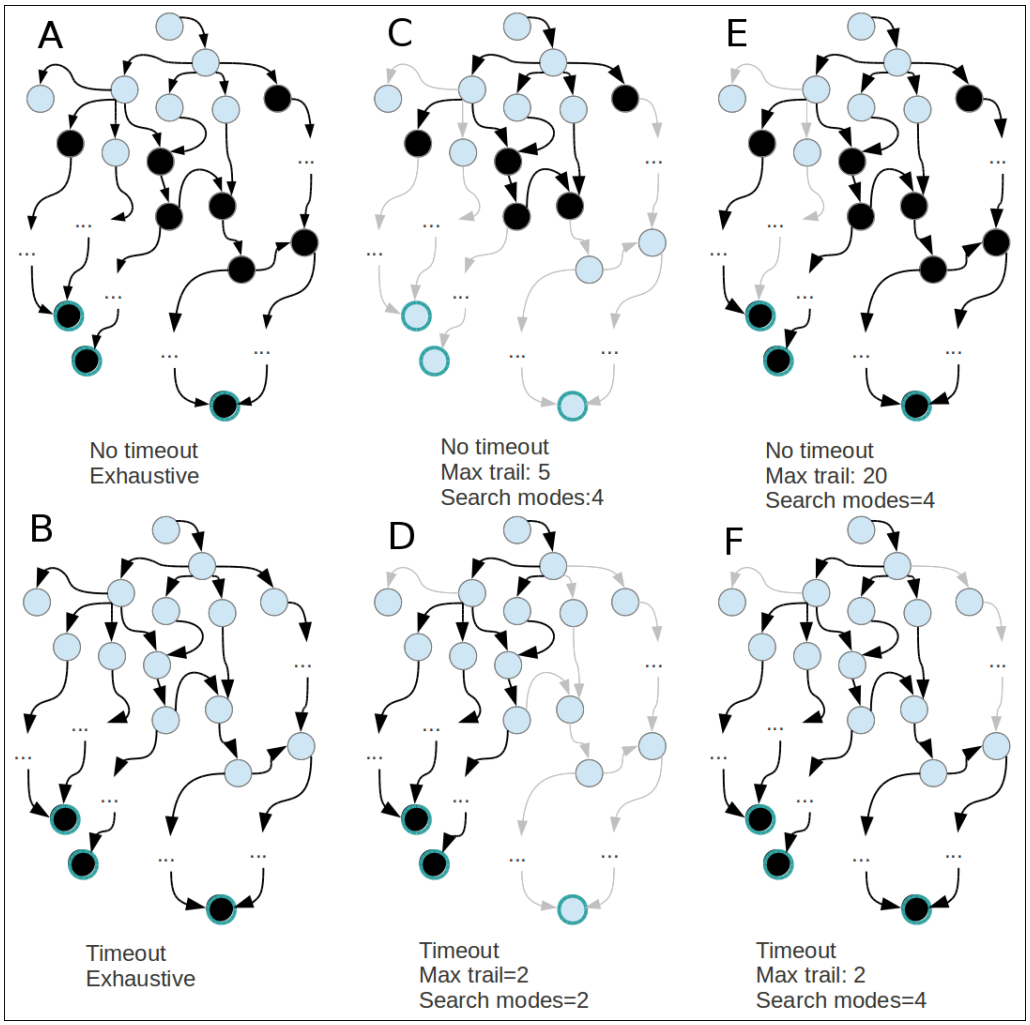

Fig. 6.2. A representation of the effects of different exploration strategies for the verification of a property on a Labeled Transition System. The thick blue-circled nodes represent final states. Black nodes represent states in which a counterexample is generated. Black arrows represent explored transitions, grey arrows unexplored ones. A: exhaustive search without timeout; B: exhaustive search with timeout; C: approximated search, without timeout; D: approximated search with timeout; E: the same of C, but with a larger setting for the *Max trail* parameter; F: the same of D, but with a larger setting for the number of *Search modes*. For approximated, non timeout searches (C, E) the max number of extracted trails affects the representativity of the extracted information. Instead, for approximated, timeout ones (D, F) is the diversification of search modes that plays an important role in the representativity of the extracted counterexamples.

Here we fixed the number of the extracted counterexamples to *max trail*=2. We first performed an approximated timeout search with 2 random *search modes*, and after we raised it to 4. The results are shown in table 6.3.

It is immediate to notice that the number of generated counterexamples, 4 and 10, is greater than the number of real final states, i.e. 3. This is due to the presence of more than one search modes, each one exploring a portion of the state space. Indeed, it may happen that these portions overlap: when only two *search modes* have been used, four counterexamples have been generated.

However, the extracted information, as can be deduced from table 6.3, are not complete. When the number of *search modes* is increased, a larger number of counterexamples is generated and all the concentration values are recovered. Of course, having so many identical counterexample is not of any use, and for this reason the number of the generated counterexamples, e.g. the *max trail* parameter should be possibly set to a low value.

All these search tuning strategies are graphically represented in figure 6.2. The results of an exhaustive search without a timeout (A) and with a timeout condition (B) are shown: in both cases all the transitions are explored. Without the timeout a counterexample is generated in all the intermediate states satisfying the property, whereas with the timeout only the counterexamples in the final states are generated. When extracting all the intermediate states (C,E), the specification of the number of extracted counterexamples has a great impact on the exploration of the states space: in C a low number of counterexample is rapidly generated by perlustrating a small part of nearby states in the topmost portion of the state space; in E an higher value of the *Max trail* parameter permits to explore more in depth the state space. When searching for timeout counterexamples (D,F), it is the number of the *search modes* that affects the results: in D only a reduced portion of the state space is explored; in F the diversification of the search strategies permits to explore more portions of the state space.

Of course this is a really simple task when the LTS is as small as the example one, but it is way more complicated on big models. Anyway, it is important to be aware of the influences of the two described parameters on the reliability of the obtained results.

We would like to conclude this section pointing out that these examples very well demonstrate, also on a very small state space, that is it possible to recover a great amount of information also with a very limited number of extracted counterexamples, as more thoroughly demonstrated in the Holtzmann et al. paper [122].

6.3 Chapter summary

In this chapter we have presented in detail how the proposed verification and behaviour filtering strategies have been implemented. A complete workflow was defined, that starts from a metabolic network model either represented as a stoichiometric matrix or as SBML file. Then a Perl module converts the metabolic network model into a Promela program that also contains the property specification, and, in case the control strategy definition. Moreover, the results of the Flux balance Analysis should be passed to the same module, if the specification of direction or a pruning is needed. The Spin model checker is used either as a verifier or as a behaviour filter, and in case some counterexamples, in the form of *trail* files, are generated. The approximated searches are managed through the swarm tool proposed by the Spin authors. Another Perl module extracts the information from the trail files. The resulting files are described in great detail, since they are the main outcome of the whole pipeline. They mainly contain: the metabolite concentrations (in generic units), expressed as ranges of values obtained in the extracted counterexamples; the reaction usage values, e.g. the total number of times in which a reaction is used, expressed as ranges of values obtained in the extracted counterexamples; the dynamics of a subset of selected metabolites, expressed in terms of the stepwise variation in each of the extracted counterexamples. Lastly, we considered the problem of tuning the swarm parameters when performing approximated searches for both timeout and non-timeout behaviour filtering. We show how a good parameter choice may result in obtaining a very

good quality of the collected results despite having extracted only a very small subset of the existing counterexamples.

7

Modeling the disease: Primary Hyperoxaluria Type I

In this chapter we will exploit the methodology described in the previous chapters to investigate on the causes and the effects of the Primary Hyperoxaluria Type I, a monogenic disease related to the misfunctioning of the liver enzyme AGT.

7.1 The Disease

The Primary Hyperoxalurias are inborn errors affecting the metabolism of glyoxylate and oxalate. Primary Hyperoxaluria type 1 (PH1) is the most common form, caused by a defect in the glyoxylate metabolism related to a low or absent activity of the liver-specific peroxisomal enzyme alanine/glyoxylate aminotransferase (AGT) [149, 150]. The

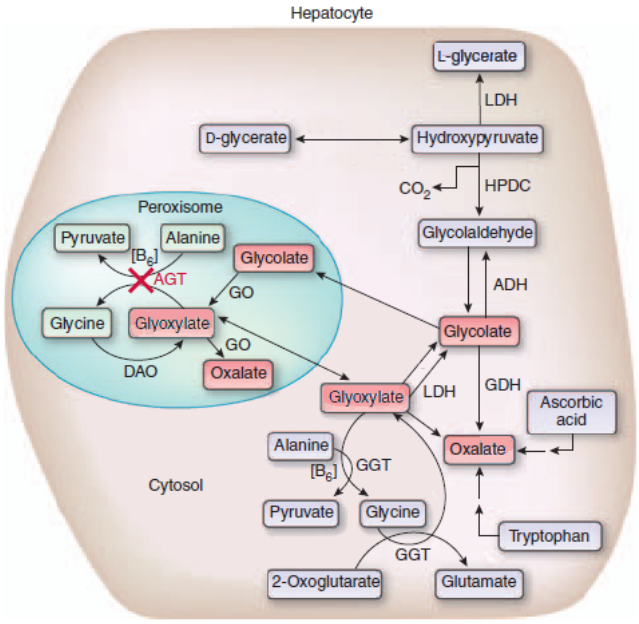

Fig. 7.1. A schematic view of the metabolic pathways involved in the Primary Hyperoxaluria Type 1 (PH1) disease. Arrows represent reactions, labels on it indicate the catalyzing enzymes. Double arrows represent postulated pathways, for which the intermediate reactions are still unknown. Metabolites directly involved in PH1 are indicated in red. Adapted from [151].

functional deficiency of AGT results in a failure to detoxify glyoxylate within the peroxisomes. Instead of being transaminated to glycine, glyoxylate is oxidized to oxalate

and/or reduced to glycolate, resulting in greatly increased urinary excretion of oxalate and glycolate [152].

Oxalate in mammals can not be further metabolized and its overproduction results in the deposition of calcium oxalate (CaOx) crystals and stone formation almost everywhere in the body [153]. Because of the poor solubility characteristics of calcium oxalate, an excess of oxalate has potentially lethal consequences: crystallization may occur in sensitive organs such as the kidney [149], eventually leading to renal failure, but may also involve eyes, heart, and bones with high invalidating effects, see [153, 154].

The PH1 is a monogenic disease caused by various mutations involving the AGXT gene [152]. These alterations may have different effects on the gene product, which cause in turn, different alterations in the functionality of the associated enzyme, that may be not produced at all, or may be produced but with alterations making it only partially active, or it may be fully active but carrying defects causing a mistargeting from the peroxisome to the mitochondria [153]. Hence, the resulting phenotypic traits cover a wide spectrum, ranging from almost silent cases to fulminant neonatal deaths ([151,153, 155]). However, they are all characterized at the biochemical and metabolic level, by an increased production of oxalate and glycolate [152, 154].

A schematic view of the fundamental reactions involved in the disease is shown in figure 7.1. The detoxification reaction catalyzed by the AGT enzyme takes place in the peroxisome of liver cells. Glyoxylate is transformed into glycine when AGT is present, otherwise the opposite reaction converts glycine into glyoxylate. The glyoxylate in excess escapes from peroxisome to the cytosol, where is converted either directly or indirectly through glycolate into oxalate. The cytosolic oxalate is then transported outside the cell in blood and urine. From the biochemical point of view the hallmark of PH1 is indeed the high production of cytosolic glycolate and oxalate.

7.2 The Extended Hepatocyte Model

HepatoNet1 (HN1) is a metabolic network reconstruction of the human hepatocyte recently published by Gille et al. in [156]. However, the proposed model is lacking all the pathways leading to oxalate production. The missing part was reconstructed and integrated into HN1 by Pagliarini et al. using knowledge derived from different sources as described in [72]. This new version, that we will call HepatoNet2 (HN2), contains 2589 reactions and 1445 metabolites, arranged in 8 cellular compartments. The model was validated through producibility analysis, to test its ability to produce all the compounds in the glyoxylate metabolism, and flux-balance analyses was used to establish a flux distribution for each of the different metabolic objectives listed in the already cited paper of Gille et al. For easiness of use, the metabolite variables inside the Promela programs are indicated by the 9 character code which identifies each species in the HN1 network, e.g. cytosolic oxalate = oxalate(c) = HC00195_c. For the newly inserted compounds we adopted a similar coding: peroxisomal oxalate = oxalate(p) = H2C0014_p. Similarly, the reactions are identified with the identification number adopted in HN1 (ranging from reaction number 1 to reaction number 2539), with the newly introduced ones that simply follow the numbering from the last used identifier (i.e. from 2539 to 2589). As an example, the detoxification reaction Alanine(p) + Glyoxylate(p) → Glycine(p) + Pyruvate(p) is indicated as:

$$\text{reaction } 2541: \ \text{H2C0010\_p} + \text{H2C0001\_p} \rightarrow \text{H2C0011\_p} + \text{HC00032\_p}$$

7.3 Model Verification and Simulation

The methodologies developed in this thesis work, and described in chapter 5, have been tested on the HepatoNet2 network. The following sections will address several important issues related with the reconstruction, simulation and verification of genome-scale

metabolic network models, with a special focus on understanding the Primary Hyperoxaluria type I disregulation. All the results have been obtained using the workflow described in chapter 6.

