MASTER IN COMPUTER SCIENCE

Modelling gene regulatory networks with timed automata

Van Goethem, Simon

Award date:
2011

Awarding institution:
University of Namur

Modelling gene regulatory networks with timed automata

Simon Van Goethem

Faculty of Informatics

Facultés Universitaires Notre-Dame de la Paix

Master Thesis

2011 June

Supervisor:
Jean-Marie Jacquet

Abstract

Ce mémoire synthétise et développe les approches existantes de modélisation des réseaux de régulation génétique avec le formalisme des automates temporels. Les réseaux de régulation génétique sont des réseaux dédiés à la régulation du niveau d'expression des gènes au sein des cellules. Le formalisme des automates temporels est un formalisme adapté à la modélisation de systèmes où la concurrence joue un rôle important. De plus, ce formalisme possède d'intéressantes propriétés pour le *model checking*. La modélisation des réseaux de régulation génétique avec les automates temporels est un domaine qui est encore peu exploré. Ce mémoire synthétise les deux approches existantes dans ce domaines, à savoir une approche booléenne basée sur le formalisme de R. Thomas auquel est ajouté un aspect temporel, et l'approche suivie par le logiciel IKNAT. Pour cette dernière approche, ce mémoire présente certaines extensions ainsi qu'une formalisation de son principe de fonctionnement. Le formalisme obtenu est alors utilisé dans une nouvelle approche développée pour éviter certains problèmes de l'approche d'IKNAT. Enfin, l'approche booléenne temporalisée, l'approche d'IKNAT et la nouvelle approche sont comparées à partir d'études de cas.

The present Master thesis summarizes and extends the existing approaches which model gene regulatory networks with timed automata formalism. Gene regulatory networks are networks dedicated to the regulation of the gene expressions in the cell. Timed automata formalism is a formalism adapted to the modelling of systems where concurrency plays an important role. Moreover, the formalism has interesting properties for model checking. The modelling of gene regulatory networks with timed automata is a domain which has been little explored so far. This thesis summarizes the two existing approaches in this domain, the timed-boolean approach based on R. Thomas's formalism, and the approach followed by the software IKNAT. For the latter approach, the thesis presents some extensions and a formalization of its principles. The obtained formalism is then used to create a new approach developed to avoid some problems raised by the IKNAT approach. Finally, the timed-boolean approach, the IKNAT approach, and the new approach are compared in case studies.

Acknowledgements

Foremost, I would like to thank my thesis supervisor, Professor Jean-Marie Jacquet, and my internship supervisor at the university of Masaryk, Professor David Safranek, for their patience, their advise, and the numerous corrections they have provided this thesis with.

I would also like to thank my proofreaders Mrs. Marylène Petit and M. Roland Cornil. This being my first real writing experience in English, they have helped me make this work as readable as possible though several language mistakes may very well remain in this thesis. I humbly apologize for this.

Contents

1

Introduction

The understanding of biological life is one of the most promising and complex tasks for the current sciences. Promising because it is an incontrovertible step to fight and solve efficiently the multiple pathologies and lesions affecting organisms. And complex because an organism like the human body can be made up to 10^{14} cells, each being the result of an expressed genome subset which can total up to 35.000 genes. Arising from this complexity, biological networks turn out to be important life driving mechanisms. Among them, we can cite for example the neuronal networks for the organism behaviors, the interacting signalling networks for the interactions between cells, and the gene regulatory networks for the expression of the genome. These networks share several properties and aspects, including their complexity which reflects the importance of their role: the regulation of thousands of elements. In this domain, bioinformatics is an emerging and promising way to model these networks and deal with their complexity.

In the bioinformatics domain, this thesis is about the modelling of gene regulatory networks (GRNs) with timed automata formalism. GRNs are networks dedicated to the control of gene expressions inside cells. They drive, by the concentration of transcription factors, the expression of genes whose the products can themselves result in transcription factors regulating the expression of other genes.

Timed automata is a formalism stemming from the extension of automata formalism with the notion of time. The formalism is thus strongly focused on the time aspect and is ideal to model systems where concurrency plays an important role. Moreover, timed automata can have a very interesting property desirable for model checking: the decidability of the assertions about theirs states. For these reasons, timed automata formalism is often used to model critic systems with concurrency.

Thesis

Modelling GRNs with the timed automata formalism is a domain which has not been much explored yet. The aim of this master thesis is to summarize, to compare and to extend the work already done in this domain. In order to do this, the model checker for discrete timed automata called *Uppaal* has been employed, and a software called *model-builder* has been developed to generate the Uppaal timed automata for a given GRN and a given modelling approach.

Related work

The first related piece of work concerns the approach mostly followed to model GRNs with timed automata. This approach, called timed-boolean approach, is studied in [1, 6, 19] and leans on a biological background established by René Thomas and al [22, 23]. The idea is to model GRNs with a boolean abstraction and to define delays before changes in the activity level of the gene products.

The second work on timed automata comes from a master thesis [10] and models interacting signalling networks (ISNs) which are, for some aspects, similar to GRNs. The approach, referred to as IKNAT (the name of the software which employs it) in this thesis, drives the concentration level of signalling molecules by a mechanism of up events and down events. The events are produced by processes representing the reactions regulating the concentrations of signalling molecules. The balance between the up events and the down events received for a signalling molecule determines the trend to increase or decrease its concentration level.

Outline

This thesis is divided into three parts. The first part provides the introductory materials necessary to understand the thesis. This part comprises Chapter 2 which introduces the GRNs and Chapter 3 which introduces the formalism of timed automata and the model checker UPPAAL. In the second part, some modelling approaches of GRNs with timed automata are introduced. These approaches are, for Chapter 4, the existing timed-boolean approach, for Chapter 5, the existing IKNAT approach that we have diverted and extended to model GRNs, and for Chapter 6, a new approach that we have developed from a formalization of the IKNAT approach to avoid some of its problems. Finally, in the last part, composed of Chapter 7, the three approaches are compared through case studies.

2

Introduction to gene regulatory networks

In this chapter the subject of Gene Regulatory Networks (GRNs) is introduced first with a presentation of the GRNs and of their role in Section 2.1. Section 2.2 discusses the GRNs reconstruction challenge and presents the ODE models. In Section 2.3 a problem with the ODE models is presented, followed by two abstract models of GRNs which will be useful later in this thesis. Finally, the last two sections present some observations about the GRN models and a graphical notation used in this thesis to chart GRNs.

2.1 Introduction

Gene regulatory Networks (GRNs) form an important mechanism in biological life which allows the driving of intra-cellular metabolisms. A GRN is a set of genes (or DNA sequences) which tune the production rate of some biological substances by inter-regulations and in reaction to some incoming signals (sugar starvation, morphogenic signal, chemical substance, ...). The expression of a gene, through its associated RNA sequence, produces a certain type of proteins which can have a given goal in the cell metabolism or which can be used to regulate the expression of other genes. In this latter case, the proteins are transcription factors. A transcription factor can promote or inhibit the expression of a gene by changing the attracting force between the binding site of its RNA sequence and between the Ribosomes, whose aim is to translate RNA sequences into proteins. The bigger the concentration of a transcription factor is, the bigger its effect on the regulated gene expression is. Indeed, the more molecules of

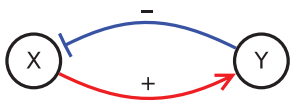


Figure 2.1: An example of simple GRN where a gene X activates a gene Y which itself inhibits the gene X.

the transcription factor there are, the more chances there are to get a molecule of the transcription factor on the right binding site of the adequate RNA segments. Note that a transcription factor can be in an ineffective form once produced, needing then the presence of a signal to become able (by changing slightly its chemical constitution) to regulate the expression of its target genes. However, in this thesis we make abstraction of this aspect by considering that a transcription factor is directly effective once produced.

A gene whose the product (i.e., the proteins produced by its expression) activates (inhibits) the expression of another gene is an activator or a promoter (inhibitor or repressor) for this other gene. We sometimes also use the terms *positive regulation* (*negative regulation*) to mean *promoter regulation* (*inhibitor regulation*). For convenience, we will refer to the product of a gene by the name of this gene. So, for a gene A, if we say that the concentration of A is high, we must understand that the concentration of the product of the gene A is high.

The usual representation of the topology of a GRN is a directed graph where the nodes of the graph represent the regulator and regulated genes and where the edges represent the regulations between genes (Figure 2.1).

Finally, note that the degradation of a protein by another (different from the inhibition of the expression of the protein gene) is also often included in the GRN models since it takes part directly in the regulation of the protein concentration.

2.2 GRN reconstruction challenge

The reconstruction (or mapping) of the GRN topologies is a very important challenge since it is required to explain and fight some diseases and to grasp the life rules. Indeed, understanding the influence network among genes is primordial to design and master the effects of a proper medicament and to understand the cause-effect mechanisms which drive the life. This reconstruction work can be separated into two main tasks: the inference of GRNs from data coming from biological experiments and the validation of these GRNs.

The first task, the inference of GRNs from experimental data can be done manually, by performing several experiments where a parameter (a mutation, the presence of a chemical substance, ...) is modified in each experiment. From the resulting modification in the biological system, the observer can deduce a possible set of genes directly dependent of the parameter. After several occurrences of such a cycle, a GRN can be designed. Another possibility is to use algorithms to browse a set of raw biological data (generated automatically from measurements) and to produce the set of all GRNs compatible with the data [20]. As the data become more complete, the algorithms can converge toward a smaller set by precluding some network configurations.

Note that it is possible to look for some well-known GRN structures to improve the comprehension of GRNs. These structures, called motifs, are recurrent in GRNs since they bring advantages to the robustness aspect (keep a GRN functional despite mutations) and to the reactivity aspect (quickly react to a change in the environment) of GRNs. One of the simplest motif is just a gene regulating itself. Another motif is a cascade of genes where a gene promotes another gene, which itself promotes another one, and so on ... These motifs and famous others are described in detail in Appendix A.

The second task, the GRNs validation, can be done by developing a model of a GRN, by making with it predictions over the behavior of the biological system under some conditions, and by confronting these predictions with the real biological behavior observed under the same conditions. According to the adequacy of the predictions with the reality, the GNR model can be said representative or not. If a GRN model is reliable, it can be used to make itself predictions over biological systems, avoiding experiments to be performed. A formalism commonly used to model GRNs is the Ordinary Differential Equation (ODE) formalism. A model using ODEs is introduced hereunder.