7.3.1 Validation of the Metabolic Network Model

The first feature of the network that we wanted to ascertain was the ability of the model to produce oxalate fulfilling all the known biological constraints, i.e. *i)* the accumulation of the oxalate in the cytosol compartment, *ii)* the transport of the glyoxylate from the peroxysome to the cytosol, and *iii)* the conversion of glyoxylate into glycolate and oxalate in the cytosol.

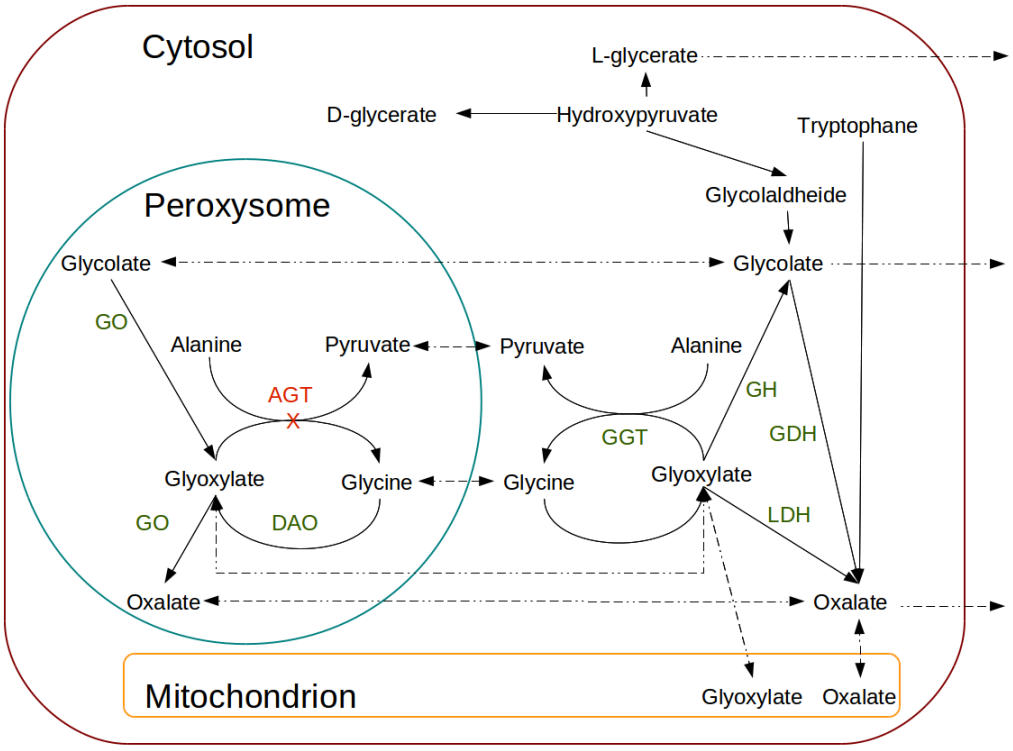

Fig. 7.2. An excerpt of the reactions and the metabolites contained in the HN2 model and directly involved in the oxalate synthesis. Colored lines enclose the compartments. Dotted lines represent transport reactions, whereas solid lines represent internal ones. The glyoxylate detoxification reaction, involved in the PH1 disease, is indicated by a red cross together with the name of the catalyzing enzyme AGT.

Model Specification

The network has several bi-directional reactions and a considerable size. Hence we decided to rely on the network pruning and direction selection techniques that we have described in subsections 5.3.5 and 2.3.4. The thermodynamic evaluation of the fluxes was performed considering $T = 37°C$ and using the values for $\Delta G^0_{r_j}$ reported in Gille et al. [156] to obtain the equilibrium constants. This approach was applied the HN1 model in order to compute flux distributions across the $l = 442$ different metabolic objectives listed in Supplementary Data 3.2 of the same paper. To further reduce the size of the resulting

network model, we decided to focus only on the compartments directly involved in the oxalate synthesis, namely the peroxysome and the cytosol. To this aim, we removed from the network all the reactions that were not containing at least one metabolite belonging to the peroxysome or the cytosol. The final number of used reactions was 952, and the metabolites 967. Moreover, we removed from the model all the transport reactions exchanging metabolites across the system boundaries, as explained in subsection 5.3.4.

The initial value for all the metabolite variables was fixed at 5 concentration unit, following the criterion described in subsection 5.3.4. However, it should be noted that the complete HN2 network contains several reactions involving very high stoichiometric coefficients. These *abstract reactions* have been inserted in HN1 to model the formation or degradation of large molecular complexes [156]. We decided to allow their presence in the network, but we did not take into account their stoichiometry when calculating the initial concentration of the metabolites.

Expected Behaviour Specification

As already outlined, the hallmark of PH1 is the high production of oxalate in the cytosolic compartment. Since we want to investigate if our hepatocyte network is able to express this phenotype (i.e. this behaviour), we extracted the behaviour in which the final cytosolic oxalate concentration is increased with respect to its initial value. To this aim, we use the negation of the property we are looking for, as explained in subsection 5.4.3. Since we are not interested in all the intermediate values, rather we want to know the states the network can reach at the end of the initial perturbation, we used a *timeout* instruction (see subsection 5.4.1) in combination with the assert clause. The property specification was inserted as a possible choice in the do cycle of the principal process, as explained in subsection 5.4.6.

It is worth to note that oxalate can be only produced from the network and can never be consumed, hence the timeout value corresponds to the maximum possible oxalate value along that computation.

Running Spin: Computational Issues.

The hepatocyte model obtained considering only the cytoplasm and peroxysome is still a very large one, even if we adopted all the possible strategies to reduce its size. Hence, it resulted very difficult to manage in terms both of memory and time needed to obtain the filtered behaviours.

Table 7.1. Number of counterexamples found and elapsed time for different Spin configurations are shown. MA indicates the minimal automaton representation. Simply increasing the available memory or the CPU number does not significantly increase the number of extracted counterexamples. The best results are obtained with swarm approach with the random ordering of the instructions and transitions.

Spin search configuration	CPUs	Available Memory	Elapsed Time	Counterexamples found (number)	Notes
Exhaustive	1	2 GB	1 hour	0	out of memory
Exhaustive	4	8 GB	12 hours	0	out of memory
Exhaustive with MA	4	8 GB	2 months	4	stopped
Approximated (bitstate)	4	8 GB	$>$ 2 days	6	
Approximated (swarm)	16	16 GB	1 hour	$\approx$ 200000	random ordering
Approximated (swarm)	16	32 GB	4 hours	$\approx$ 800000	random ordering

With an incremental approach, it has been run on machines of growing capabilities. The largest part of the results, however, were obtained on the Nirvana cluster at

Tigem. Different Spin verification configuration were used, ranging from the exhaustive default settings to the swarm approach [12]. The results, in terms of time and number of counterexamples obtained, are given in Table 7.1.

Two main aspects emerge: the first is that, of course, the amount of available memory dramatically influences the performances of Spin in terms of explored portion of the computation space. The second is that an exhaustive search approach is still out of reach for genome-scale metabolic network models. Moreover, since the instructions are executed in the order in which they are declared in the Promela program (see [12], an unlucky ordering of the instructions may affect also the approximated bitstate approach, as happens in the example of table 7.1, where more than two days of work resulted in few counterexamples found.

Hence, we decided to adopt the swarm approach that was designed exactly to address the verification of models with a very large state space. Indeed, using swarm and specifying the random instruction and transition ordering, (see chapter 6 for all the details) we were able to find, using the Nirvana cluster, a very huge amount of executions in few hours.

Swarm settings

The swarm search was tuned as follows (see chapter 6). In particular, a very large number of counterexamples were extracted for each of the executed search jobs. Each of the possible search modes are included only once. They include the normal search, the reversed instruction and transition ordering, and the random ordering searches.

Max search depth: 10000
Max trail: 50000
Cpu Number: 16
RAM amount: 32 Gigabytes
Time: 4 hours
Hash function range: 3 – 8
Search modes: 4 normal ordering, 4 random ordering, 4 reversed ordering.

Results

To investigate on the correctness of our model, we extracted from more than 800.000 counterexamples all the final metabolite concentrations and the *reaction usage* values as detailed in subsection 6.1.5. We also selected, for each final oxalate value, the reaction usages for the gyoxylate transport, the glyoxylate detoxification, and the cytosolic glycine production. Since different reaction usages may obtained for the same oxalate concentration, we decided to work with the average values.

The oxalate final concentration ranges from a minimum of 20 units to a maximum of 103. However, the absolute value is not enough to clearly understand what happened into the network. Instead, we have to consider how much oxalate is produced with respect to its maximum producible amount. We can estimate an upper bound for the oxalate production by evaluating the maximum producible amount of cytosolic glycolate and glycine. In fact, glycine (through glyoxylate) and glycolate are directly related to the oxalate production, see figure 7.1. It is worth noting that this maximum might be a bit overestimated, since it is very unlikely that in a cell all the glycine would be transformed to oxalate, being glycine an essential aminoacid used to accomplish several fundamental cellular functions (see Lehninger [70]). More precisely, we calculated for each oxalate value the mean usage value of all the reactions producing glycine or glycolate in the cytosol, or moving these compounds into the cytosol from an outside compartment. The considered reactions are listed hereafter. The letters in brackets indicate the metabolite compartment (c=cytosol, s=sinusoidal space, m=mitochondria):

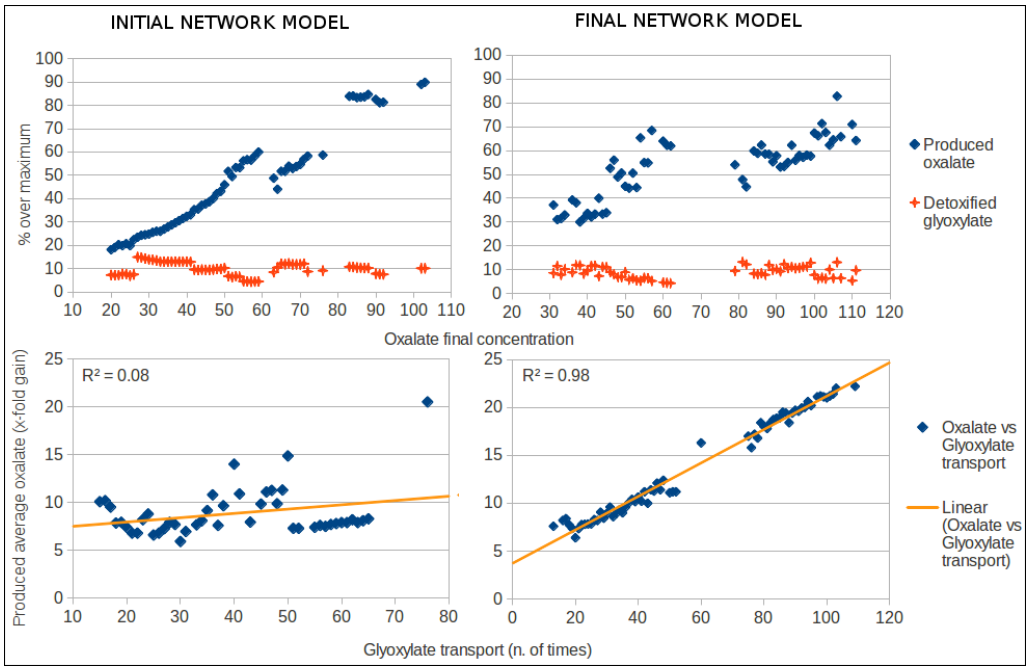

Fig. 7.3. Analyses of the extracted behaviours of cytosolic oxalate accumulation. The results referring to the initial network version (left column) and the modified final one (right column) are shown. The first row displays the produced oxalate and the detoxified glyoxylate both weighted on the maximum producible oxalate (calculated from glycolate and glycine). The second row shows the average usage of the transport reaction of glyoxylate from peroxysome to cytosol with respect to the oxalate production. The straight curve is a linear fit of the data. The coefficient of determination is also shown.

$$\mathrm{CoA}_{(c)} + \mathrm{Glycochenodeoxycholate}_{(c)} \longrightarrow \mathrm{Glycine}_{(c)} + \mathrm{Chenodeoxycholoyl{-}CoA}_{(c)}$$
$$\mathrm{CoA}_{(c)} + \mathrm{Glycocholate}_{(c)} \longrightarrow \mathrm{Glycine}_{(c)} + \mathrm{Choloyl{-}CoA}_{(c)}$$
$$\mathrm{Glycine}_{(m)} \longrightarrow \mathrm{Glycine}_{(c)}$$
$$\mathrm{Glycine}_{(s)} \longrightarrow \mathrm{Glycine}_{(c)}$$
$$\mathrm{Glycolaldehyde}_{(c)} + \mathrm{NAD}^{+}_{(c)} \longrightarrow 1\mathrm{Glycolate}_{(c)} + 2\mathrm{NADH}_{(c)}$$
$$\mathrm{Serine}_{(c)} + \mathrm{THF}_{(c)} \longrightarrow \mathrm{H_2O}_{(c)} + \mathrm{Glycine}_{(c)} + ({}_{51}0){-}\mathrm{Methylene{-}THF}_{(c)}$$

The obtained maximum is used to weight not only the oxalate production but also the usage of the detoxification reaction, which directly corresponds to the amount of potentially producible oxalate that has instead been detoxified.