2.2.1 ODE modelling framework

The Ordinary Differential Equations (ODE) constitute a common formalism to model the average behavior of a GRN. By an average behavior, we mean the behavior observed on average among a large number of cells. This approach has to be set in opposition with stochastic approaches where the focus is put on the behavior of one cell (for a review of the different GRN modelling approaches, see [18]). Since there are several methods to extract such kinds of average information from cells, ODEs are often used with them. The ODE approach leans on several rates working over the average concentrations of transcription factors (a.k.a. Mass-Action Kinetics). These rates describe the global evolution of a given protein concentration following the different reactions modifying it. We can class these reactions into two main groups. In one group there are reactions

which are responsible for the decrease of the protein concentration, they are:

- **Mitose**: when a mother cell splits itself to produce two daughter cells (i.e., cell replication), the initial concentration of the protein in the mother cell is diluted on the volume of the two daughter cells.
- **Degradation**: there are in cells some enzymes whose purpose is the degradation of the protein in order to limit its concentration. The concentration of these enzymes can be function of a given transcription factor concentration (sometimes the degraded protein itself).
- **Inhibition**: if some inhibitor transcription factors of the gene of a protein are active in a cell, the synthesis of the protein will drop.

In the other group, there are reactions which are responsible for the increase of the protein concentration. The principal factor of such an increase is the presence of transcription factors which activate the synthesis of the protein.

The resulting dynamic of this set of chemical reactions is well described on average by ODEs. Following the Uri Alon's book [2], we give in the rest of this section a quick introduction to some important basic formulas used to model these reactions with ODEs.

Modelling concentration dynamics with ODEs

If Y is a protein, the variation of the Y concentration (denoted $[Y]$) is due to the difference between the production rate of Y (denoted β_Y) and the degradation/dilution rate of Y (denoted α_Y). We have thus

$$dY/dt = \beta_Y - \alpha_Y [Y] \tag{2.1}$$

where α_Y is multiplied by the Y concentration since the dilution/degradation of Y is proportional to its concentration.

The *steady state concentration* of Y (denoted $[Y]_{st}$) is the concentration reached by Y when $dY/dt = 0$. We can obtain the steady state value with $[Y]_{st} = \beta_Y/\alpha_Y$.

The solution of the system 2.1 - for Y starting from the steady state concentration and with a stopped production ($\beta_Y = 0$) - is

$$Y(t) = Y_{st}\, e^{-\alpha_t} \tag{2.2}$$

If the protein starts from the concentration zero and is submitted to a strong activation, the solution is

$$Y(t) = Y_{st}\,(1 - e^{-\alpha_t}) \tag{2.3}$$

An important concept associated with the steady state concentration is the *response time*. The response time (denoted $T_{1/2}$) is the time needed to reach the halfway steady state concentration from a given concentration level (often 0). For Equations 2.2 and 2.3, we can find $T_{1/2} = log(2)/\alpha_Y$ and thus, in these situations, the response time is independent of the β_Y rate and conversely dependent of the α_Y rate.

Note that a degradation of Y depending of the varying concentration of an enzyme Z can be modeled by adding to Equation 2.1 a subtraction factor multiplied by $[Y][Z]$.

For convenience, we will no more use the brackets [ ] to denote the concentration. We will just note Y for the Y concentration (instead of $[Y]$).

Modelling of simple regulation dynamic with ODEs

If X is a transcription factor which regulates Y, then the production rate of Y (β_Y) is a function of the concentration of X: $f(X)$. A set of proper differential equations which give good result to model the reactions on the binding site are Hill functions.

A Hill function is a monotonic and S-shaped function (Figure 2.2). If X is an activator we have the Hill function

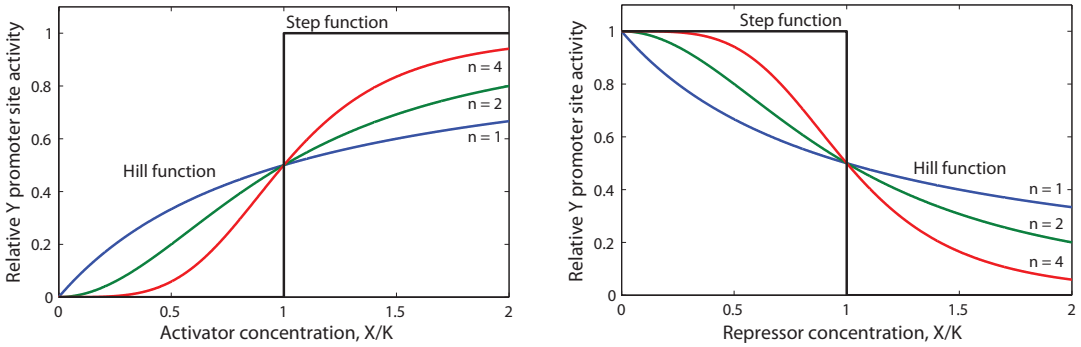
$$h(X) = \frac{X^n}{K^n + X^n}$$

where K is the activation coefficient which specifies the concentration of X required to have a significant effect. Then $f(X) = \beta \times h(X)$ with β being the maximal production rate dedicated to the binding site of X. In the reverse situation, when X is an inhibitor, we have the Hill function

$$h(X) = \frac{1}{1 + (\frac{X}{K})^n}$$

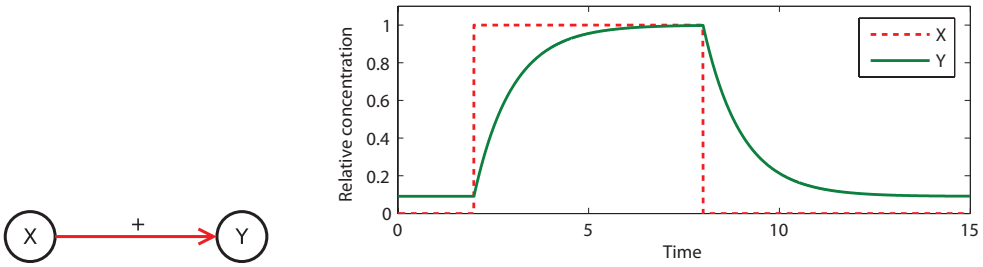
where K is the repression coefficient. Then $f(X) = \beta \times h(X)$ with β being the *basal production rate* of Y, i.e., the Y production rate in absence of its regulator. In both cases, n is the Hill coefficient and governs the steepness of the function (with generally moderate value between 1 and 4). As can be seen on Figure 2.2, the bigger the Hill coefficient is, the more stepwise like the curves are. On the extreme, with $\lim n \to \infty$, we have step functions (or boolean functions: $h(X) = \Theta(X \geq K)$ and $h(X) = \Theta(X < K)$) which are used as approximation in boolean models of GRNs where each regulation can be only active or inactive (example Figure 2.3).

The concentration reached by a gene product with only its basal production rate (in absence of its regulators) is called the *basal concentration*.



(a) Activator Hill function **(b)** Inhibitor Hill function

Figure 2.2: The hill functions for an activator regulation (a) and an inhibitor regulation (b), for the values n 1, 2, and 4. The step functions are Hill functions with $n \to \infty$.



(a) GRN topology **(b)** ODE simulation

Figure 2.3: An example of GRN topology (a) and its ODE simulation in (b). The gene X activates the gene Y. For the simplicity, in the simulation, the presence of the transcription factor X is simulated with a boolean abstraction (as regulation where $n \to \infty$), passing directly from a concentration null to a maximal concentration and vice versa. The ODE rates are the followings: the degradation rate of Y is 1, the basal production rate of Y is 0.2, and the production rate of Y induced by X is 2. When the regulator X is present ($X = 1$), the steady state concentration of Y is $(0.2+2)/1 = 2.2$, which is the maximal concentration than Y can reach. When the regulator X is absent, the steady state concentration of Y is $0.2/1 = 0.2$. The chart shows the relative concentrations which are for each protein, the protein concentration divided by the maximal concentration reachable by the protein (for Y: 2.2).

Modelling complex regulation dynamic with ODE

When a gene is the target of several regulators, the corresponding influences of the regulations can take several forms according to the kind of "*cooperation*" between their transcription factors. We do not state here any example of ODEs for cooperations since they are not easy to develop and are out of range for the work of this thesis. Instead, we mention some useful approximations of cooperation for a bi-regulation with regulators X and Y (generalizable to more regulators). We call the approximation of a functional cooperation for several regulations a *gate*.

One possibility, called a *SUM gate*, is the simple superposition of the regulation effects. The production rate of such a gate is:

$$f(X, Y) = \beta_X + \beta_Y$$

where each β is the production rate regulated by the associated transcription factor. A related possibility is the *multiplicative (MULT) gate* where the operator is a multiplication instead of an addition:

$$f(X, Y) = \beta_X \times \beta_Y$$

Finally, there are the boolean *gates AND* and *OR*:

$$f(X, Y) = \beta\Theta(X \geq K_X \ AND \ Y \geq K_Y)$$

$$f(X, Y) = \beta\Theta(X \geq K_X \ OR \ Y \geq K_Y)$$

where the function $\Theta()$ returns 1 if the condition inside is fulfilled, 0 otherwise, where the values K are thresholds over the concentration of their associated transcription factor, and where the β value is the production rate associated to the gate. Note that the use of the $\geq$ comparative operator is for promoters. In the case of an inhibition, the comparative operator commonly used is $<$ (e.g., if X is an inhibitor: $X < K_X$).

There are several other gates, but the ones presented here are often used to easily approximate the result of ODE cooperations.

Some examples of concentration trajectories regulated with boolean gates and with Hill functions are charted in the Appendix A dedicated to the GRN motifs.

Remarks

The ODE models are numerical models which are focused on quantitative parameters that are the rates. Since rates are specified in unit of time, there are in them inherent time information about the modeled GRNs.

2.3 Analysis and complexity challenge

The second challenge presented here is about the analysis and the complexity of GRNs. The ODE models are numerical models wit often no analytical solution to their differential equations. The only way to get analytical information from them is to run several numerical simulations with different values for the equation parameters and then observe the results. The huge number of possible combinations of these parameter variations (which take their values from a continuous range) induces the impossibility to do it exhaustively. Moreover, the GRNs are often large, increasing the number of genes and interactions, and therefore increasing the number of differential equations. More abstract models than ODE models can then be used in order to efficiently process analyses. In the remainder of this chapter, two of these abstract models (used further) are presented.