The produced oxalate over the maximum can be as much as the 90% of the maximum producible, as shown in the first row, left column of figure 7.3. This is really interesting because it shows that the network is able to generate, starting from the same quantity of initial metabolite concentrations, very different amount of cytosolic oxalate. This should be due to the different usage of the detoxification reaction. At the same time we expect the glyoxylate transport reaction to be directly related to the oxalate production. However, if we investigate on the usage of these reactions, we find very different results, as shown in the left column of figure 7.3. The topmost figure shows the detoxified oxalate (also weighted over the estimated maximum) together with the relative oxalate production. The detoxification reaction is used a very low number of times, and there is no relation between the percentage of oxalate produced and the detoxified one. Furthermore, no relation between the glyoxylate transport and the oxalate production emerges as shown in the bottom left panel of figure 7.3. Here the oxalate gain, expressed as the x-fold increment with respect to the initial value, is shown in relation with the usage number

of the transport reaction. This is confirmed also by the very low value of the linear fit determination coefficient R^2, whose value is $R^2 \approx 0.08$.

We can immediately conclude that the network we are using is not a good model for the glyoxylate and oxalate metabolism: although it correctly simulates the oxalate accumulation, it fails in being consistent with the known biological evidences on the glyoxylate detoxification and transport. Hence, we focused on all the reactions producing or transporting oxalate, and revised the initial compiled model removing some reactions. The obtained model is shown in figure 7.4. The reactions modeling the displacement of

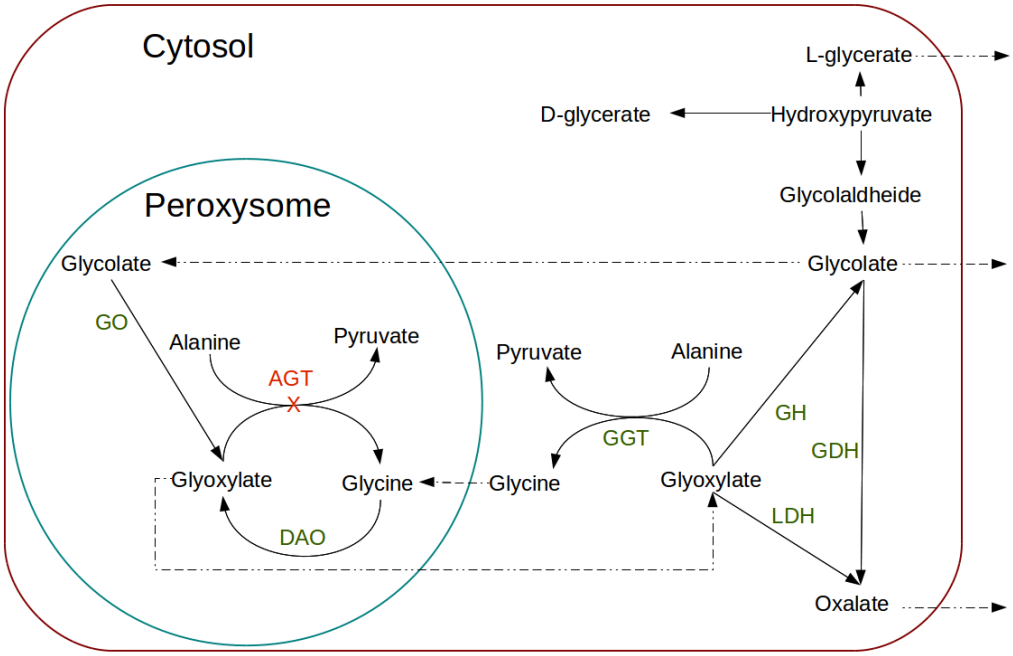

Fig. 7.4. The final network model of the peroxysome and cytosol reactions involved in PH1. As in figure 7.2 the colored lines enclose the compartments. Dotted lines represent transport reactions, whereas solid lines represent internal ones. The glyoxylate detoxification reaction, involved in the PH1 disease, is indicated by a red cross together with the name of the catalyzing enzyme AGT. Enzymes catalyzing other reactions are indicated in green.

oxalate and glyoxylate in the mitochondrial compartment have been proposed in HN1 (see the supplementary materials of [156]). However, as already said, HN1 does not model PH1 and since these transport reactions can not be found in the PH1 dedicated literature, we removed them from the model. The reaction accounting for the direct production of oxalate from tryptophan is proposed in some literature (see for instance [151]) but is still not widely accepted, and deserve further investigations before it can be properly included in the model. Lastly, the reaction modeling the direct conversion of glycine into glyoxylate was removed because we could not find biological evidences for it.

After these changes we started again with the FBA step, that resulted in a new pruning of the network. The new model was then analyzed again in search of the same behaviour. The results are shown in the right column of figure 7.3. The absolute amount of produced oxalate varies from 31 to 111, whereas relative oxalate production is only slightly lower than in the initial model. This is very likely due to distinct portions of space explored in the two models. Also in this case, starting from the same quantity of initial metabolite concentrations, the network is able to generate very different amount of cytosolic oxalate. This is due to the different usage of the detoxification reaction, as can be

seen from the upper right image of figure 7.3, but also to the different fate of the involved metabolites. This is really interesting, since the disease may show different levels of severity related to the oxalate production and concentration in blood and urine [154]. It is also interesting to note that the maximum produced oxalate with respect to glycine is the 85%. This is consistent with the expected behaviour of a real hepatocyte, where glycine is normally involved in a large variety of reactions and not only in glyoxylate/oxalate production [157].

For what concerns the relation between the detoxification reaction, it is not easy to see at a glance what is happening, because the searched space results in a very fragmentary distribution of the final values (topmost panel of the right column of figure 7.3). This effect was present also in the first network (where some "clusters" are clearly visible in the topmost left panel of figure 7.3), but the overall effect was much easier to interpret. Here we have few point per "clusters" and we have to investigate on the single groups. However, looking at the values we find that almost everywhere there is an inverse correlation, as expected. Furthermore the glyoxylate transport and oxalate production are very nicely correlated, with $R^2 \approx 0.98$, as can be deduced from the linear fit and the data shown in the bottom right panel and right column of figure 7.3).

This final metabolic network model exhibits the behaviours we are expecting from the actual biological knowledge, and will be the one used for all the following investigations. We will refer to it as HepatoNet3 (HN3).

7.3.2 Model Comparison: Wild Type vs Loss of Function

Once defined the correct network, we decided to investigate the differences between the model containing the detoxification reaction, the so called Wild Type (WT) model and the one in which this reaction is not functional, i.e. the Loss of Function (LoF) one.

Model Specification

We started from the HN3 model obtained in the previous section. For the WT model, the network pruning was calculated as described in subsection 7.3.1. Instead, in order to simulate the effect of a loss-of-function mutation of an enzyme regulating a reaction r_j, the same flux-balance problems described in subsection 2.3.4 were solved by constraining the fluxes through v_j to zero, that is, $v_j^{(+)} = v_j^{(-)} = 0$. In both cases the reactions carrying a zero flux were omitted, and only the cytosol and peroxisome compartments were considered.

The resulting network models are slightly different: the WT model contains 915 reactions and 951 compounds, whereas the LoF one contains 911 reactions and 950 compounds. Furthermore, they differs in terms of included/excluded reactions and compounds, and in terms of reaction directions. Also in this case, the initial values for all the metabolite variables was fixed at 5 concentration unit.

Expected Behaviour Specification

We are interested in investigating the production of oxalate in the cytosolic compartment, both for the WT and the LoF model. Hence, we use the same approach as in subsection 7.3.1, i.e. we define the negation of the property we are looking for, and we combine the resulting assert clause with a *timeout* instruction inserted in the do cycle of the principal process. Furthermore, to better understand the differences between the two models, other steps were performed to obtain the maximum and minimum oxalate final concentration values for both the models, with the invariant specifications expressed as explained in subsection 5.4.3.

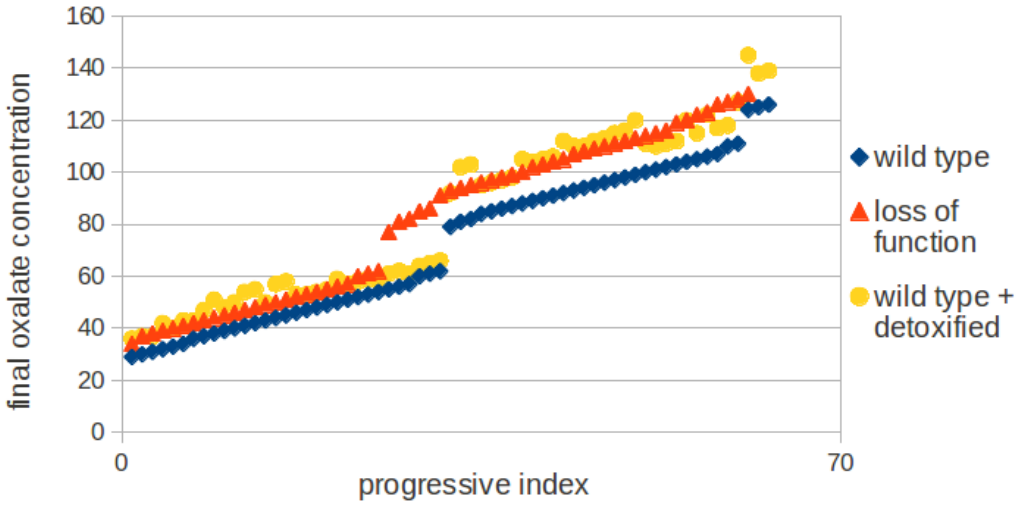

Fig. 7.5. The absolute final oxalate concentration value is shown for the wild type model and for the loss-of-function one. To show the effects of the glyoxylate detoxification, a final value is shown, obtained adding the amount of the detoxified glyoxylate to the wild type concentrations. The results are ordered from the minimum produced oxalate to the maximum (progressive index).

Swarm settings

The swarm search was tuned as follows (see chapter 6). Here the number of extracted counterexamples per job was lower than before, but a larger number of search modes was specified. The aim is to investigate the larger possible part of the state space using several search strategies, especially the ones with random ordering. Moreover, since we are searching also for maxima, we allowed a higher limit for the depth search.

Max search depth: 20000
Max trail: 200
Cpu Number: 16
RAM amount: 32 Gigabytes
Time: 1 day
Hash function range: 3 – 8
Search modes: 4 normal ordering, 20 random ordering, 4 reversed ordering.

Results

The final oxalate concentration values (including minimum and maximum) for both the WT and LoF models are shown in figure 7.5. The results, ordered by increasing values, clearly show that in the LoF model the final concentrations, as well as the minimum and maximum, are higher than in the WT model. This is due to the presence of the detoxification reactions: when the detoxified amount is added to the final WT oxalate concentrations, the obtained values are higher than the LoF value. It is worth noting that in the WT filtered disease behaviour the detoxification reaction always takes place, even in those cases where the final oxalate production is very high. The percentage of produced oxalate for both the WT and LoF models is shown in figure 7.6. In this case, especially for low oxalate productions, the WT model results in higher percentages than the LoF. This is related to the new network pruning induced by the removal of the detoxification reaction. In fact, in the WT model a relevant portion of the glycine (up

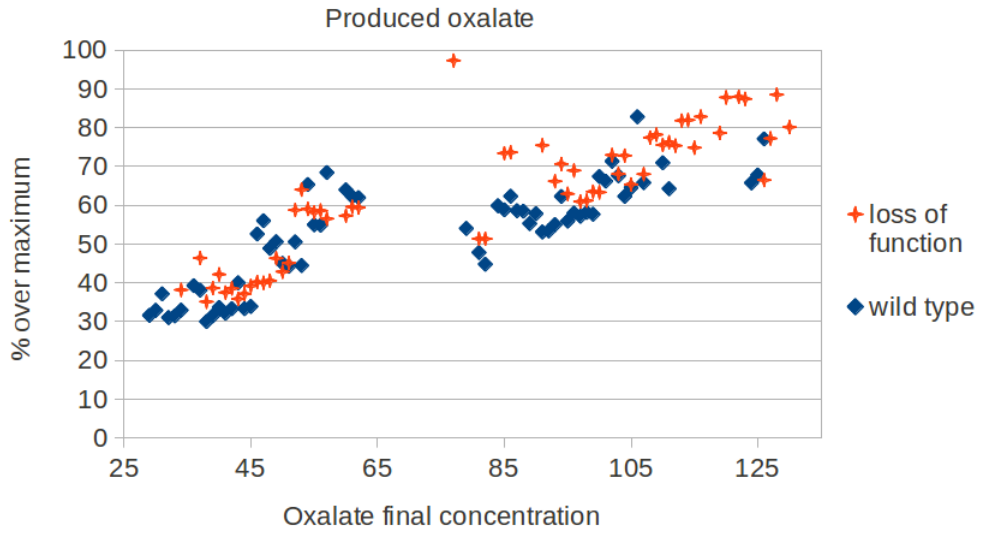

Fig. 7.6. The produced oxalate is shown for both the wild type and loss of function models. The final concentrations are weighted on the maximum producible oxalate (calculated from glycolate and glycine).

to 21 concentration unit) is consumed in the cytosolic reaction Glycine + Cysteine $\longrightarrow$ H_2O + CysGly, thus reducing the total amount of glycine that may be converted into oxalate, and consequently lowering the maximum producible oxalate that is used to weight the obtained data. The above reaction carries a zero flux int the LoF model, and hence was removed in the pruning step. Moreover, in the LoF model two more reactions that move glycine from an outside compartment to the cytosol are present, whereas they are not included in the WT model (since they have null fluxes).

Nonetheless, when very high amounts of oxalate are produced this difference is not so relevant. In particular, it is possible to observe that *i)* the maximal value of oxalate production is reached in the LoF model, and *ii)* the LoF model results in higher percentages than the WT, for high oxalate productions. These results agree with the PH1 expected phenotype [154].

7.3.3 Satisfaction of Metabolic Objectives

Another useful investigation involves the ability of the network to fulfill some metabolic objectives. A metabolic objective (MO) is a cellular function that should be accomplished by the biological system using as input a subset of compounds. Typically a metabolic objective is specified as the combined production and consumption of some specific metabolites given the input ones.

Here, we focused on a small subset (containing 50 functions) of the metabolic objectives that have been used in Gille et al. [156] to simulate on HepatoNet1 different physiological hepatocyte functions. More precisely, our reduced set was obtained considering only the MO in which there is an oxalate production, i.e. the fluxes through the oxalate producing reactions are different from zero *and* the specified MO was fulfilled. We will refer to this set as the *Oxa_MO* set.

Each MO is usually characterized by an *Objective* indicating the metabolite that should be produced (or consumed) by the network, and by a set of *Constraints* containing the metabolites that the network should consume and produce to fulfill the MO. Table 7.2 shows a minimal subset, taken from the Oxa_MO set, of MO together with their Objective and Constraints. The MES, DES, and MIPES are special sets of constraints, described in Gille et al. [156]. Both for the Objective and the Constraints a minus sign (-)

indicates the consumption of a metabolite, a plus sign (+) its production and an equal sign (=) the initial availability of a metabolite, that can be both consumed and produced by the network.

Table 7.2. Some examples of Metabolic Objectives taken by the Oxa_MO set. The *Simulation ID* describes the expected target of the MO. The *Objective* is the metabolite that should be produced or consumed by the network while fulfilling the limits listed in the *Constraints* column. Minus (-) signs indicates the consumption of a metabolite, whereas a plus sign (+) indicates its production. An equal (=) sign indicates that the metabolite may be both consumed and produced by the network. The letter in brackets indicate the metabolite compound. MES, DES, and MIPES are predefined sets of constraints described in [156].

Simulation ID	Objective	Constraints
ATP(s) production	+ATP(s)	MES
Uridine(s) waste	-1 Uridine(s)	DES
ATP(c) salvage from Hypoxanthine	+ATP(c)	-Hypoxanthine(c) -Glucose(s) -Pi(s) -Glutamine(s) +CO2(s) -O2(s) =H2O(s)
Glycine degradation	-1 Glycine(c)	-Palmitate(s) -O2(s) +ATP(c) -ADP(c) -Pi(c) =H2O(s) +Urea(s) -Methionine(s) +H2S(s) +CO2(s)
Synthesis of stearate	+Stearate(c)	MIPES

These information will be exploited to build both the model and the expected behaviour specifications. The ability of the network to fulfill the MO contained in the Oxa_MO set will be tested both for the WT and the LoF model. The aim is to investigate on the effect of the chosen pruning on the obtained network, and on the differences among the wild type and diseased (LoF) models.

Model Specification

We used the pruning already defined in subsection 7.3.2 both for the WT and the LoF models. However, since MO are defined on the whole network, we included all the compartments, discarding only the zero-flux reactions. This pruning resulted in 951 compounds and 1153 reactions for the WT model and 950 compounds e 1150 reaction for the LoF one. However, as will be explained in detail in the results, the network was able to fulfill only a small subset of the Oxa_MO objectives. Hence, to understand if this behaviour was an effect of not considering the zero-flux reactions, we tested the ATP production MO (which is the first in table 7.2 and the one producing more oxalate among all the MO contained in the Oxa_MO set) on the whole network. We decided to establish the zero-flux reaction directions using the information contained in the HN1 model: if a reaction r_i was considered mono-directional in HN1, we set its direction in HN3 accordingly. Otherwise, a random choice should be made (although this was not the case, since the information from HN1 and HN3 together were enough to define all the directions). The whole network contains 2576 reactions and 1445 metabolites.

Since the constraints specify which metabolite should produced and which consumed, we decided to set the initial value of the network metabolites as follows: let C be the set of constraints of a MO, and O the set of corresponding Objectives. Let be m_i a metabolite and v_i its initial value, where i ranges from 1 to the number of metabolites contained in the network. Hence:

$$
\begin{aligned}
&\text{if } m_i \in O \text{ and } \text{sign}(m_i) : + \Rightarrow v_i = 5 \\
&\text{if } m_i \in O \text{ and } \text{sign}(m_i) : - \Rightarrow v_i = 30 \\
&\text{if } m_i \in C \text{ and } \text{sign}(m_i) : + \Rightarrow v_i = 5 \\
&\text{if } m_i \in C \text{ and } \text{sign}(m_i) : - \Rightarrow v_i = 30 \\
&\text{if } m_i \in C \text{ and } \text{sign}(m_i) := \Rightarrow v_i = 30 \\
&\text{if } m_i \notin C \text{ and } m_i \notin O \quad \Rightarrow v_i = 5
\end{aligned}
$$

Since the metabolic network needs at least a minimum concentration of metabolites to work, we decided to set to 5 concentration unit all the initial values of the metabolites that are either not involved in the MO or are involved, but should be only produced. Instead, the metabolites that should be consumed (the ones with the - sign) or may be also consumed (the ones with the = sign) are set to 30. Please do note that for each MO contained in the Oxa_MO set a different Promela program was built and verified.

Expected Behaviour Specification

Each MO was specified as described in subsection 5.4.3. However, we decide to exclude from the assert clause all the metabolites in the constraints having an = sign specification: in fact, these species may be both produced and consumed, hence we can not specify the direction of their changing. The assert clause was used either in combination with a timeout clause, either alone, since the timoeout is a very strong requirement for a biological system.

```
/* ATP salvage from Hypoxanthine  */

/* EXPECTED MO: +HC00012_c -HC00238_c -HC00040_s -HC00019_s -HC00067_s
 +HC00021_s -HC00017_s =HC00011_s  */

/* THE METABOLITE VARIABLES */
 ....
 short HC00011_s = 30; /* NAME: Glycine(s) */
 short HC00012_c = 5;  /* NAME: ATP(c) */
 short HC00017_s = 30; /* NAME: O2(s) */
 short HC00019_s = 30; /* NAME: Pi(s) */
 short HC00021_s = 5;  /* NAME: CO2(s) */
 short HC00040_s = 30  /* NAME: Glucose(s) */
 short HC00067_s = 30; /* NAME: Glutamine(s) */
 short HC00238_c = 30; /* NAME: Hypoxanthine(c) */
 ....

/* THE REACTION PROCESS */
   proctype REACT_process() {
     do
      .... /* all the network reactions*/
      :: timeout  →   assert( (HC00012_c <= 5) || (HC00238_c >= 30) || (HC00040_s >= 30)
                                        || (HC00019_s >= 30) || (HC00067_s >= 30)  || (HC00021_s <= 5)
                                        || (HC00017_s >= 30));
     od
   }

/* START THE PROCESS  */
      init{run REACT_process()}
```

Promela code snippet 12: An excerpt of the Promela program implementing the timeout verification of the MO *"ATP(c) salvage from Hypoxanthine"*.

The resulting Promela program for the MO *ATP(c) salvage from Hypoxanthine* is shown in the Promela code snippet 12. The other 49 MO contained in the Oxa_MO set have been similarly implemented.

Swarm settings

The swarm search was tuned as before, except for the max search depth that was incremented. Also, only a small number of counterexamples were considered.

Max search depth: 30000
Max trail: 2
Cpu Number: 16
RAM amount: 32 Gigabytes
Time: 4 hours
Hash function range: 3 – 8
Search modes: 4 normal ordering, 20 random ordering, 4 reversed ordering.

Results

The results of the timeout verification were all negative: the network was not able to fulfill in the final state any of the required objects, both in the WT and in the LoF models. Without the timeout clause some result is obtained, as shown in table 7.3. Apart

Table 7.3. List of the MO that the WT and LoF models are able to satisfy. A check mark indicates that the corresponding MO has been fulfilled in the specified network. These results have been obtained without the timeout clause.

Simulation ID	WT	LoF
CTP(s) waste	✓	
Uridine(s) waste		✓
CDP-diacylglycerol-VLDL-PI-pool(l) production	✓	
ATP salvage from Hypoxanthine	✓	✓
Glutamate degradation	✓	✓
Glycine degradation	✓	✓
Histidine degradation	✓	✓
Glutamine degradation	✓	✓
Leucine degradation	✓	✓
Lysine degradation	✓	✓
Proline degradation	✓	✓
Serine degradation	✓	✓
Tryptophan degradation	✓	✓
Valine degradation	✓	✓
Homocysteine degradation	✓	✓
Ornithine degradation	✓	✓

from the first three MO, all the others are fulfilled both from the WT and from the LoF model. However, it is a very small amount (16 MO on 50, to be precise) and this is rather unexpected considering that the verification is not performed at timeout. To exclude any influence due to the omission of the zero-flux reactions, we verified the first MO of table 7.2 using the whole network with all the reactions and the assert clause without the timeout. Also in this case, the result was negative, both for the WT and the LoF model.

This is very likely due to the consequences of the mean-fluxes pruning strategy (see subsection 2.3.4). In fact, fluxes are calculated in each of the FBA problems used to satisfy a MO. However, the mean of the fluxes among **all** the MO is used to prune the network: this corresponds to consider only the mean behaviour of the network. This is a very good strategy when exploring the network for less constraining properties, but may be not appropriate when working on a single metabolic objective. Since a different Promela program is built for each MO, it might be more useful to determine the network pruning using only the fluxes resulting from the FBA step performed for the corresponding MO, rather than considering always the same mean-flux pruning for all of them. We will explore this direction in future works.