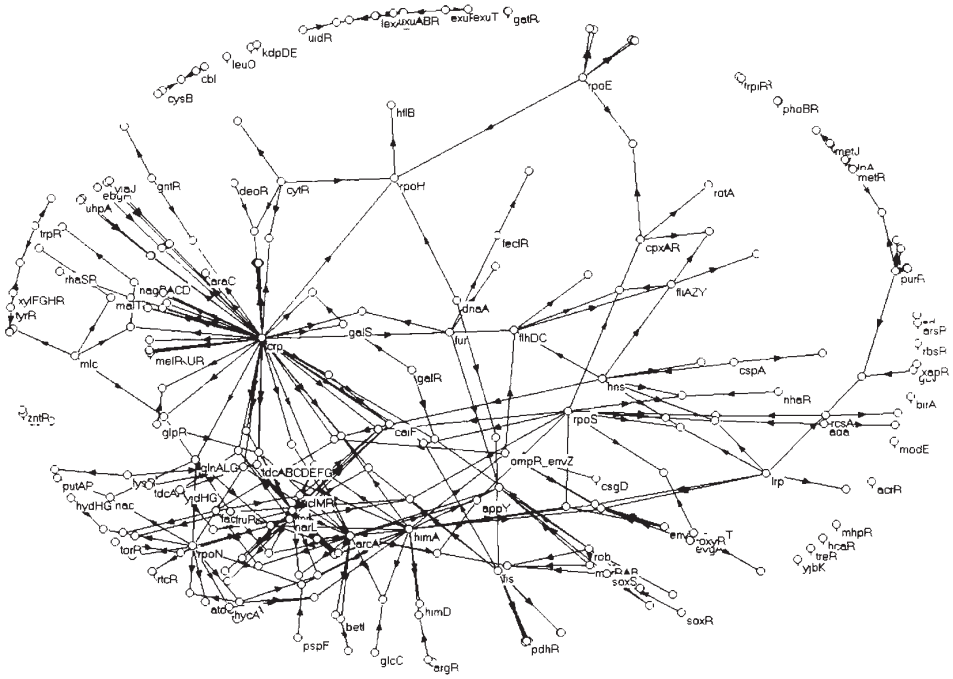


Figure 2.4: From the book of Uri Alon [2], a GRN which displays, with 420 nodes and 520 edges, 20 percents of the global GRN of the E. coli bacterium.

2.3.1 Piecewise linear ODE modelling

A piecewise linear ODE model for a GRN is a model which uses only linear functions. The one stated here comes from [9], is derived directly from the ODE model and is a generalization of the model presented in [21] where, for this latter, only one threshold by regulator is specified.

As we have seen in the ODE model, the evolution of a gene product Y is driven by the differential equation

$$dY/dt = fct(R_1, ..., R_n) - \alpha Y \tag{2.4}$$

where α is the dilution/degradation rate of Y and where $fct(...)$ is the Y production rate under the influence of the cooperations between the Y regulators R_1, ..., R_n. By using the $\Theta(...)$ function returning 1 if the condition inside is true, 0 otherwise, we can approximate the cooperation $fct(...)$ by a sum

$$\sum_{k=1}^{n} \sum_{i} \beta_{ki} \Theta(R_k \geq t_{ki})$$

where each t_{ki} is a threshold over the R_k concentration (with $\forall i \in \mathbb{N}_{>1} : t_{ki} > t_{k(i-1)} > 0$), and each β_{ki} is the assigned contribution of the R_k concentration $t_{ki} - t_{k(i-1)}$ to the Y production rate. Note that for certain cooperation functions, it is possible to get a really precise approximation by using the adequate β_{ki} and t_{ki} values with the proper granularities (the range of i for each R_k of the cooperation).

Now, by approximating in the same way the dilution/degradation term αY, we obtain

$$dY/dt = \sum_{k=1}^{n} \sum_{i} \beta_{ki} \Theta(R_k \geq t_{ki}) - \sum_{j} \alpha_j \Theta(Y \geq t_j)$$

which is Equation 2.4 with the cooperation $fct(...)$ and the dilution/degradation term αY replaced by their linear approximation, and where the relevant information about the concentration of the regulators is in terms of concentration levels delineated by their threshold values.

Note that in this model, the basal production rate of Y is included in the first β values of its inhibitors. If we extract the basal production rates of these values in an aggregated basal production rate (denoted $\mathcal{B}$) we obtain the following result:

$$dY/dt = \mathcal{B} + \sum_{k=1}^{n} \sum_{i} \beta_{ki} \Theta(R_k \geq t_{ki}) - \sum_{j} \alpha_j \Theta(Y \geq t_j) \tag{2.5}$$

where all the β parameters associated with a inhibitor have a negative value.

2.3.2 Boolean modelling

A boolean model of a GRN is an abstract model where the regulations can only be active or inactive (without effect), and where the cooperations between the regulations are boolean gates. In a boolean model, the concentration of a gene product is given in term of activity levels. For each gene product, its number of activity levels is minimal, i.e., the product has no useless activity level from the point of view of its outgoing regulations. So, if a product is a regulator for two genes, its maximal number of activity levels is three (its product concentration can be ineffective, sufficient to activate only the gene requiring the less concentration, or sufficient to activate both genes). Note that boolean models which can have more than one activity level for a product are sometimes qualified of models with multi-valued logic.

The boolean modelling approach stated here is the one of René Thomas presented on the original article *Boolean formalization of genetic control circuits* [22]. This approach requires the specification of two aspects for a GRN model: the *topology* and the *logicals parameters* of the GRN. These aspects - which are specified in what we call the Thomas's formalism - are used to build the model dynamics given with the *state transition graph*.

Note that to simplify the definition of the formalism which is heavy, we will speak here about activity levels of genes and not of their product. So, when we say that the activity level of a gene is 2, we must understand that the activity level of the gene product is 2.

Topology

In Thomas's formalism, the topology of a GRN is specified by a tuple $\Gamma = (G, sgn, tsd, r)$ where

- G = (V, E) is a directed graph with vertices set $V = \{\alpha_1, ..., \alpha_n\}$ and edges set $E \subset V \times V$.
- sgn is a function $E \to \{+, -\}$.
- tsd is a function $E \to \mathbb{N}_{\geq 1}$.
- r is a function $V \to \mathbb{N}_{\geq 1}$ such that, if $succ(\alpha_i) = S_{\alpha_i} \subset E$ with $\forall(\alpha_a, \alpha_b) \in S_{\alpha_i} : a = i$, $r(\alpha_i) = max(max(T(succ(\alpha_i))), 1)$, where $T(\{v_1, ..., v_m\}) = \{tsd(v_1), ..., tsd(v_m)\}$ with all $v_j \in E$. In other words, $r(\alpha_i)$ returns the maximal threshold of the regulation where the gene α_i is the regulator, or 1 if α_i is not a regulator.

The n vertices of the directed graph are the genes, the edges are the regulations. Each edge has a threshold given by tsd and a sign (+ for an activation, - for an inhibition) given by sgn. We call the set $\{k \in \mathbb{N}_0 \mid k \leq r(\alpha_i)\}$ the range of the gene α_i (denoted $range_i$), and $r(\alpha_i)$ is its maximal activity level. Since we require a minimal number of activity levels, if $(range_i > 1)$ we have $\forall k \in range_i : (k \neq 0) \Rightarrow (\exists (a,b) \in V : (a = \alpha_i) \wedge t((a,b)) = k))$.

We denote by e_{ij} the edge (α_i, α_j). The current activity level of a gene α_i is given by $l(\alpha_i)$, with $l(\alpha_i) \in range_i$.

Logical parameters

In a GRN, a gene α_i is a resource for a gene α_j if $e_{ij} \in E$ and if

- $l(\alpha_i) \geq tsd(e_{ij})$ with $sgn(e_{ij}) = +$ or
- $l(\alpha_i) < tsd(e_{ij})$ with $sgn(e_{ij}) = -$.

In other words, a gene α_i which regulates another is a resource for this one if it is an active promoter (over the threshold of the regulation) or if it is an inactive inhibitor (under the threshold of the regulation).

The set of all possible resources of a gene α_i is denoted by Res_i. For each subset p of Res_i, we define a logical parameter $K_{i,p}$ which states the activity level targeted by the gene α_i. So, for each gene α_i we have $0 \leq K_{i,p} \leq r(\alpha_i)$ for any $p \in Res_i$. It is with these logical parameters that the boolean gates of a gene is specified. The set of all logical parameters for a GRN topology Γ is denoted by $K(\Gamma)$.

State transition graph

Now we can define the main feature of this formalism, which is the state transition graph S_N for a GRN $N = (\Gamma, K(\Gamma))$. A state transition graph for N is a directed graph where the nodes are all the possible states of the GRN and where the arrows are all the possible transitions of the GRN from one state to another. A given state of the GRN is defined by a tuple of gene activity levels: $(l_1, ..., l_n)$ where $0 \leq l_i \leq r(\alpha_i)$ for any $i \in \mathbb{N}_{[0,n]}$.

All the possible states of the GRN are denoted as $state_\Psi = \{state_1, ..., state_m\}$, a transition from a $state_e$ to another $state_f$ is denoted by $state_e \longrightarrow state_f$ (with $state_e, state_f \in state_\Psi$) and exists iff it is one of the following types of transition:

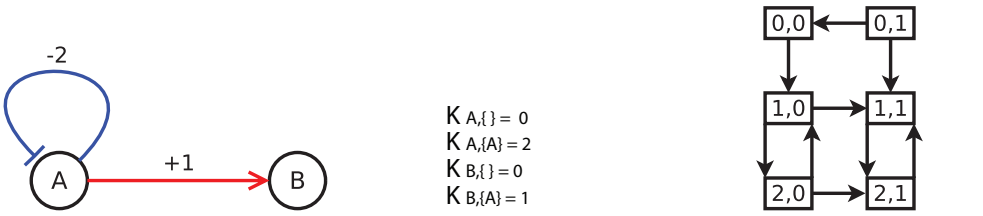
- **Increasing transition**: $(l_1, ..., l_t, ..., l_n)_e \longrightarrow (l_1, ..., l_t + 1, ..., l_n)_f$ if $K_{t,Re(state_e,\alpha_t)} > l_t$.
- **Decreasing transition**: $(l_1, ..., l_t, ..., l_n)_e \longrightarrow (l_1, ..., l_t - 1, ..., l_n)_f$ if $K_{t,Re(state_e,\alpha_t)} < l_t$.

where Re is a function $states_\Psi \times V \rightarrow \Omega$ with Ω being the set of all possible subsets of V and such that $Re(state_s, \alpha_j) = \omega \Leftrightarrow (\forall \alpha_i \in \omega : e_{ij} \in E \wedge ((l(\alpha_i) \geq tsd(e_{ij}) \wedge sgn(e_{ij}) = +) \vee (l(\alpha_i) < tsd(e_{ij}) \wedge sgn(e_{ij}) = -)))$. In other words, Re returns the set of resources for a given gene and for a given state of the GRN.

According to this definition, a gene is allowed to change its activity level in a state e only if its level is lower or bigger than the level corresponding to its active logical parameters in the state e.

With this definition of the transitions, it appears that, in the Thomas's formalism, the change of activity level are asynchronous (only one gene changes its activity level per transition) and a gene can change its activity level of only one level in a step.

Figure 2.5 gives an example of GRN with its state transition graph.



(a) Topology **(b)** Logical parameters **(c)** State transition graph

Figure 2.5: An example of GRN topology (a), logical parameters (b), and the associated state transition graph (b) in Thomas's formalism. In this example, the gene A promotes the gene B when its activity level is equal to or bigger than 1, and the gene A is self-regulated in a negative manner when its activity level is equal to 2. The range of A is $\{0, 1, 2\}$ and the range of B is $\{0, 1\}$. A state tuple is (A, B). As we can see on the state transition graph, the system loops infinitely between the state (1, 1) and (2, 1).

Remarks

The Thomas's formalism provides a way to simulate GRNs in a boolean manner. The result of these models is interesting by the qualitative information they provide. With the state transition graph, a model gives qualitative information about the possible

evolutions of a GRN. Moreover, there are several feasible static analyses on the state transition graph which can provide very useful information about the cycles, the stability and the instability of the GRN. Some of them are described in [24]. There is also a software called GINsim [15] which allows the graphical creation of boolean networks and the generation of their state transition graph. However, GINsim deals with a slight variant of this formalism since it allows the specification of regulations with more than one threshold.

It is important to note that there is no time information in this formalism except the state sequences given by the paths in the state transition graph. But these sequences show only the possible state orders and nothing about the duration of the states or the time required by a gene product to change its activity level. In consequence, some evolutions in the state transition space of a GRN can be revealed as unrealistic by a model with a temporal aspect more developed.

2.4 Observations

From the previously introduced GRN models it is possible to isolate several aspects and characteristics which are useful to classify and to compare the GRN models with timed automata presented in this thesis.

Time aspects

Since the formalism used for the modelings in this thesis (i.e., timed automata formalism) leans preeminently on a time aspect, the role and importance of the time in the GRN modellings is a central point in the explored approaches. Indeed, several characteristics as for example the specification easiness and the modelling precision depend on the time aspect. These characteristics, except the one stated hereunder, are developed according to each approach in their dedicated chapter.

The characteristic that we introduce here is the ability to model noise effect in the timing of a model. For this, the model is specified with time intervals instead of precise time values. The idea is no more to say that an event has to be realized after a delay X but to let the possibility for the event to occur between a delay X and a delay Y. This ability, which is not present in ODE models, introduces a certain non-determinism which can be used when we have no precise idea over the delays or when the occurrence delays of the biological events seem to be confronted to a lot of biological noise.

Concentration levels

In the GRN models which are explore in this thesis, the concentration of the gene products are defined as discrete levels. These levels can be directly linear to the concentration of the gene products (i.e., the level 2 correspond to a concentration two times bigger than the level 1) or can be an abstraction of this concentration as it is the case for the Thomas's formalism (the levels correspond to the thresholds of the regulations). In this latter formalism, since the concentration level is linked to the activity of the regulation, we speak of activity levels instead of concentration levels.

Note that with asynchronous models where the events are not processed in parallel, there is a non-determinism related to the processing order of the level changes (the events). Indeed, when two genes have to change their product level simultaneously, the processing of one gene before the other can entail a different evolution of the GRN from the converse situation.

Two concentration levels are noticeable for a gene product. The first is the basal concentration level which represents the level reached by the gene product with only its basal production rate. The ability for a model to allow non-zero basal concentration levels is important for the modelling of GRNs. Note that if we speak of basal concentration level, it can in fact correspond to a situation with oscillation between two neighboring concentration levels.

The second noticeable concentration level is the maximal concentration level that a gene product can attain. This value can be either non-limited or, more often, limited. For this last case and following the kind of model (boolean, ...), the freedom let to the designer for the maximal concentration levels of gene products is not the same and influences the expressiveness of the model.

Gates

The possibility for a model to represent the principal gates that are the boolean gates (AND and OR) and the non-boolean gates (SUM and MULT) is an important aspect to simulate the cooperation between regulations. As well, we can consider the ability of a model to allow modelling of more complex gates.

Multi-effects regulations

Another aspect related to the regulations is the possibility to specify regulations which are not only active or inactive (like in boolean models), but which can have several levels in the intensity of their effects following the concentration of their regulator.

Degradation regulations

As stated before, beside the normal regulations which are the inhibitions and the promotions of a protein by transcription factors, a protein can be degraded by an enzyme which is itself a protein whose the effect depends on its concentration. This degradation can be considered as a regulation acting in a different way and thus can require to be modeled in a different way. For example, in the ODE models, an inhibition is a division of a basal production rate by a factor depending on the regulator concentration, whereas a degradation is a subtraction of a factor which is multiplied by the regulator concentration and by the regulated protein concentration.

The ability for a model to allow this difference is an interesting aspect.

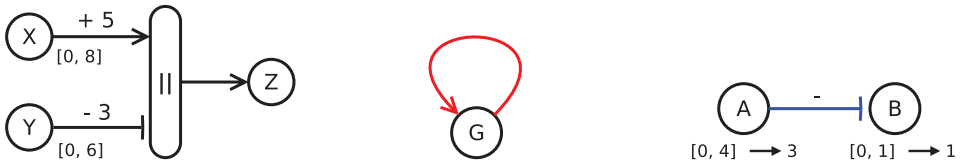
2.5 Graphical notation

To end this chapter, we present here the graphical notation used to express the GRNs. This notation has already been used in this chapter for simple GRNs and it has only a few elements since we do not need to state several details about the chemical reactions, but only the basic topology of the GRNs.

In this notation, each gene is represented by a circle with its name inside. The regulations between the genes are represented by an arrow for an activation, a "T arrow" for an inhibition and a "bullet arrow" when both are possible or when the kind of regulation is undetermined. We also sometimes add the sign - (+) or use a blue (red) color for an inhibition (activation) to emphasize the type of the regulation.

If we want to make explicit the existence of a gate as the one presented in Section 2.2.1, we use a rounded rectangle with inside a $\sum$, $\times$, & ,or || for respectively a SUM, MULT, AND, and OR gate.

A number over the edge of a regulation is the threshold value of the concentration level of the regulator above which the regulation is active.



(a) Or gate **(b)** Auto activation **(c)** Inhibition

Figure 2.6: Several examples of notations. In (a) an OR gate with threshold and range 5, $[0, 8]$ for X and 3, $[0, 6]$ for Y. In (b) a positive auto-regulation and in (c) an inhibition where A has a range $[0, 4]$ and go spontaneously to the concentration level 3 and where B has a range $[0, 1]$ and go spontaneously to the concentration level 1.

Finally, if we want to indicate on the graph the range of concentration levels and the basal concentration level of a gene product, we use respectively the notation *[x, y]* and the notation $\longrightarrow z$.

2.6 Conclusion

In this chapter, the necessary materials required to understand the GRN topic were introduced. In these materials, we explore the topology of GRNs which states a GRN as a graph where the genes are the nodes and where the regulations are the edges, we examine the challenges associated to the GRNs, and we briefly introduce the notion of ODE models working with rates

Moreover, in response to the problem raided by ODE models, two abstract models were presented: the piecewise linear ODE model directly related to the ODE model, and the boolean model using boolean expressions and abstract activity levels. These two models have not been chosen at random since they are used in timed automata models presented and analyzed in the sequel of this thesis.

Finally, with these materials and abstract models, we have delineated several concepts and aspects which are useful to compare the capabilities of different GRN models.

3

Introduction to timed automata

This chapter provides a short introduction to the formalism of timed automata and its implementation in Uppaal [7], a model checker for discrete timed automata. In Section 3.1, a short informal explanation about timed automata is given. In Section 3.2, the relevant part of the formalism of timed automata to our thesis is defined. The last section gives some informal explanations about the discrete timed automata used in Uppaal, their extensions, and what the latter implies with respect to the aspects of timed automata.

3.1 Introduction

A timed automaton is a finite state automaton extended with the notion of time and used to simulate and validate system where the time aspect and the concurrency aspect are crucial. For this reason, when timed automata are used, it is often with critical systems in engineering domains. Such systems can be for example the dispatching system of trains in a station, or the control system of a rocket.

3.1.1 Finite state automata

A finite state automaton (or finite state machine, but we will restrict ourself to the name finite automaton) is basically a theoretical machine dedicated to the validation of input words of the alphabet and the grammar of a regular language. In order to do this, an automaton is composed of a finite set of *locations* (or *states*) linked to each other with *switches* (or *transitions*). The locations are graphically represented each by a circle and the switches each by an arrow from the source location to the destination

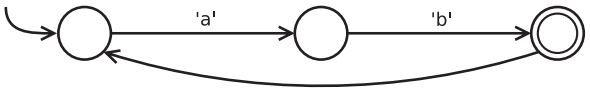


Figure 3.1: An automaton which accepts only words respecting the regular expression *(ab)+* (e.g. “ab”, “abab”, ...). The locations are the circles. The switches are the arrows. The final location is represented with a nested circle. The initial location has an orphaned arrow.

location. Moreover, the set of locations is devised in three parts: the initial location (represented with an input arrow without source location), the set of final locations (represented with a nested circle), and the set of normal locations. At a given moment, only one location can be active. Naturally, the first active location of the automaton is the initial location of the automaton.

Each switch has a *guard* indicating the input symbol required (from the alphabet of the language) to fire the switch if the source location of the switch is active. When a location is active, an output switch of a location can be fired if the current input symbol is allowed by the guard of the switch. If the switch is fired, the target location of the switch becomes active instead of the source location of the switch, and the current input symbol of the automaton is switched to the next symbol of the word. A switch can also be guarded with an empty symbol, allowing the fire of the switch irrespectively of the input symbol, since the validation of this one is delayed to the next switch. When an active location has many switches which can be fired, one of them is fired randomly.

The input word is validated (recognized as respecting the grammar of the regular language for which the automaton is dedicated) when a final location of the automaton is active and when the whole word is processed. If the automaton is stuck on a not final location without switch to fire, there is a deadlock and the word is not validated.

3.1.2 Timed automata

A timed automaton is an automaton with one or several associated *clocks* and with some constraints over these clocks in the guards of the switches. A constraint over a clock C can be for example $[C = 10]$ or $[9 \leq C \leq 10]$. Moreover, when a switch is fired, a defined subset of the clocks can be reset to the value 0. The grammar validated by such automaton is said to be a timed-regular grammar and is employed to specify a timed-regular language. A timed-regular language contains timed-regular words. A timed-regular word is a regular word where each symbol is associated with a timed value. Timed words can be represented by a sequence of couple (a, t) where a is a symbol and t is the time associated with the symbol (see Figure 3.2 for an example).