Nonetheless, it is interesting to note that some MO objectives are still conserved into both the WT and LoF networks obtained with the mean-fluxes approach: these MO are

mainly related with the amino acids catabolism, which includes essential functions that take place almost exclusively in the liver cell [70].

7.3.4 Monotonic Behaviours

An important feature of a metabolic network is its ability to only produce or only consume some metabolites or, complementary, to both produce and consume some of them. We will call "initial" a metabolite that is monotonically consumed by the network, and "final" one that is monotonically produced. Instead, we will call "intermediate" a metabolite that is both produced and consumed by the network. After extracting the information from both the WT and LoF networks, a comparative analysis may be performed that can be useful in revealing similarities and differences between the two models.

Model Specification

Similarly to what we have done in subsection 7.3.3, we considered the whole network discarding only the zero-flux reactions, both for the WT and the LoF model. In this case, the initial value for each metabolite variable was fixed at 5 concentration unit.

Expected Behaviour Specification

The property specification in this case was based on a LTL always formula, as explained in subsection 5.4.4. To assign a metabolite m_i to one of the above defined classes, we proceed as follows:

1. a verification of m_i being monotonically increasing is performed;
2. a verification of m_i being monotonically decreasing is performed;
3. a) if a counterexample is found both for the increasing and decreasing verification steps, then m_i is "intermediate";
   b) if a counterexample is found only for the increasing verification step, then m_i is "initial";
   c) if a counterexample is found only for the decreasing verification step, then m_i is "final";
   d) if a counterexample is not found neither for the decreasing verification step neither for the increasing one, then m_i is considered as "never used".

Please do note that this approach implies that two verification steps should be performed for each of the 951 WT and each of the 950 LoF compounds, for a total of 3802 generated and verified Promela programs. Since the monotony increasing (decreasing) property is checked placing the asserts only immediately after the reactions in which the compound is consumed (produced), see subsection 5.4.4, if the pruning does not include consuming (producing) reactions, then is useless to start the program verification. Rather, the property is considered to be banally fulfilled and the metabolite m_i is assigned to the "final" ("initial") or to the "never used" class depending on the results of the complementary step.

However only 114 out of 3802 compounds fell in this case, and all the remaining Promela programs had to be executed.

Swarm settings

The swarm search here is aimed at finding a single counterexample, in the lower possible time. The number of CPU and the assigned RAM were set to values lower than before due to the large number of Promela program to execute.

Max search depth: 35000

Max trail: 1
Cpu Number: 4
RAM amount: 16 Gigabytes
Time: 2 hours
Hash function range: 3 – 8
Search modes: 12 random ordering

Results

Despite the very high number of Promela programs to elaborate, the specification of properties as LTL formulae allowed to speed up the whole verification process, that terminated in about a month. The results are shown in tables 7.4 and 7.5. In table 7.4

Table 7.4. Results of the classification for the WT and LoF models: the number of compounds falling in each category is shown, together with the total number of compounds belonging to each model.

	Initial	Final	Intermediate	Never used	Total number
WT	90	51	800	10	951
LoF	90	52	798	10	950

the number of compounds falling into each category is shown. Despite the differences in the pruning, the compound classification is quite the same for both the models. The only two differences are due to the compounds dCTP(c), that has a different classification in the two models, see table 7.5 and Cys-Gly(c) that is only present in the WT model (due to the pruning). A small number of compounds is never used: this happens to the species that

Table 7.5. A list of some interesting metabolites, together with their classification as *intermediate*, *final*,*initial* or *never used*, for both the WT and LoF models.

Metabolite Name	WT Classification	LoF Classification
Cys-Gly(c)	intermediate	—
dCTP(c)	intermediate	final
Oxalate(c)	final	final
Bilirubin(c)	intermediate	intermediate
Bilirubin(s)	initial	initial
Bilirubin-bisglucuronoside(b)	final	final
Cholesterol(b)	final	final
Urea(s)	final	final
Tryptophan(s)	initial	initial
Isoleucine(s)	initial	initial
Arginine(m)	never used	never used
Cholesterol-ester-pool(l)	never used	never used

are involved only in the so-called *abstract reactions* (see subsection 7.3). Some of them are listed in table 7.5. Oxalate is considered a final compound, in perfect agreement with the biological knowledge. Several other interesting information can be deduced from this list: for instance bilirubin is imported from the sinusoidal space (Bilirubin(s) is initial) then it moves into the cytoplasm and there is converted (Bilirubin(c) is indeed intermediate) to the water-soluble compound Bilirubin-bisglucuronoside(b) that is a final product found in the bile canaliculi. Similarly, urea is a final product released into the sinusoidal space.

Lastly, several amino acids in the sinusoidal spaces are initial, and will be then moved to other compartments (in which they results as intermediate) to perform a wide range of hepatic functions. These behaviours are in agreement with the known liver functions, thus demonstrating the validity of the model and the usefulness of the method.

7.4 Exploiting the Dynamics

Until now we have used only the information on the used reactions and the final values of the metabolites. However, the methodology developed in this thesis may be used to investigate on the temporal features of the changes in metabolite concentrations over time. The behaviours representing the maximum and minimum producible oxalate were

Table 7.6. Final concentration values (in generic units) of five compounds involved in PH1 for both the WT and the LoF models, and for the maximum and minimum producible oxalate properties. The minimum final value, the maximum final value, and the mean of the final value across the extracted counterexamples together with its dispersion are shown.

Property	Model type	Compound Name	Minimum final value	Maximum final value	Mean ± dispersion
Maximum oxalate	WT	oxalate(c)	230	248	233.07 ± 5.16
		glycolate(c)	0	0	0 ± 0
		glyoxylate(c)	0	0	0 ± 0
		glycine(p)	0	5	0.49 ± 1.08
		glyoxylate(p)	0	0	0 ± 0
	LoF	oxalate(c)	230	263	235.7 ± 7.74
		glycolate(c)	0	0	0 ± 0
		glyoxylate(c)	0	0	0 ± 0
		glycine(p)	0	28	3.89 ± 9.28
		glyoxylate(p)	0	0	0 ± 0
Minimum oxalate	WT	oxalate(c)	27	41	36.8 ± 3.55
		glycolate(c)	0	0	0 ± 0
		glyoxylate(c)	0	0	0 ± 0
		glycine(p)	116	203	160.56 ± 16.94
		glyoxylate(p)	0	0	0 ± 0
	LoF	oxalate(c)	40	41	40.67 ± 0.48
		glycolate(c)	0	0	0 ± 0
		glyoxylate(c)	0	0	0 ± 0
		glycine(p)	143	172	168.5 ± 3.54
		glyoxylate(p)	0	0	0 ± 0

extracted at timeout for both the WT and LoF models (as described in 5.4.3. About 350 counterexamples were collected for the maximum WT, 500 for the maximum LoF, 800 for the minimum WT, 200 for the minimum LoF.

In table 7.6 some information on the final values of five metabolites relevant for the PH1 disease are shown. However, while for oxalate and peroxisomal glycine some information may be immediately extracted from the table, for all other compounds not so much information is obtained nor from the final values nor from the mean and dispersion values. For instance, the peroxisomal glycine has very low initial and final values in the maximum, both for the WT and LoF models. However, from these file we can not infer its behaviour between the initial and final values.

In figure 7.7 the mean dynamics (across all the extracted counterexamples) of the same five compounds are shown. together with the dispersion measure (i.e. the variance) that represents the variability of a metabolite concentration over the time steps. Since the

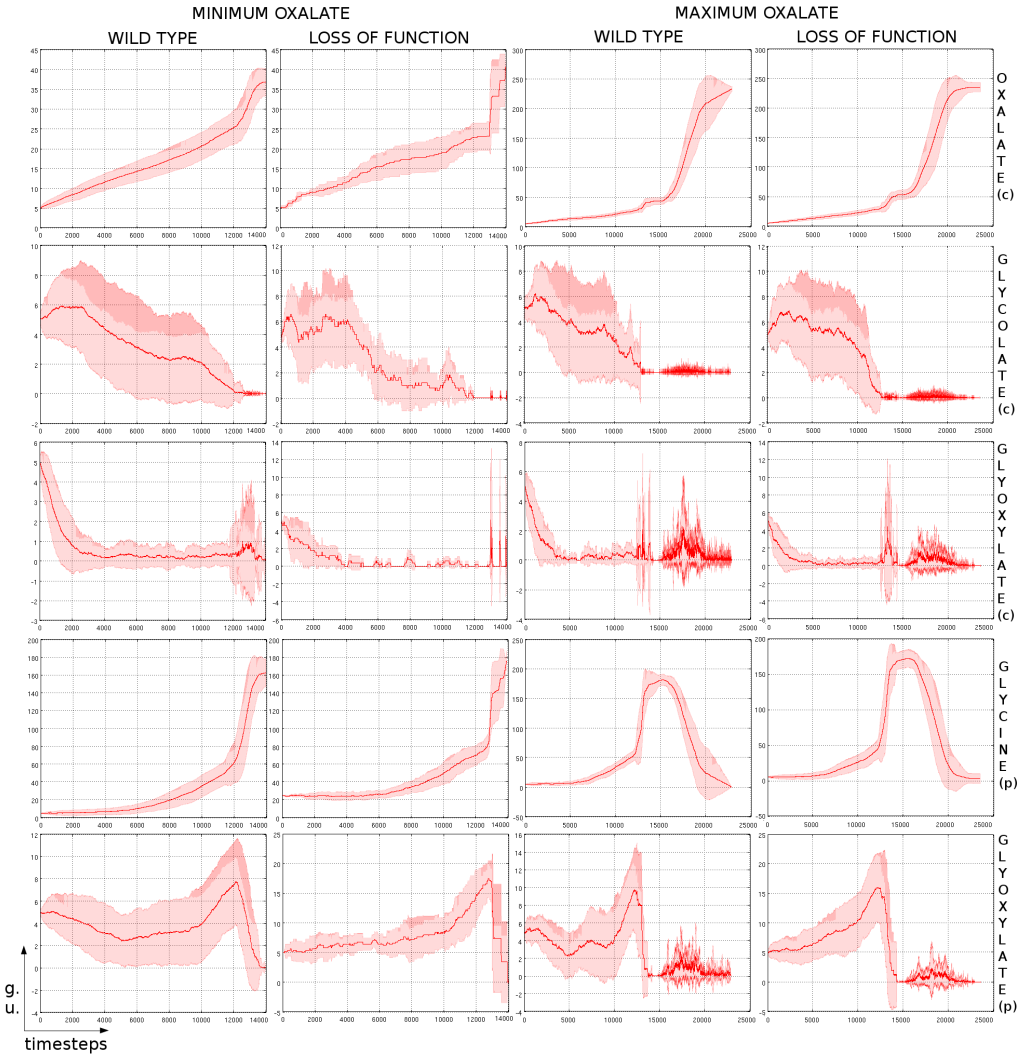

Fig. 7.7. The mean dynamics of five compounds are shown (red lines), together with their dispersion measures (shadowed areas). Mean and dispersion of the metabolite concentrations were calculated across several counterexamples obtained for the maximum and minimum producible oxalate behaviours, both for the WT and the LoF models. The concentrations (y axes) are expressed as usual in generic units, and the time steps (x axes) are given by the sequences of the elaboration steps.

single reactions may be in principle executed in very distant time steps, unless they are indispensable for other reactions to take place, this dispersion measure is only indicative on how much constrained is the dynamic of a metabolite over the time. For instance, in the first row of figure 7.7 the oxalate dynamics are shown for both the models and the behaviours. The dispersion of this metabolite is quite small (as the shadowed area indicates), since it increases monotonically during time, and the requested behaviours filter out only the traces in which oxalate is accumulating. Instead, for citosolic glycolate (second row) the dispersion measure indicates that the consumption of this metabolite may take place at very different time steps. Since the dynamics of each compound may have different lengths (i.e. different timings) in distinct counterexamples, to permit their comparison the maximum possible time step was determined across all the counterexamples, and the dynamics was extended setting the newly added point to the final value reached by the metabolite.

Is interesting to note that the WT and LoF models present very similar dynamics for all the metabolites, both for the minimum and for the maximum behaviour. This is expected, since we are extracting from both the models the behaviours in which oxalate is overproduced. In fact, we are extracting from the WT model the behaviours corresponding to the PH1: indeed the WT model contains both the behaviours corresponding to the disease (obtained when the AGT reaction takes place few times or is not executed at all) and the "normal" ones, where the detoxification reaction functionality is untouched. However, the presence of the detoxification reaction in the WT model influences the levels of both the minimum and maximum oxalate obtained, that is always lower than the LoF counterpart (WT=27 g.u., Lof=41 g.u.) and the maximum (WT=248 g.u., LoF=263 g.u.).

On the contrary, differences arise when comparing the minimum and maximum behaviours, for both the models: for instance the peroxisomal glycine (fourth row) has an accumulating behaviour in the minima, and falls down to zero when the maximum oxalate is searched. Similarly, the dynamics of citosolic glycolate and glyoxylate (second and third row), and of peroxisomal glyoxylate (fifth row) show that these three compounds are rapidly consumed in the minima, whereas they oscillate a lot in the maxima.

Concluding, the dynamics are a very useful tool to reason about the metabolite behaviours, and to infer qualitative properties of the model at study.

7.5 Chapter summary

In this Chapter we tested our methodology on a genome–scale metabolic network model of human hepatocyte. In particular we addressed the study of a monogenic disorder called Primary Hyperoxaluria Type 1 (PH1). The functional deficiency of AGT results in a failure to detoxify glyoxylate within the peroxisomes of the liver cells. Instead of being transaminated to glycine, glyoxylate is oxidized to oxalate and/or reduced to glycolate, resulting in greatly increased urinary excretion of oxalate and glycolate (that may be directly converted to oxalate). Oxalate in mammals can not be further metabolized and its overproduction results in the deposition of calcium oxalate (CaOx) crystals and stone formation almost everywhere in the body. We performed different test on the hepatocyte network. The first was aimed at verifying the congruence of the model with the known biologic evidences of the disease. Here we demonstrated that the model was not compliant with the expected behaviours, and was hence revised and corrected. After we compared the wild type (WT) model with a Loss Of Function (LoF) one, extracting relevant information from the obtained results. Afterwards, we used the developed techniques to verify the satisfaction of a set of Metabolic Objectives. Lastly we performed an analysis to determine the initial, final and intermediate metabolite of both the WT and LoF models. In all of the tests, the results of a FBA analysis step were considered to both decide directions and lower the model size. The last section described how the results of the dynamics may be exploited to extract interesting qualitative knowledge about the temporal behaviour of the underlying network model.

8

Conclusions

Biological network models are crucial in the context of Systems Biology. Their reconstruction, propelled by the availability of large amounts of genome-scale *omics* data coming from high–throughput sequencing technologies, is challenging as much as their analysis and simulation.

However, the simple organization of data into networks and the study of their topological properties are not enough to gain insight on the features of the system as a whole. A system-level understanding of an organism should address both the system structure (i.e. its components and the way in which they are related), and the system dynamics (i.e. how it behaves over time under various conditions). This is an essential step to unravel the relationships intercurring between the genotype and the phenotype of living organisms.

To this aim, the biological information should be necessarily complemented by mathematical and/or computational models that may be executed or simulated to reveal the adequacy of the biological assumptions and hypotheses on which they were based, in case leading to a refinement of the model and/or to new biological experiments. This hypothesis-driven research approach, together with the cyclic process of information integration and *in silico* model building, is essential in the Systems Biology paradigm.

Mathematical approaches are limited by the large size that biological networks typically exhibit, and by the availability of precise biological data to build the model. Computational models may go beyond these limitations and work also on partial data, and on more complex systems with a qualitative approach that relies on the goodness of the underlying abstraction, rather than on the faithfulness of the mathematical implementation. The obtained results, although not precise, may be essential in unraveling the complexity of the modeled biological systems. The differences between the computational and the biological models may be used to build new hypotheses, to refine the model, and even suggest new experiments that can help in validate or reject the model. This **executable biology** methodology is based on the close interplay between *in silico* simulations and biological validation of the data.

Model checking techniques may be successfully used to built executable models of biological systems, since their formal languages allow to describe both the constituent components and the way in which they interact, with the overall behaviour emerging from the interplay of the components, in a perspective that is very similar to the Systems Biology paradigm. Moreover, these formal techniques enforce abstraction, an essential feature for dealing with large models. Lastly, they can give partial information on a biological systems through simulations of the model, i.e. a partial exploration of the execution space.

In this thesis we proposed the use of model checking techniques for building, simulating and verifying executable models of genome–scale metabolic networks. Several original contributions were given, and in particular:

- we proposed a strategy to translate a metabolic network into an executable model on which the checking phase can be performed. This approach is based on considering the reactions as parallel running processes that try to access concurrently to shared variables representing metabolite concentrations. Reactions may take place only if all the needed metabolites are available. To select between different executable reactions a non–determinism choice was considered;
- we identified interesting biological properties that can be checked on a metabolic network model once expressed in a suitable property specification language;
- we devised a non–standard use of the model checker as a *behaviour filter*, i.e. as generator of *witnesses* confirming that a model is able to exhibit a certain property. Using the negation of the property of interest, the model may be queried to extract relevant information on the executions (i.e. the used reactions and the metabolite concentrations) that led to the satisfaction of the required property;
- we choose a model checker to implement the above described operations. In particular, we selected Spin (instead of NuSMV) since its model specification language, Promela, allowed a very simple translation of a metabolic network into an executable model. Moreover Spin permits the generation of more than a single counterexample, condition that is essential to our behaviour filtering strategy. Lastly, Spin allows for approximated search strategies, that were widely used in this thesis.
- we presented abstraction strategies, both for the model and for the property specification phases, that are essential in reducing the complexity of the overall verification/behaviour filtering process;
- we proposed a novel approach in which the results of a Flux Balance Analysis step are used to decide the reaction directions or to prune the network obtaining a smaller model, in a biologically sound manner;
- we designed control strategies which permit to steer the checking process only on interesting portions of the computation space;
- we discussed in detail the strengths and limits of the proposed methodology, addressing also the problem of the results coming from approximated searches;
- we implemented a complete workflow, based on Perl modules and *bash* scripting, that allows to import a metabolic network, either in SBML format or as a stoichiometric matrix, and translate it into a Promela model, that can be executed using the model checker Spin. The extracted counterexamples are processed, and useful information are extracted from them and saved in textual files. Moreover, we discussed how the parameter tuning of the approximated searches may influence the obtained results;
- we tested this novel methodology on a real biological model, i.e. a human hepatocyte metabolic network. In particular, we focused on a mendelian disease called Primary Hyperoxaluria Type I, showing that interesting information may be deduced from the application of the proposed methodology.

8.1 Perspectives

Although the work here presented is complete and self–contained, still other things may be done in the future. We will give here a brief description of the possible directions that in our opinion may deserve further efforts.

At first, we are planning to further investigate the hepatocyte model of chapter 7 to elucidate the pathway that links the oxalate production to the tryptophan metabolism. The existence of a relation between this essential aminoacid and the oxalate metabolism has been postulated in some literature, but is still not completely understood. Hence we plan to modify the metabolic network model, extending it with all the reactions supposed to be in the pathway. After, the new model will be explored and simulated, and the resulting data compared with the known biological evidences of the PH1 disease.

At the same time, we want to further improve our methodology. In fact, we are planning to implement new control strategies. We would like to introduce enzimatic

control by modeling priorities among the executable reactions. A simple mechanism based on priority levels has been already implemented, as described in chapter 5. However, in the proposed approach, if a reaction with higher priority may be executed, the one at lower priority will be always skipped. We would like to change this behaviour, permitting the execution of the lower priority reactions in a rate that should be based on the Michaelis K_M enzymatic constant. This will have two beneficial effects: first, it would permit a more fine-grained representation of the modeled system. Second, it will allow for the presence of interesting bidirectional reactions, whose execution will be regulated by the strenght of the correspoding catalizing enzymes;

Another direction we would like to explore takes into account the introduction in the network model of a process producing metabolites at finite rates. This would permit to study the network taking into account a more dynamical representation of the intake/export fluxes within the external environment. Of course, this will also imply the definition of more complex properties than the simple reachability ones we have already implemented here.

Lastly, we would like to design a user–friendly interface that should permit a smooth use of the methodology also to people that are not skilled in formal techniques. In particular, we are thinking to develop a meta-language that should permit an easy definition of the properties to check on the metabolic network; moreover a database may be beneficial to store and make more easily accessible the extracted data, and it will provide a knowledge base on top of which it will be possible to build more complex data mining procedures.

Acknowledgments

After three years of Ph.D. thesis I would like to thank many people: my tutor Adriano Peron, that believed in the feasibility of this project, and shared this adventure (and some Annapurna's wheats) with me; my co-tutor Diego di Bernardo, that hosted me at TIGEM, gaving me the unique opportunity to test my programs on the Nirvana cluster, and my *macarons* on him; Roberto Pagliarini, that always encouraged me with good advices and good rice; Lele Castello that always answered to my stupid question on important biological problems; the "Auletta Reietta Marco Santorelli" guys that shared with me the living space, the coffees and the laughs: you have been the funniest part of this Ph.D. thesis; all the people in the di Bernardo's Lab, for the philosophical lunches and for all the small bites of life that we shared.

Lastly, I would like to thank all my friends and my whole family, that accompanied me during several years on the long and winding road that brings (hopefully) to personal and professional satisfaction.

References

1. H.-Y. Chuang, M. Hofree, and T. Ideker, "A decade of systems biology," *Annual review of cell and developmental biology*, vol. 26, p. 721, 2010.
2. L. Chong and L. B. Ray, "Whole-istic biology," *Science*, vol. 295, no. 5560, pp. 1661–1661, 2002.
3. H. Kitano *et al.*, *Foundations of systems biology*. MIT press Cambridge, MA, 2001.
4. T. Ideker, T. Galitski, and L. Hood, "A new approach to decoding life: systems biology," *Annual review of genomics and human genetics*, vol. 2, no. 1, pp. 343–372, 2001.
5. Z. N. Oltvai and A.-L. Barabási, "LifeâĂŹs complexity pyramid," *Science*, vol. 298, no. 5594, pp. 763–764, 2002.
6. H. Kitano, "Systems biology: a brief overview," *Science*, vol. 295, no. 5560, pp. 1662–1664, 2002.
7. B. Palsson, "Properties of reconstructed networks," *Cambridge: Systems Biology*, 2006.
8. J. Fisher and T. A. Henzinger, "Executable cell biology," *Nature biotechnology*, vol. 25, no. 11, pp. 1239–1249, 2007.
9. L. Brim, M. Češka, and D. Šafránek, "Model checking of biological systems," in *Formal Methods for Dynamical Systems*, pp. 63–112, Springer, 2013.
10. T. Melham, "Modelling, abstraction, and computation in systems biology a view from computer science," *Progress in Biophysics and Molecular Biology*, 2012.
11. P. Baldan, N. Cocco, A. Marin, and M. Simeoni, "Petri nets for modelling metabolic pathways: a survey," *Natural Computing*, vol. 9, no. 4, pp. 955–989, 2010.
12. G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 1 ed., 2003.
13. H. Ge, A. J. Walhout, and M. Vidal, "Integrating âĂŸomicâĂŹinformation: a bridge between genomics and systems biology," *TRENDS in Genetics*, vol. 19, no. 10, pp. 551–560, 2003.
14. H. Kitano, "Scientific challenges in systems biology," in *Introduction to Systems Biology*, pp. 3–13, Springer, 2007.
15. D. B. Kell, "Metabolomics, modelling and machine learning in systems biology–towards an understanding of the languages of cells," *Febs Journal*, vol. 273, no. 5, pp. 873–894, 2006.
16. B. H. Junker and F. Schreiber, *Analysis of biological networks*, vol. 2. John Wiley & Sons, 2008.
17. H. Kitano, "Biological robustness," *Nature Reviews Genetics*, vol. 5, no. 11, pp. 826–837, 2004.
18. A.-L. Barabási and Z. N. Oltvai, "Network biology: understanding the cell's functional organization," *Nature Reviews Genetics*, vol. 5, no. 2, pp. 101–113, 2004.
19. U. Alon, "Biological networks: the tinkerer as an engineer," *Science*, vol. 301, no. 5641, pp. 1866–1867, 2003.
20. E. Alm and A. P. Arkin, "Biological networks," *Current opinion in structural biology*, vol. 13, no. 2, pp. 193–202, 2003.
21. N. M. Luscombe, M. M. Babu, H. Yu, M. Snyder, S. A. Teichmann, and M. Gerstein, "Genomic analysis of regulatory network dynamics reveals large topological changes," *Nature*, vol. 431, no. 7006, pp. 308–312, 2004.
22. X. Zhu, M. Gerstein, and M. Snyder, "Getting connected: analysis and principles of biological networks," *Genes & development*, vol. 21, no. 9, pp. 1010–1024, 2007.