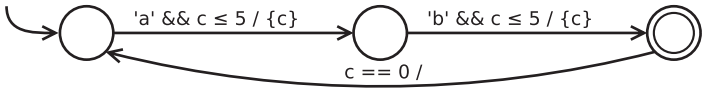


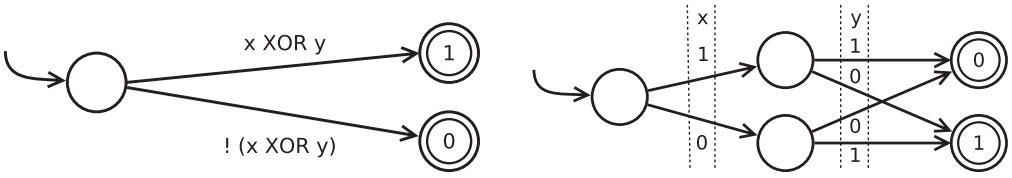
Figure 3.2: A timed automaton with a clock c which accepts only timed words respecting the regular grammar *(ab)+* and where each symbol follows the previous in a delay of maximum 5 time units (e.g. "(a, 3)(b, 8)(a, 9)(b, 13)"). The formula before "/" is the guard and the set after are the reset clocks.

Finally, a discrete timed automaton is an automaton where the time takes only discrete values (or natural numbers).

3.1.3 High level automata

From the previous automata dedicated to the language validation, it is possible to specify high level automata which allow the use of variables, invariants for the locations, guards with elaborate conditions, and complex operations in the switches. All these high level specifications are just translated (by preprocessing or on-the-fly) in new locations and switches to come back to the original automata formalism (see Figure 3.3).

Therefore, the use of automata (timed or not) is not restricted to the only domain of language validation, but can be extended to the validation of various aspects in diverse domains provided with a representation (or modelling) in an automaton.



(a) High level automaton **(b)** Equivalent classic automaton

Figure 3.3: An automaton which computes the XOR result over two boolean variables x and y. In (a) the high level automaton, in (b) an equivalent classic automaton which reads the word constituted of x and y concatenated.

3.2 Timed automata formalism

After the previous informal presentation of the timed automata formalism through simple automata, this section gives more formal definitions about timed automata. These definitions are given following the original article *A theory of timed automata* [4] and the article *Decision Problems for Timed Automata: A Survey* [5].

3.2.1 Definition of a timed automaton

A timed automaton is a finite automaton augmented with a set of clocks. The language of a timed automaton is a timed-regular language. A timed language over an alphabet Σ is a language composed of timed words. A timed word over an alphabet Σ is a sequence of $(a_0, t_0), (a_1, t_1), ..., (a_k, t_k)$ where each $a_i \in \Sigma$,each $t_i \in \mathbb{R}_{\geq 0}$ and $t_0 \leq t_1 \leq ... \leq t_k$.

A timed automaton T over an alphabet Σ is a tuple $\langle L, l_0, L_F, X, E \rangle$, where

- L is a finite set of locations,
- $l_0 \in L$ is the initial location,
- $L_F \subseteq L$ is a set of final locations,
- X is a finite set of clocks,
- $E \subseteq L \times \Sigma^e \times \Phi(X) \times 2^X \times L$ is a set of switches where $\Phi(X)$ is the set of possible constraints over the clocks X. A switch $\langle l, a, g, \lambda, l' \rangle$ represents an edge from the location l to the location l' for the symbol a. The guard g is a clock constraint over X which specifies when the switch is enabled. The update $\lambda \subseteq X$ gives the clocks to be reset to 0 with this switch. $\Sigma^e = \Sigma \cup \{\epsilon\}$ where ϵ is the empty symbol.

The set of all possible clock constraints on a set X of clocks ($\Phi(X)$) is defined by the grammar

$$g := x \leq c \mid c \leq x \mid x < c \mid c < x \mid g \wedge g$$

where $x \in X$ and $c \in \mathbb{Q}$.

A clock valuation v for a set X of clocks assigns a real value to each clock of the set, in other words v is a mapping from X to $\mathbb{R}_{\geq 0}$. For $\delta \in \mathbb{R}_{\geq 0}$, $v + \delta$ is the clock valuation which maps every clock x to the value $v(x) + \delta$. In a timed automaton, the time elapses only on the locations (the switches are instantaneous), and all the clocks are synchronous (i.e. $(v(x) + \delta) \Leftrightarrow (v + \delta)$ for the elapsed time $\delta \in \mathbb{R}_{\geq 0}$ in a same location (without reset of clocks) and for any clock $x \in X$).

Subsequently we will subscript the elements of the tuple with the identifier of a timed automaton to denote the elements of the automaton (e.g., L_F^T denotes the set L_F of the automaton T).

Semantics

The semantics of a timed automaton T is defined by a related infinite automaton $S(T)$ over the alphabet $\Sigma \cup \mathbb{R}_{\geq 0}$. A state of $S(T)$ is a pair (l, v) such that l is a location of T ($l \in L_T$) and v is a clock valuation for X^T. A state (l, v) of $S(T)$ is an initial state if l is the initial state of T ($l = l_0^T$) and if $v(x) = 0$ for all clocks x of X^T. A state (l, v) is a final state of S_T if l is a final location of T ($l \in V_F$). There are two types of transitions in $S(T)$:

- **Elapsing time**: for a state (l, v) and a time increment $\delta \in \mathbb{R}_{\geq 0}$, $(l, v) \xrightarrow{\delta} (l, v+\delta)$.
- **Location switch**: for a state (l, v) and a switch $\langle l, a, g, \lambda, l' \rangle$ such that v satisfies the guard g, $(l, v) \xrightarrow{a} (l', v\,[\lambda := 0])$. Where $v\,[\lambda := 0]$ denotes the clock valuations v where the clock of $\lambda \subseteq X^T$ are reset to the value 0.

For a timed word $w = (a_0, t_0), (a_1, t_1)...(a_k, t_k)$ over Σ^e, a run of $S(T)$ over w is a sequence

$$q_0 \xrightarrow{t_0} q'_0 \xrightarrow{a_0} q1 \xrightarrow{t_1 - t_0} q'_1 \xrightarrow{a_1} q_2 \xrightarrow{t_2 - t_1} q'_2 \xrightarrow{a_2} q_3 \xrightarrow{t_3 - t_2} ... \xrightarrow{a_k} q_{k+1}$$

where q_0 is the initial state of $S(T)$. The word is accepted if q_{k+1} is a final state of $S(T)$.

For example, if we reuse the timed automaton of Figure 3.2, a run over the timed word "(a, 3)(b, 8)(a, 9)(b, 13)" of the automaton - representing the semantic of the timed automaton - is the sequence

$$(l_0, 0) \xrightarrow{3} (l_0, 3) \xrightarrow{a} (l_1, 3) \xrightarrow{5} (l_1, 8) \xrightarrow{b} (l_2, 8) \xrightarrow{e} (l_0, 0) \xrightarrow{1} (l_0, 1) \xrightarrow{a} (l_1, 1) \xrightarrow{4} (l_1, 5) \xrightarrow{b} (l_2, 5)$$

where l_0 denotes the initial location of the timed automaton, l_2 the final location, and l_1 the central location.

Note that we have take care to use here the words *state* and *transition* instead of the synonyms *location* and *switch* to speak of the automata defined for the semantics (as we will also do for the state space).

3.2.2 Discrete timed automata

From the formalism of timed automata, a discrete timed automaton is an automaton with switches occurring only at an integral valuation time ($v \in \mathbb{N}^{|X|}$). This entails

that each constant c used in the guard (according to the grammar g) is an natural number ($c \in \mathbb{N}$). More generally, we speak about timed automata with sampled rate f for timed automata where the clock valuation of switches has to be a multiple of f. A timed automaton T with sampled rate f is denoted T^f. Note that every timed automaton with a rational sampled rate f can be transformed into a discrete timed automaton by an upscaling of the time values by $1/f$.

State space

The state space $\mathcal{S}(T)$ of a time automaton T^f is an infinite automaton where the state are the possible configurations of T^f. A state of $\mathcal{S}(T)$ is a couple (l, v) where $l \in L^T$ and v is a valuation of X^T. Each switch of $\mathcal{S}(T)$ has a guard e and represents a possible change in the configuration of T^f (an elapsing time of f or a change of active location). The initial state of $\mathcal{S}(T)$ is the state (l_0^T, v) with $v(x) = 0$ for all clocks x of X^T.

A path in a state space is a sequence of state which are linked by transitions.

The state space $\mathcal{S}(T)$ is a concept similar to the automaton $S(T)$ described for the semantics of T. The difference here is that there is an elapsing-time transition fired for every elapsed time f. Note that such a state space is not possible with a real timed automaton (without sample rate f).

3.2.3 Decidability of the reachability problem

There are several properties and facts known about timed automata. Here we present an important property associated to timed automata with sample rate. This property is the decidability of the reachability problem.

The reachability problem consists simply to know whether a given state of an automaton can be reached or not. Formally, for an automaton T and a state s of T, the matter is to know if s is included or not in the state space $\mathcal{S}(T)$. For such a problem, we have the following theorem:

Theorem 1 *For a timed automaton, the reachability problem is decidable [3].*

This decidability of the reachability problem is an important property for an timed automaton since the matter is the possibility, for a model checker, to verify every assertion about the state space of the automaton. Indeed, the veracity of an assertion about the state space of the automaton is definitely decidable if the model checker

can know if the states which infirm (for safeness assertions) or confirm (for liveness assertions) the assertion are reachable. For this reason, timed automata are often used for the modelling of critic systems.

3.2.4 Timed automata extensions

There are several extensions to the timed automata formalism which were proposed in order to extend its expressiveness or simply for the convenience of writing. Sometimes they affect the automata properties. Some of them are relevant for our work since they are used in Uppaal. We review them here.

Invariant: An invariant allows one the possibility to specify constraints over clocks in each location of an automaton. The constraints specified for a given location have to be respected while the location is active. This extension does not improve the expressiveness nor change the property of an automaton. Indeed, an invariant of a location can be easily replaced by adding (with a conjunction) the invariant into the guards of the outgoing or ingoing switches.

Stopwatch: A stopwatch is a clock which can be frozen in a location. When a clock is frozen in a location, and when the location is active, the valuation of the clock does not evolve while the time elapses. This extension improves the automata expressiveness [13] but also impacts the reachability property (i.e., the decidability of the reachability problem) since the reachability problem is no more decidable with the use of stopwatches [3].