23. S. Kikuchi, D. Tominaga, M. Arita, K. Takahashi, and M. Tomita, "Dynamic modeling of genetic networks using genetic algorithm and s-system," *Bioinformatics*, vol. 19, no. 5, pp. 643–650, 2003.
24. R. J. Prill, P. A. Iglesias, and A. Levchenko, "Dynamic properties of network motifs contribute to biological network organization," *PLoS biology*, vol. 3, no. 11, p. e343, 2005.
25. A. Blais and B. D. Dynlacht, "Constructing transcriptional regulatory networks," *Genes & development*, vol. 19, no. 13, pp. 1499–1511, 2005.
26. S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon, "Network motifs in the transcriptional regulation network of escherichia coli," *Nature genetics*, vol. 31, no. 1, pp. 64–68, 2002.
27. E. H. Davidson, *The regulatory genome: gene regulatory networks in development and evolution*. Academic Press, 2010.
28. M. Levine and E. H. Davidson, "Gene regulatory networks for development," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, no. 14, pp. 4936–4942, 2005.
29. A. A. Margolin, I. Nemenman, K. Basso, C. Wiggins, G. Stolovitzky, R. D. Favera, and A. Califano, "Aracne: an algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context," *BMC bioinformatics*, vol. 7, no. Suppl 1, p. S7, 2006.
30. G. Karlebach and R. Shamir, "Modelling and analysis of gene regulatory networks," *Nature Reviews Molecular Cell Biology*, vol. 9, no. 10, pp. 770–780, 2008.
31. U. Alon, *Introduction to Systems Biology: And the Design Principles of Biological Networks*, vol. 10. CRC press, 2007.
32. J. Hasty, D. McMillen, F. Isaacs, and J. J. Collins, "Computational studies of gene regulatory networks: in numero molecular biology," *Nature Reviews Genetics*, vol. 2, no. 4, pp. 268–279, 2001.
33. I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, "Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks," *Bioinformatics*, vol. 18, no. 2, pp. 261–274, 2002.
34. M. Zou and S. D. Conzen, "A new dynamic bayesian network (dbn) approach for identifying gene regulatory networks from time course microarray data," *Bioinformatics*, vol. 21, no. 1, pp. 71–79, 2005.
35. H. Herranz and S. M. Cohen, "Micrornas and gene regulatory networks: managing the impact of noise in biological systems," *Genes & development*, vol. 24, no. 13, pp. 1339–1344, 2010.
36. J.-D. J. Han, N. Bertin, T. Hao, D. S. Goldberg, G. F. Berriz, L. V. Zhang, D. Dupuy, A. J. Walhout, M. E. Cusick, F. P. Roth, *et al.*, "Evidence for dynamically organized modularity in the yeast protein–protein interaction network," *Nature*, vol. 430, no. 6995, pp. 88–93, 2004.
37. J.-F. Rual, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, N. Li, G. F. Berriz, F. D. Gibbons, M. Dreze, N. Ayivi-Guedehoussou, *et al.*, "Towards a proteome-scale map of the human protein–protein interaction network," *Nature*, vol. 437, no. 7062, pp. 1173–1178, 2005.
38. T. K. Prasad, R. Goel, K. Kandasamy, S. Keerthikumar, S. Kumar, S. Mathivanan, D. Telikicherla, R. Raju, B. Shafreen, A. Venugopal, *et al.*, "Human protein reference databaseâĂŤ2009 update," *Nucleic acids research*, vol. 37, no. suppl 1, pp. D767–D772, 2009.
39. S. Navlakha and C. Kingsford, "The power of protein interaction networks for associating genes with diseases," *Bioinformatics*, vol. 26, no. 8, pp. 1057–1063, 2010.
40. A. M. Feist, M. J. Herrgård, I. Thiele, J. L. Reed, and B. Ø. Palsson, "Reconstruction of biochemical networks in microorganisms," *Nature Reviews Microbiology*, vol. 7, no. 2, pp. 129–143, 2009.
41. U. Sauer, M. Heinemann, and N. Zamboni, "Getting closer to the whole picture," *Science(Washington)*, vol. 316, no. 5824, pp. 550–551, 2007.
42. N. E. Lewis, H. Nagarajan, and B. O. Palsson, "Constraining the metabolic genotype–phenotype relationship using a phylogeny of in silico methods," *Nature Reviews Microbiology*, vol. 10, no. 4, pp. 291–305, 2012.
43. L. Van Aelst and C. DâĂŹSouza-Schorey, "Rho gtpases and signaling networks," *Genes & development*, vol. 11, no. 18, pp. 2295–2322, 1997.
44. U. S. Bhalla and R. Iyengar, "Emergent properties of networks of biological signaling pathways," *Science*, vol. 283, no. 5400, pp. 381–387, 1999.

45. K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger, and G. P. Nolan, "Causal protein-signaling networks derived from multiparameter single-cell data," *Science*, vol. 308, no. 5721, pp. 523–529, 2005.
46. X. Liu and M. D. Betterton, *Computational Modeling of Signaling Networks*. Springer, 2012.
47. H. M. Sauro and B. N. Kholodenko, "Quantitative analysis of signaling networks," *Progress in biophysics and molecular biology*, vol. 86, no. 1, pp. 5–43, 2004.
48. J. V. Olsen, B. Blagoev, F. Gnad, B. Macek, C. Kumar, P. Mortensen, and M. Mann, "Global, in vivo, and site-specific phosphorylation dynamics in signaling networks," *Cell*, vol. 127, no. 3, pp. 635–648, 2006.
49. E. J. Molinelli, A. Korkut, W. Wang, M. L. Miller, N. P. Gauthier, X. Jing, P. Kaushik, Q. He, G. Mills, D. B. Solit, *et al.*, "Perturbation biology: inferring signaling networks in cellular systems," *PLoS computational biology*, vol. 9, no. 12, p. e1003290, 2013.
50. "An online Metabolic Network Reconstruction Catalogue." `http://gcrg.ucsd.edu/InSilicoOrganisms/OtherOrganisms`. Last Accessed: 12-02-2014.
51. "The online Roche Metabolic Pathway Atlas." `http://web.expasy.org/cgi-bin/pathways/show_thumbnails.pl`. Last Accessed: 12-02-2014.
52. G. Michal and D. Schomburg, *Biochemical pathways: an atlas of biochemistry and molecular biology*. John Wiley & Sons, 2013.
53. M. Kanehisa and S. Goto, "Kegg: kyoto encyclopedia of genes and genomes," *Nucleic acids research*, vol. 28, no. 1, pp. 27–30, 2000.
54. M. Kanehisa, M. Araki, S. Goto, M. Hattori, M. Hirakawa, M. Itoh, T. Katayama, S. Kawashima, S. Okuda, T. Tokimatsu, *et al.*, "Kegg for linking genomes to life and the environment," *Nucleic acids research*, vol. 36, no. suppl 1, pp. D480–D484, 2008.
55. "Kyoto encyclopedia of genes and genomes." `http://www.genome.jp/kegg/`. Last Accessed: 18-02-2014.
56. H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabási, "The large-scale organization of metabolic networks," *Nature*, vol. 407, no. 6804, pp. 651–654, 2000.
57. J. Stelling, S. Klamt, K. Bettenbrock, S. Schuster, and E. D. Gilles, "Metabolic network structure determines key aspects of functionality and regulation," *Nature*, vol. 420, no. 6912, pp. 190–193, 2002.
58. A. Bordbar, J. M. Monk, Z. A. King, and B. O. Palsson, "Constraint-based models predict metabolic and associated cellular functions," *Nature Reviews Genetics*, vol. 15, no. 2, pp. 107–120, 2014.
59. R. Steuer, "Computational approaches to the topology, stability and dynamics of metabolic networks," *Phytochemistry*, vol. 68, no. 16, pp. 2139–2151, 2007.
60. D. J. Wilkinson, "Stochastic modelling for quantitative description of heterogeneous biological systems," *Nature Reviews Genetics*, vol. 10, no. 2, pp. 122–133, 2009.
61. N. D. Price, J. L. Reed, and B. Ø. Palsson, "Genome-scale models of microbial cells: evaluating the consequences of constraints," *Nature Reviews Microbiology*, vol. 2, no. 11, pp. 886–897, 2004.
62. "A comprehensive database of scientific literature utilizing constraint-based modeling to predict biological phenotypes." `http://sbrg.ucsd.edu/cobra-predictions`. Last Accessed: 12-02-2014.
63. S. A. Becker, A. M. Feist, M. L. Mo, G. Hannum, B. Ø. Palsson, and M. J. Herrgard, "Quantitative prediction of cellular metabolism with constraint-based models: the cobra toolbox," *Nature protocols*, vol. 2, no. 3, pp. 727–738, 2007.
64. I. Rocha, P. Maia, P. Evangelista, P. Vilaça, S. Soares, J. P. Pinto, J. Nielsen, K. R. Patil, E. C. Ferreira, and M. Rocha, "Optflux: an open-source software platform for in silico metabolic engineering," *BMC systems biology*, vol. 4, no. 1, p. 45, 2010.
65. A. Hoppe, S. Hoffmann, A. Gerasch, C. Gille, and H.-G. Holzhütter, "Fasimu: flexible software for flux-balance computation series in large metabolic networks," *BMC bioinformatics*, vol. 12, no. 1, p. 28, 2011.
66. T. Dandekar, A. Fieselmann, S. Majeed, and Z. Ahmed, "Software applications toward quantitative metabolic flux analysis and modeling," *Briefings in bioinformatics*, vol. 15, no. 1, pp. 91–107, 2014.
67. M. Lakshmanan, G. Koh, B. K. Chung, and D.-Y. Lee, "Software applications for flux balance analysis," *Briefings in bioinformatics*, vol. 15, no. 1, pp. 108–122, 2014.
68. A. Wagner, "Metabolic networks and their evolution," in *Evolutionary Systems Biology*, pp. 29–52, Springer, 2012.

69. T. Khazaei, A. McGuigan, and R. Mahadevan, "Ensemble modeling of cancer metabolism," *Frontiers in physiology*, vol. 3, 2012.
70. D. L. Nelson, A. L. Lehninger, and M. M. Cox, *Lehninger principles of biochemistry*. Macmillan, 2008.
71. 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, *et al.*, "The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, pp. 524–531, 2003.
72. R. Pagliarini and D. di Bernardo, "A genome-scale modeling approach to study inborn errors of liver metabolism: Toward an in silico patient," *Journal of Computational Biology*, vol. 20, no. 5, pp. 383–397, 2013.
73. H.-G. Holzhütter, "The principle of flux minimization and its application to estimate stationary fluxes in metabolic networks," *European Journal of Biochemistry*, vol. 271, no. 14, pp. 2905–2922, 2004.
74. J. Whittaker, "What is software testing? and why is it so hard?," *Software, IEEE*, vol. 17, no. 1, pp. 70–79, 2000.
75. D. J. Richardson, S. L. Aha, and T. O. O'Malley, "Specification-based test oracles for reactive systems," in *Proceedings of the 14th International Conference on Software Engineering*, ICSE '92, (New York, NY, USA), pp. 105–118, ACM, 1992.
76. V. AA., *Model-Based Testing of Reactive Systems*, vol. 3472 of *Lecture Notes in Computer Science*. Springer, 2005.
77. R. W. Floyd, "Assigning meanings to programs," *Mathematical aspects of computer science*, vol. 19, no. 19-32, p. 1, 1967.
78. C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.
79. D. Harel, "Statecharts: a visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.
80. E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Proceedings of the Workshop on Logics of Programs*, vol. 131 of *Lecture Notes in Computer Science*, pp. 52–71, Springer, 1981.
81. J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in cesar," in *International Symposium on Programming*, pp. 337–351, Springer Berlin Heidelberg, 1982.
82. A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*, pp. 46–57, IEEE, 1977.
83. D. Harel and A. Pnueli, "On the development of reactive systems," in *Logics and Models of Concurrent Systems* (K. Apt, ed.), vol. 13 of *NATO ASI Series*, pp. 477–498, Springer Berlin Heidelberg, 1985.
84. M. Y. Vardi, "From Church and Prior to PSL," in *25 Years of Model Checking*, vol. 5000 of *Lecture Notes in Computer Science*, pp. 150–171, Springer, 2008.
85. S. A. Kripke, *Semantical Considerations on Modal Logic; Naming and Necessity*. Oxford University Press, 1971.
86. E. M. Clarke, "The birth of model checking," in *25 Years of Model Checking*, vol. 5000 of *Lecture Notes in Computer Science*, pp. 1–26, Springer, 2008.
87. E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.
88. E. A. Emerson, "The beginning of model checking: A personal perspective," in *25 Years of Model Checking*, vol. 5000 of *Lecture Notes in Computer Science*, pp. 27–45, Springer, 2008.
89. M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *Proceedings of the First Symposium on Logic in Computer Science*, IEEE Computer Society, 1986.
90. O. Grumberg and H. Veith, *25 years of model checking: history, achievements, perspectives*, vol. 5000 of *Lecture Notes in Computer Science*. Springer, 2008.
91. C. Baier and J.-P. Katoen, *Principles of model checking*. The MIT press, 2008.
92. J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill, "Sequential circuit verification using symbolic model checking," in *Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE*, pp. 46–51, IEEE, 1990.