Additional clock constraints: Additional clock constraints allow one to use in guards (and invariants) constraints of the form $c_1 - c_2 \ \# \ k$, $c_1 + c_2 \ \# \ k$, where c_1 and c_2 are clocks, k is a constant, and # stands for a binary operator in $\{\leq, \geq, =, \neq, <, >\}$. This extension improves the expressiveness of timed automata and preserves the reachability property [12].

3.3 The Uppaal discrete timed automata

There are several tools which can process model checking on timed automata. We can cite Kronos [14], Hytech [17], IF [11], and Uppaal [7]. Kronos is a old model checker for real timed automata. Hytech is a tool for the model checking and for the development of linear hybrid systems, which are automata where "clocks" can evolve with different real rates. IF is a tool based on some algorithms of Kronos for the model checking of real timed automata. Finally, Uppaal is a tool dedicated to the conception and to the

model checking of discrete timed automata.

Since Uppaal provides a very efficient model checker, and since there is a strong community and an up-to-date documentation about it, we have chosen it to deal with timed automata in this research. Subsequently, after the previous introduction to the concepts and formalism of timed automata relevant for our work, we end this chapter by the relevant specificities for us of the discrete timed automata of Uppaal. These specificities concern the allowed high level specifications, the timed automata graphical notations, and the Uppaal model checking capabilities.

3.3.1 Timed automata extensions

Uppaal supports all the previously stated extensions of the timed automata formalism. Thus, it is possible with Uppaal to specify an *invariant* in each location, to use *stopwatches* (only in the currently under-development version 4.1), and to define clock constraints with *differences* and *sums* over two clocks. In Uppaal, a clock c in a given location is frozen or not according to an invariant over the derivation of the clock: `c'==#` where `#` stands for the value 0 (to freeze) or 1.

In addition to these extensions, Uppaal allows the specification of *urgent locations* and some other important features which are presented herein. An urgent location is a location where the time is frozen for all clocks of the automaton. It is equivalent to the addition of an extra clock X with the association of an invariant $X \leq 0$ to the location.

3.3.2 High level specifications

Uppaal provides a set of discrete high level expressions which can be used with timed automata. These allow the use of variables and constants with boolean and integer types and with the usual discrete operations on them. It is also possible to specify arrays, functions, if-then-else statements, and loop statements. However, since all types are explicitly bound in Uppaal, the specification of a function over an array is only possible for a given size of array. For example, it is not possible to specify a function over arrays of size 2 or 3, we have to specify two different functions, one for the size 2 and another for the size 3. The situation is the same with loop-statements over arrays.

Expressions over constants and variables can be used in guards, invariants, and in update fields associated with each switch. An update field is processed when the associated switch is fired.

3.3.3 Processes

The structure of a timed automaton in Uppaal is composed of processes. A process is an instantiation of a template. A template is the structure of a single timed automaton (like the one described previously) which can be instantiated for several processes. Each template provides a signature which is used to launch and to parametrize the instantiation of the template, in the same sense as the signature of a function. A parameter for a template can be a variable, a clock, an array, ..., or a synchronization channel (see below). A parameter can be passed by value or by reference (by preceding the name of the parameter with the symbol "&"). Each template has its own private declaration space to define clocks, variables, ...

The ability to define several processes for a time automaton is very useful since it allows to run several automata side by side with synchronizations between them. Indeed, without the system of processes, it would be necessary to do manually a concurrent composition of the automata in a unique automaton. But with the system of processes, each automaton is defined as a single process and their composition is done on-the-fly by the model-checker.

Process synchronization

In Uppaal, we can specify on the switches two kinds of process synchronization: the *channel* and *broadcast channel* synchronizations. With a channel Z, a switch emitter of a synchronization on Z can be fired only if there is another switch receiver of the synchronization on Z and which can be fired simultaneously. For a broadcast channel, the principle is the same, but several receiver switches can be fired for one emitter switch fired on the broadcast channel. Moreover, unlike for a synchronization channel, a broadcast channel is not blocking (if there is no receiver switch which can be fired, the emitter switch can still be fired). There are several constraints and rules over synchronizations, including:

- Only one synchronization per switch.
- No clock constraint on a switch which receives a broadcast synchronization.
- The update operations of the switches are made after the synchronization, starting with the operations associated to the switch emitting the synchronization.

In Uppaal, the emission of a synchronization over a channel Z is defined by `Z!`, and the reception of a synchronization is defined by `Z?`.

Committed locations

With the processes, Uppaal allows the specification of *committed location*. A committed location is a location which has to be quited immediately, even before urgent locations. When several processes are each in a committed location, the next switch fired in the automaton is chosen among these processes. The committed locations are useful in the sense that they create restrictions on the automaton execution and thus reduce the size of the state space.

3.3.4 Graphical notation

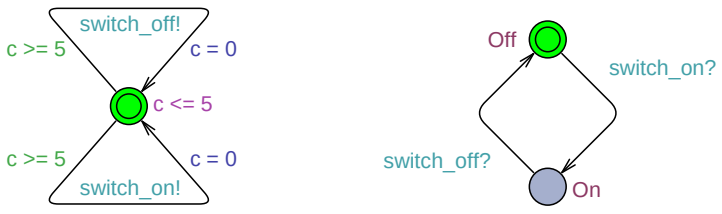
Since all the timed automata employed in the thesis are made for Uppaal, we will use henceforth the Uppaal graphical notation to represent them. This one differs from the standard notations in a few aspects.

Initial locations and final locations: Unlike the normal representation which distinguishes initial locations and final locations with an orphaned input arrow and with a nested circle, Uppaal uses the nested circle for initial locations. For the final locations, there is no graphical distinction since these locations are not necessary. We extend the notation of initial locations by setting their color to green.

Urgent locations and committed locations are annotated with a nested 'U' and with a nested 'C'. We extend this distinction by setting a yellow color to the urgent locations and a white color to the committed locations.

Labeling: each location and switch are labeled with visual constructs as follows

- **For a location**: the name (light purple) and the invariant (dark purple).



(a) Template controller **(b)** Template light

Figure 3.4: An Uppaal timed automaton with two instantiated templates, the controller and the light. The controller switches the light on/off every five time units. The signature of the two templates are: **light**(chan &switch_on, chan &switch_off) and **controller**(chan &switch_on, chan &switch_off).

- **For a switch**: the guard (green), the synchronization (light blue) and the update (dark blue).

3.3.5 Model checking

Finally, we briefly introduce the model checking here and the Uppaal capabilities in this domain. A model checker is a software which makes some verifications about a model in order to confirm or contradict an assertion (or request) about it. In the case of a timed automaton, an assertion can be "*The location L can be active when the clock Y is bigger than C*" or "*The variable V is always lower than 10*". To validate such assertions, the model checker must explore the state space of the automaton to find a situation which infirms or confirms the assertion. Sometimes, this entails the exploration of all the state space (like for the second assertion).

In Uppaal, the state space is handled in a symbolic form to minimize its size. In a symbolic state space, a symbolic state has a valuation of the clocks in a symbolic form and stands for several concrete states (i.e., normal states). Which such a system, an infinite state space can be sometime reduced in a finite symbolic state space.

As already said, the decidability of the reachability problem is very important to ensure the decidability of the assertions. For example, the specification of stopwatches in an automaton reduces the ability of the model checker to give a response since the reachability problem is undecidable with them.

To specify an assertion, the language provided by the Uppaal model checker uses the quantifiers of the classical temporal logic [8]. An example of assertion with such a quantifier is `[E<> P.L && C > 10]` which states that a location L (of the process P) is active at least one time with the clock C over the value 10. The temporal quantifiers of the Uppaal model checker are:

- `E<>` **p** *possible* assertion stating that there is a state with the property p in the state space.
- `A[]` **p** *invariant* assertion stating that for all states in the state space we have the property p.
- `E[]` **p** *potentially always* assertion stating that it is possible that the property p is always true. i.e., there is a path in the state space where each state of the path has the property p.
- `A<>` **p** *eventually* assertion stating that for each possible path in the state space

there is a state inside with the property p.

- $\mathbf{p} - - > \mathbf{q}$ *lead to* assertion stating that in each possible path in the state space, if there is a state with the property p, then follows (not necessary directly) in the path a state with the property q.

It is also possible to invoke the deadlock state property (no more switches can be fired) for a location L of the process P by invoking *P.L.deadlock*.

The result of Uppaal to an assertion can be "maybe" (if undecidability), "yes", or "no", with for the latter two an example or counter example trace representing a path in the state space (in a concrete or symbolic form).

3.4 Conclusion

In this chapter, the timed automata formalism was presented with, in addition, its use in Uppaal, an efficient and well-supported model checker for discrete timed automata.

Basically, timed automata are theoretical machines dedicated to the validation of timed words according to a timed regular grammar. But timed automata, with their model checking possibilities (e.g., the reachability property), are also often used for the modelling and the verification of critic systems where the time and concurrency aspects play an important role. For these reasons, the modelling of GRNs with timed automata can be an interesting alternative to the precise but heavy modelling with the ODE formalism.

Now that we have the necessary materials to understand GRNs and to handle modellings with timed automata, we can explore this alternative in the next chapters with, in each one, a different modelling approach of GRNs with timed automata.

4

The timed-boolean approach

This chapter present a first approach stemming from Thomas's formalism (see Section 2.3.2) to model GRNs with the discrete time automata formalism. The theoretical background of this approach is developed in [23] by René Thomas and Richard D'Ari, and is implemented with timed automata in [1, 6, 19], with, for each of these implementations, only slight differences. This chapter is then mainly a descriptive one depicting an existing approach with however, at the end, a section with several observations about it.

4.1 Principles

A timed-boolean model is a boolean model leaning on an extension of Thomas's formalism with a timed aspect. This timed aspect imposes a delay to wait before each change of activity level for a gene product. So, in the GRN dynamics, the transition from one activity level to another is now conditioned by these delays, preventing some evolution of the GRN which are allowed by the untimed Thomas's formalism.

More precisely, the extension is the following. In each gene of a boolean network, for each activity level of the gene product, two delays are defined. One delay that the gene product has to wait before decreasing the activity level of one unit, and another delay that the gene product has to wait before increasing the activity level of one unit. To determine when the delays are elapsed, a clock is associated to each gene. The clock of a gene is reset when the gene product changes its activity level or when it begins to wait the defined delay before a change of activity level.