93. S. Ben-David, C. Eisner, D. Geist, and Y. Wolfsthal, "Model checking at ibm," vol. 22, pp. 101–108, Kluwer Academic Publishers.
94. G. J. Holzmann, "Design and validation of protocols: a tutorial," *Computer Networks and ISDN Systems*, vol. 25, no. 9, pp. 981–1017, 1993.
95. W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda, "Model checking programs," *Automated Software Engineering*, vol. 10, no. 2, pp. 203–232, 2003.
96. J. L. Peterson, "Petri net theory and the modeling of systems," 1981.
97. R. Milner, *The polyadic π-calculus: a tutorial*. Springer, 1993.
98. O. Grumberg and H. Veith, *Communicating Sequential Processes. The First 25 Years*, vol. 3525 of *Lecture Notes in Computer Science*. Springer, 2004.
99. J. Hillston, *A compositional approach to performance modelling*, vol. 12. Cambridge University Press, 2005.
100. M. A. Arbib, *Theories of abstract automata (Prentice-Hall series in automatic computation)*. Prentice-Hall, Inc., 1969.
101. G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.
102. E. F. Moore, "Gedanken-experiments on sequential machines," *Automata studies*, vol. 34, pp. 129–153, 1956.
103. J. Richard Büchi, "Symposium on decision problems: On a decision method in restricted second order arithmetic," *Studies in Logic and the Foundations of Mathematics*, vol. 44, pp. 1–11, 1966.
104. R. Alur, "Timed automata," in *Computer Aided Verification*, pp. 8–22, Springer, 1999.
105. K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 134–152, 1997.
106. S. Yovine, "Kronos: A verification tool for real-time systems," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 1, no. 1, pp. 123–133, 1997.
107. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *Computer Aided Verification*, pp. 359–364, Springer, 2002.
108. K. L. McMillan, "The smv language," *Cadence Berkeley Labs*, pp. 1–49, 1999.
109. A. Cimatti and M. Roveri, "Nusmv 1.0: User manual," tech. rep., Technical report, ITC-IRST, Trento, Italy, 1998.
110. E. M. Clarke and I. Draghicescu, "Expressibility results for linear-time and branching-time logics," in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pp. 428–437, Springer, 1989.
111. E. A. Emerson and J. Y. Halpern, "âĂIJsometimesâĂİ and âĂIJnot neverâĂİ revisited: on branching versus linear time temporal logic," *Journal of the ACM (JACM)*, vol. 33, no. 1, pp. 151–178, 1986.
112. M. Y. Vardi, "Branching vs. linear time: Final showdown," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 1–22, Springer, 2001.
113. G. S. Avrunin, J. C. Corbett, and M. B. Dwyer, "Benchmarking finite-state verifiers," *International Journal on Software Tools for Technology Transfer*, vol. 2, no. 4, pp. 317–320, 2000.
114. G. S. Avrunin, J. C. Corbett, M. B. Dwyer, C. S. Pasareanu, and S. F. Siegel, "Comparing finite-state verification techniques for concurrent software," *University of Massachusetts, Amherst, MA*, 1999.
115. Y. Dong, X. Du, Y. Ramakrishna, C. Ramakrishnan, I. Ramakrishnan, S. A. Smolka, O. Sokolsky, E. W. Stark, and D. S. Warren, "Fighting livelock in the i-protocol: A comparative study of verification tools," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 74–88, Springer, 1999.
116. D. Peled, "Ten years of partial order reduction," in *Computer Aided Verification*, pp. 17–28, Springer, 1998.
117. D. Peled, "Combining partial order reductions with on-the-fly model-checking," in *Computer aided verification*, pp. 377–390, Springer, 1994.
118. M. Frappier, B. Fraikin, R. Chossart, R. Chane-Yack-Fa, and M. Ouenzar, "Comparison of model checking tools for information systems," in *Formal Methods and Software Engineering*, pp. 581–596, Springer, 2010.
119. E. Clarke and H. Veith, *Counterexamples revisited: Principles, algorithms, applications*. Springer, 2004.

120. C. Courcoubetis, M. Vardi, P. Wolper, and M. Yannakakis, "Memory-efficient algorithms for the verification of temporal properties," in *Computer-Aided Verification*, pp. 129–142, Springer, 1993.
121. Y. Choi, "From nusmv to spin: Experiences with model checking flight guidance systems," *Formal Methods in System Design*, vol. 30, no. 3, pp. 199–216, 2007.
122. G. J. Holzmann, J. R., and A. Groce, "Swarm verification techniques," *IEEE Transactions on Software Engineering*, vol. 37, no. 6, pp. 845–857, 2011.
123. J. Bard, T. Melham, E. Werner, D. Noble, *et al.*, "Plenary discussion of the conceptual foundations of systems biology.," *Progress in biophysics and molecular biology*, 2012.
124. N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages, and V. Schächter, "Modeling and querying biomolecular interaction networks," *Theoretical Computer Science*, vol. 325, no. 1, pp. 25–44, 2004.
125. F. Fages, S. Soliman, and N. Chabrier-Rivier, "Modelling and querying interaction networks in the biochemical abstract machine biocham," *Journal of Biological Physics and Chemistry*, vol. 4, pp. 64–73, 2004.
126. R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and computation*, vol. 104, no. 1, pp. 2–34, 1993.
127. G. Behrmann, A. David, and K. G. Larsen, "A tutorial on uppaal," in *Formal methods for the design of real-time systems*, pp. 200–236, Springer, 2004.
128. H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal aspects of computing*, vol. 6, no. 5, pp. 512–535, 1994.
129. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time markov chains," in *Computer Aided Verification*, pp. 269–276, Springer, 1996.
130. D. Bošnački and R. S. Mans, "Modeling and analysis of biological networks with model checking," *Algorithms in Computational Molecular Biology: Techniques, Approaches and Applications*, pp. 915–940.
131. G. Batt, D. Ropers, H. De Jong, J. Geiselmann, R. Mateescu, M. Page, and D. Schneider, "Validation of qualitative models of genetic regulatory networks by model checking: Analysis of the nutritional stress response in escherichia coli," *Bioinformatics*, vol. 21, no. suppl 1, pp. i19–i28, 2005.
132. E. Simao, E. Remy, D. Thieffry, and C. Chaouiya, "Qualitative modelling of regulated metabolic pathways: application to the tryptophan biosynthesis in e. coli," *Bioinformatics*, vol. 21, no. suppl 2, pp. ii190–ii196, 2005.
133. F. Ciocchetta and J. Hillston, "Bio-pepa: A framework for the modelling and analysis of biological systems," *Theoretical Computer Science*, vol. 410, no. 33, pp. 3065–3084, 2009.
134. P. Ballarini, R. Mardare, and I. Mura, "Analysing biochemical oscillation through probabilistic model checking," *Electronic Notes in Theoretical Computer Science*, vol. 229, no. 1, pp. 3–19, 2009.
135. R. Ghosh and C. Tomlin, "Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modelling: Delta-notch protein signalling," *Systems Biology*, vol. 1, no. 1, pp. 170–183, 2004.
136. J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn, "Probabilistic model checking of complex biological pathways," in *Computational Methods in Systems Biology*, pp. 32–47, Springer, 2006.
137. S. Schivo, J. Scholma, B. Wanders, R. A. U. Camacho, P. E. van der Vet, M. Karperien, R. Langerak, J. van de Pol, and J. N. Post, "Modelling biological pathway dynamics with timed automata," in *Bioinformatics & Bioengineering (BIBE), 2012 IEEE 12th International Conference on*, pp. 447–453, IEEE, 2012.
138. J. Scholma, S. Schivo, R. A. Urquidi Camacho, J. van de Pol, M. Karperien, and J. N. Post, "Biological networks 101: Computational modeling for molecular biologists," *Gene*, vol. 533, no. 1, pp. 379–384, 2014.
139. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "Prism: A tool for automatic verification of probabilistic systems," in *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 441–444, Springer, 2006.
140. H. de Jong, J. Geiselmann, C. Hernandez, and M. Page, "Genetic network analyzer: qualitative simulation of genetic regulatory networks," *Bioinformatics*, vol. 19, no. 3, pp. 336–344, 2003.
141. J. Barnat, L. Brim, I. Černá, S. Dražan, J. Fabriková, J. Láník, D. Šafránek, and H. Ma, "Biodivine: A framework for parallel analysis of biological models," *arXiv preprint arXiv:0910.0928*, 2009.

142. L. Calzone, F. Fages, and S. Soliman, "Biocham: an environment for modeling biological systems and formalizing experimental knowledge," *Bioinformatics*, vol. 22, no. 14, pp. 1805–1807, 2006.
143. C. Chaouiya, "Petri net modelling of biological networks," *Briefings in bioinformatics*, vol. 8, no. 4, pp. 210–219, 2007.
144. M. Carrillo, P. A. Góngora, and D. A. Rosenblueth, "An overview of existing modeling tools making use of model checking in the analysis of biochemical networks," *Frontiers in plant science*, vol. 3, 2012.
145. E. M. Clarke, O. Grumberg, K. L. McMillan, and X. Zhao, "Efficient generation of counterexamples and witnesses in symbolic model checking," in *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, pp. 427–432, ACM, 1995.
146. J. Fisher and N. Piterman, "The executable pathway to biological networks," *Briefings in functional genomics*, vol. 9, no. 1, pp. 79–92, 2010.
147. "The perl programming language." `http://www.perl.org/`. Last Accessed: 03-03-2014.
148. "The bioperl project." `http://www.bioperl.org/`. Last Accessed: 03-03-2014.
149. R. P. Lifton, S. Somlo, G. H. Giebisch, and D. W. Seldin, *Genetic diseases of the kidney*. Academic Press, 2009.
150. J. Harambat, S. Fargue, J. Bacchetta, C. Acquaviva, and P. Cochat, "Primary hyperoxaluria," *International journal of nephrology*, vol. 2011, 2011.
151. B. Hoppe, "Evidence of true genotype–phenotype correlation in primary hyperoxaluria type 1," *Kidney international*, vol. 77, no. 5, pp. 383–385, 2010.
152. E. Leumann and B. Hoppe, "The primary hyperoxalurias," *Journal of the American Society of Nephrology*, vol. 12, no. 9, pp. 1986–1993, 2001.
153. C. J. Danpure, "Primary hyperoxaluria type 1: Agt mistargeting highlights the fundamental differences between the peroxisomal and mitochondrial protein import pathways," *Biochimica et Biophysica Acta (BBA)-Molecular Cell Research*, vol. 1763, no. 12, pp. 1776–1784, 2006.
154. P. Cochat and G. Rumsby, "Primary hyperoxaluria," *N Engl J Med*, vol. 369, no. 7, pp. 649–658, 2013.
155. V. Lorenzo, A. Alvarez, A. Torres, V. Torregrosa, D. Hernandez, and E. Salido, "Presentation and role of transplantation in adult patients with type 1 primary hyperoxaluria and the i244t agxt mutation: single-center experience," *Kidney international*, vol. 70, no. 6, pp. 1115–1119, 2006.
156. C. Gille, C. Bölling, A. Hoppe, S. Bulik, S. Hoffmann, K. Hübner, A. Karlstädt, R. Ganeshan, M. König, K. Rother, *et al.*, "Hepatonet1: a comprehensive metabolic reconstruction of the human hepatocyte for the analysis of liver physiology," *Molecular systems biology*, vol. 6, no. 1, 2010.
157. R. P. Holmes, J. Knight, and D. G. Assimos, "Origin of urinary oxalate," in *Renal Stone Disease(AIP Conference Proceedings Seris Volume 900)*, vol. 900, pp. 176–182, 2007.