With this extension, a gene product is considered to be in an intermediate activity level

if it is waiting for an increase or a decrease of its activity level. A gene product which waits for nothing is considered to be in a regular activity level. A GRN is considered as being in an intermediate state if it has at least one gene product in an intermediate level, otherwise, it is considered as being in a regular state. The intermediate states are new states which extend the transition state space. This last, with the temporal constraints on the new transitions and on the new intermediate states, can be defined as a timed automata.

This extension of the Thomas's formalism is stated formally hereunder. The parts of the formalism unchanged are not restated and remain the same as in Section 2.3.2. Note that as in this latter Section, we simplify the formalism by associating the activity level of a product to its gene.

4.1.1 Delay extension

The topology $\Gamma = ((V, E), sgn, tsd, r)$ of a GRN is extended with

- for each gene $\alpha_i \in V$, the set $Inter_i$ of intermediate activity levels, where $Inter_i = Inter_i^- \cup Inter_i^+$ and
  - $Inter_i^+ = \{l^+ \mid l \in \mathbb{N}_0 \wedge 0 < l < r(\alpha_i)\} \cup \{0^+\}$.
  - $Inter_i^- = \{l^- \mid l \in \mathbb{N}_0 \wedge 0 < l < r(\alpha_i)\} \cup \{r(\alpha_i)^-\}$.
- a function Δ: $V \times \Omega \to \mathbb{N}_0 \times \mathbb{N}_0$ where $\Omega = \bigcup_i Inter_i$ for all $\alpha_i \in V$ and such that $\Delta(\alpha_i, l) = (a, b)$ with $a \leq b$ for each gene $\alpha_i \in V$ and for each intermediate activity level $l \in Inter_i$, .

We denote a gene α_i with the activity level l, l^+ or l^- by α_{il}, α_{il}^+ and α_{il}^-. We denote the value a and b from $\Delta(\alpha_i, l) = (a, b)$ by respectively $\delta_{i,l}^{min}$ and $\delta_{i,l}^{max}$ (with $l \in Inter_i$), $\delta_{i,l}^{min}$ and $\delta_{i,l}^{max}$ being the bounds of the delay interval required by the gene α_i to achieve its transition of activity level (see below).

The set of all activity levels for a gene α_i is denoted AL_i. The set Reg_i of regular activity levels for a gene α_i is $AL_i \setminus Inter_i$. A state e (denoted $state_e$) of the GRN (with n genes) is a tuple of gene levels $(k_1, ..., k_n)$ where each $k_j \in AL_j$. The $state_e$ is called *intermediate* if there is at least one intermediate activity level inside ($\exists i \in \mathbb{N}_{[0,n]} : k_i \in Inter_i$), otherwise it is called *regular*. The set of all the possible states is denoted as $states_\psi = \{state_1, ..., state_m\}$.

Since we have $l(\alpha_{il}) = l(\alpha_{il}^+) = l(\alpha_{il}^-)$, the definition of a resource for a gene remains unchanged.

4.1.2 Delay-extended transition state space

Timed automata

Since the topology of a GRN N is extended with a timed aspect in term of delay values, we can refine and express the transition state graph S_N by means of a timed automaton where

- The set V of locations is the set of states $\{state_1, ..., state_m\}$ where each location corresponding to an intermediate $state_e$ has an invariant $\forall i : 1 \leq i \leq n : (k_i \in Inter_i) \Rightarrow (c_i \leq \delta_{i,k_i}^{max})$, with k_i being the activity level of the gene α_i in the $state_e$.
- The set X of clocks is $\{c_1, ..., c_n\}$, where each clock c_i is associated with a gene α_i.
- The set of switches E is composed of two subsets: the delay switches and the condition switches as defined below.

The use of discrete values for the delays results in a discrete timed automaton.

Note that the words *location* and *switch* - that we use preferably with automata than *state* and *transition* - are also synonym here for the words *state* and *transition* in the context of the extended transition state space.

Delay switches

The delay switches are switches which update an intermediate activity level of a single gene into the adequate regular activity level when the delay is elapsed. There are two types of delay switches:

- **Increasing**: $(k_1, ..., k_t, ..., k_n) \xrightarrow{(c_t \geq \delta_{tk}^{min}) \setminus \{c_t\}} (k_1, ..., l(k_t) + 1, ..., k_n)$ where $k_t \in Inter_t^+$.
- **Decreasing**: $(k_1, ..., k_t, ..., k_n) \xrightarrow{(c_t \geq \delta_{tk}^{min}) \setminus \{c_t\}} (k_1, ..., l(k_t) - 1, ..., k_n)$ where $k_t \in Inter_t^-$.

where $\xrightarrow{G \setminus C}$ is a switch with a guard G and which resets the clock of C.

As we can see, each delay switch changes the activity level of only one gene, what is an asynchronous mode for the point of view of the temporal aspect.

Condition switches

Condition switches are switches which change the activity level of some genes according to the activity level of other genes. They are defined as switches of the form

$$(k_1, ..., k_n) \xrightarrow{\setminus C} (q_1, ..., q_n)$$

where $\forall i \in \mathbb{N}_{[0,n]}$:

$$q_i = \begin{cases} l_i^+ \text{ if } K_{\alpha_i, Re(state_e, \alpha_i)} > l \text{ and } k_i = l_i \\ l_i^- \text{ if } K_{\alpha_i, Re(state_e, \alpha_i)} < l \text{ and } k_i = l_i \\ l_i \text{ if } K_{\alpha_i, Re(state_e, \alpha_i)} \leq l \text{ and } k_i = l_i^+ \\ l_i \text{ if } K_{\alpha_i, Re(state_e, \alpha_i)} \geq l \text{ and } k_i = l_i^- \end{cases}$$

with all $k_i, q_i \in AL_i$, and with $\forall i : (q_i \in Reg_i \wedge q_i \neq k_i) \Leftrightarrow (c_i \in C)$.

As several genes can have their activity level changed in a single condition switch, the condition switches execute in a synchronous mode. Note that it is necessary to set urgent each location with outgoing condition switches and without outgoing delay switch. This setting prevents the time automaton to stay in locations without clock invariant for infinite time despite the fact that a change of activity level is required.

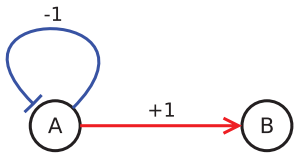
4.1.3 Example

Figure 4.1 gives an example of GRN with its topology and its logical parameters. This example is a simplified version of the one of Figure 4.1a where the threshold of the self-inhibition of A is set to 1. This modification induces a maximum activity level for A of one instead of two. The extended state transition graph of this example is the timed automata of Figure 4.2.

4.1.4 Modes

The collapsed transition state space is the one obtained by merging the intermediate states to their corresponding regular state. Formally, the corresponding regular state of an intermediate state $(k_1, ..., k_n)$ is the one obtained from $(l(k_1), ..., l(k_n))$.

In Figure 4.3, we can see in (a) the extended transition state space of the example of Figure 4.1 without the time constraints. In (b), we can see the corresponding collapsed transition state space, and in (c), the normal transition state space according to the Thomas's formalism. If we compare the normal transition state space of the Thomas's formalism (4.3c) with the collapsed one (4.3b), we can remark that several



(a) Topology

$K_{A,\{\}} = 0$
$K_{A,\{A\}} = 1$
$K_{B,\{\}} = 0$
$K_{B,\{A\}} = 1$

(b) Logical parameters

Figure 4.1: Example of a GRN with its topology (a) and its logical parameters (b). The delays associated with this GRN are $\delta^{min}_{A,0^+} = 4$, $\delta^{max}_{A,0^+} = 5$, $\delta^{min}_{A,1^-} = 6$, $\delta^{max}_{A,1^-} = 8$, $\delta^{min}_{B,0^+} = 3$, $\delta^{max}_{B,0^+} = 4$, $\delta^{min}_{B,1^-} = 2$, $\delta^{max}_{B,1^-} = 3$. Over each regulation, the sign and the threshold of the regulation are indicated.

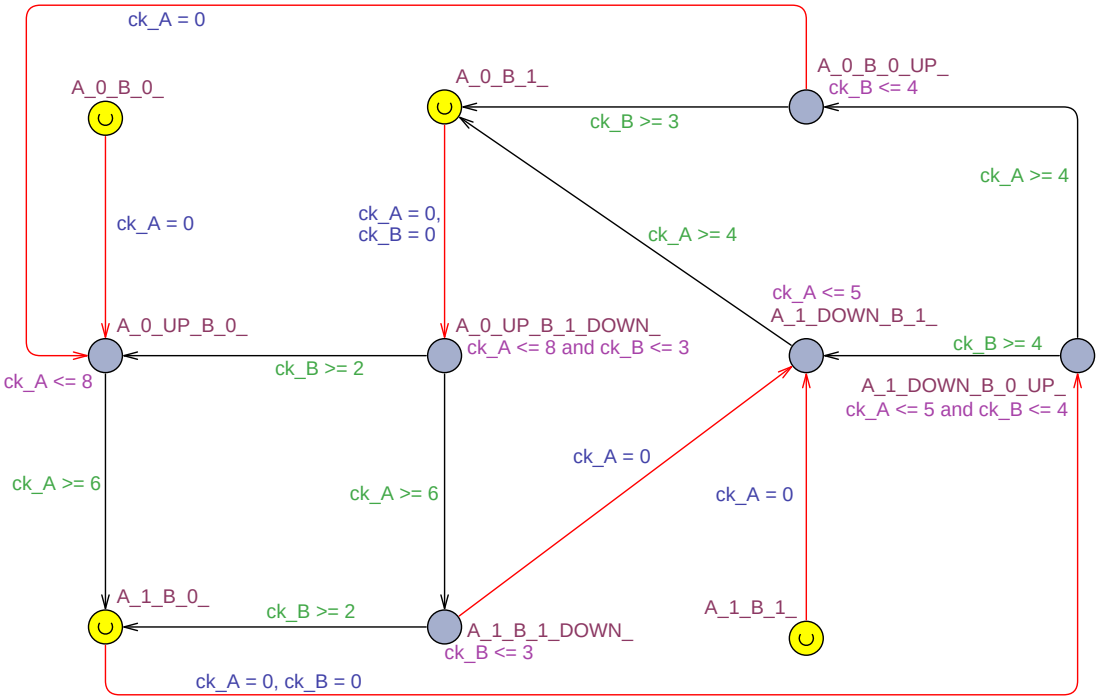
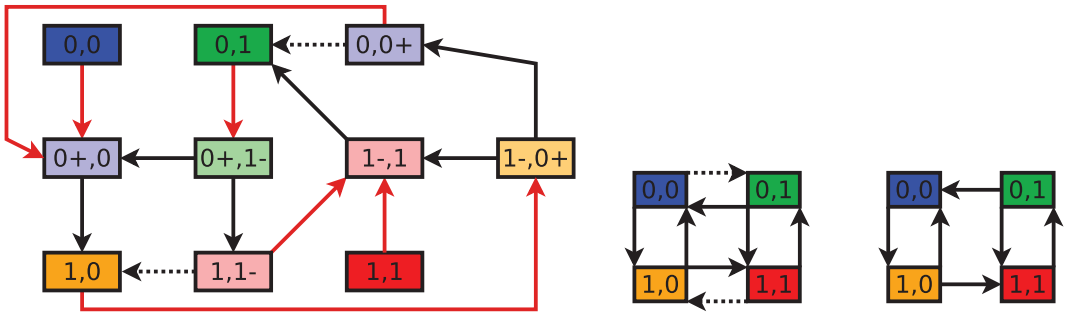


Figure 4.2: The timed automaton of the state transition graph for the GRN of Figure 4.1. *ck_A* and *ck_B* are respectively the clocks of A and B. The name of a state is the level of A followed by the level of B. The levels l^+ and l^- are represented by l_UP and l_DOWN. The condition switches are in red, the delay switches are in black. This timed automaton is not a functional version since the part with the initial location is not showed. Moreover, in the implemented version, the delays are saved in constant and a variable for each gene is present to save its activity level. These variables can be used in the model checker assertions.



(a) Extended transition state space

(b) Collapsed transition state space

(c) Normal transition state space

Figure 4.3: Different representations of the transition state space of the example of Figure 4.1. In (a), the extended state space without the delay constraints. The condition switches are in red, the delay switches are in black. In (b) the extended transition state space which is collapsed. In (c) the normal state space. The dotted transitions in (b) are the new ones. The dotted transitions in (a) are the responsible transitions for the new ones in (b). The states with the same kind of color correspond to the same regular state.

new transitions are possible (the dotted). These artifact transitions are the result of delay switches (the dotted ones in the extended transition state space) which lead to some improper changes of level from improper intermediate states. To solve these improper changes of level, it is necessary to leave each improper intermediate state by firing a condition switch before the firing of a dotted delay switch. To deal with this problem three variants or modes for the automaton are proposed in [19].

Normal mode

In this mode, the improper changes of activity level are tolerated without any restriction. The corresponding timed automaton (Figure 4.2) is left unchanged.

Urgent mode

The urgent mode allows an improper change of activity level only if it can be done immediately once in the improper intermediate state. This involves to set urgent the improper intermediate states in the timed automaton. In the example, these urgent states would be the state $(0, 0^+)$ and the state $(1, 1^-)$. With this, improper changes of activity level are prevented when the minimal delay of their corresponding switch is too long to allow the firing of the switch immediately.

This mode is maybe the most natural one regarding the real behavior of GRNs.

Overridden mode

Finally, the overridden mode deals with the problem by forbidding the firing of switches resulting in an improper change of activity level. The easiest way to embody this mode in the timed automaton is simply to delete the switches responsible for the improper changes of level (the dotted ones in Figure 4.3a).

4.2 Observations

In this section are presented some aspects and characteristics that we have observed on the timed-boolean approach. These aspects and characteristics follow those delineated in Section 2.4. As it can be expected, some of these characteristics and aspects come from the very nature of the Thomas's formalism.

Time aspects

In the timed-boolean model, the time is specified directly in the form of delays temporizing each change of product activity level in both directions (increase and decrease). This timed approach, proposed and discussed in [23], relies originally on three abstractions:

a) The delays are independent of the regulators.
b) The delays are independent of the history.
c) The delays remain constant.

Allowing very simple modellings, these abstractions are declared as naive for the following reasons:

With the abstraction a), the delay for a product to switch from one activity level x to the activity level $x+1$ is always constant, whatever the steady state activity level of the product is (i.e., the target activity level: $x+2$, $x+5$, ...), steady state activity level which is determined by the regulators through the logical parameters. But in the reality, a high (low) difference between the steady state concentration and the current concentration of a product comes with a high (low) rate of change in this concentration, entailing a fast (low) reactivity for the product to switch its concentration in a point toward this steady state concentration. In the modelling proceeded in the timed-boolean approach, this reactivity to switch the concentration is embodied in the delays which are constant, entailing an unrealistic constant reactivity for the products.

For the abstraction b), the non-consideration of the history, consider a case where a gene switches from a situation where it increases its product concentration, to a situation where its product concentration is stable. In the reality, the effect of the concentration increase depends on the times during which the gene stays in this situation, whatever that time. But in the timed-boolean approach, if the concentration increase does not last enough time to entail a change of activity level, the situation has no effect on the product activity level. This example underlines the problem of intermediate levels whose effect is lost if they do not last enough time. However, this problem can be smoothed by using more activity levels than allowed by the Thomas's formalism. Indeed, with more activity levels, an intermediate level with a long delay would be split into several intermediate levels with shorter delays and separated by regular levels.

For the abstraction c), the specification of constant delays ignores the biological noise. However, this last abstraction does no more apply for this approach since its implementation allows the specification of interval for the delays.

Finally, we can underline that with these abstractions on the delays coupled to the boolean abstraction (which imposes a limited number of activity levels), the size of the automata state space remains tractable. This fact allows the efficient use of algorithms on the automata to automatically determine which constraints the delays must respect to produce a given evolution of a GRN. This process, called the *parameters synthesis*, is followed in [1].

Activity levels

Since the approach relies on Thomas's formalism, we have activity levels for the gene products which are not linear to their concentration but which are linked to their effects as regulator. Since the modelling is boolean for the regulation, the maximal activity level of each product is constrained by its number of regulations provided with a distinct threshold.

The non-zero basal activity levels are - in the same way according to Thomas's formalism - defined with the specification of the logical parameters. For example, the logical parameter $K_{A,\{B\}} = 1$ indicates a basal activity level of 1 for the product A if B is its only inhibitor.

Gates

The only gates allowed in this approach are boolean gates. Their specifications are made with logical parameters. For example, if two regulators A and B promote a gene C (range $[0-1]$), an AND gate in C for these regulation is specified by $K_{C,\{\}} = 0$, $K_{C,\{A\}} = 0$, $K_{C,\{B\}} = 0$ and $K_{C,\{A,B\}} = 1$. With this system of logical parameters, there is no limitation on the boolean formula definable for a boolean gate.

Multi-effects regulations

According to the Thomas's formalism, the regulations cannot have several effects. Each regulation can be only active or inactive following its regulator activity levels and its associated threshold. However, Thomas's formalism can be extended to support the specification of several thresholds for a regulation. In such a case it is necessary to identify a threshold for each resource included in the definition of logical parameters. This extension is implemented in the software GINsim [15].

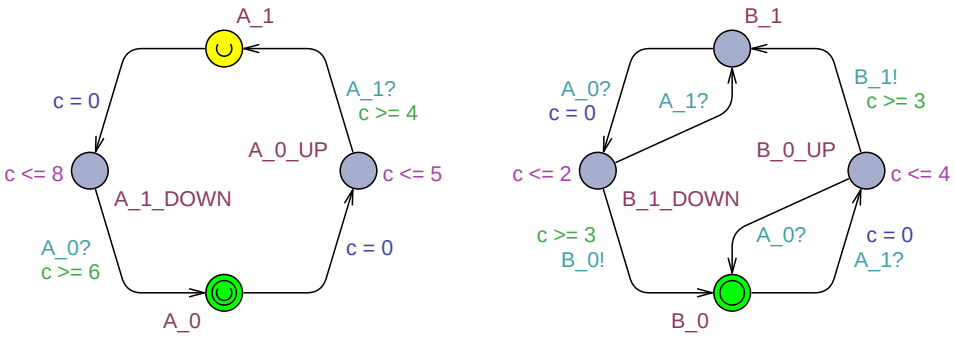
Degradation regulations

Since the effect of a regulation is only translated in the steady state level reached, there is no distinct way to handle the inhibition regulations and the degradation regulations.

Timed automata

The timed automata considered in this approach are very simple and do not require sophisticate constructions like many processes or functions. The only part of them which is outside of the original timed automata formalism is the use of invariants and urgent locations (which have no incidence on the decidability of the reachability property).

However, the simplicity of the implemented automata trades off the complexity of the construction. Indeed, the number of locations grows exponentially with the number of genes and activity levels. In the worst case, the addition of a single activity level to a gene product multiplies the number of locations by 3. The combinational computation - which can be done by Uppaal with the use of templates for each gene - is thus done before by the model-builder and entails very large automata. Therefore, the usage of the current implementation should be restrained to GRNs with a limited size.



(a) Template of gene A **(b)** Template of gene B

Figure 4.4: An implementation of the GRN in Figure 4.1 with the use of many processes. For each process template, a local clock c is declared. The element *A_0*, *A_1*, *B_0*, and *B_1* are broadcast channels embodying the *conditions switches*. In each template, the location declared initial (here *A_0* and *B_0*) determines the start activity level of the gene product. The urgent locations of the template of A are required by the self-inhibition of A.

For large GRNs, a different implementation of the extended Thomas's formalism can be found in [6, 19]. This implementation constructs automata where each gene has its own process. In that case, the regulation logic is no more translated into switches (*the condition switches*) between locations of the same process. Instead, they are embodied by synchronizations between the gene processes. Although entailing more work for the model checker, this choice can be fruitful since it prevents the generation of locations which can never be explored by the model checker. Figure 4.4 shows such an implementation for the GRN of Figure 4.1.

4.3 Conclusion

The timed-boolean approach described in this chapter finds its theoretical background in [23] of René Thomas and Richard D'Ari, and is implemented in timed automata in [1, 6, 19]. This approach relies on a simple timed extension of the logical model provided by the Thomas's formalism and described in Section 2.3.2. The abstraction of the approach allows, in its models, an easy definition of the time aspect and of complicated boolean cooperations between regulations. However, since the non-boolean aspects are ignored and since strong compromises on the time aspect are made, such an abstraction cannot be achieved without limiting the expressiveness of the approach in some situations. In such situations, as situation leaning strongly on SUM gates or on delays driven by regulations, the timed-boolean models can be inadequate and another approach can be required to deal with them. We describe such an approach in the next chapter.

5

The extended IKNAT approach

The second approach explored in this thesis comes from IKNAT. IKNAT (for Interactive Signalling Network Analysis Tool) is a prototype software developed by W.J Bos for his Master thesis at the university of Twente, in 2009 [10]. The purpose of the software is to allow biologists to model Interacting Signalling Networks (ISN) in Uppaal. ISN are biological networks which describe the dynamics of the interacting chemical substances. Inside a cell, these networks can find their roots in receptors ligands, and the resulting effect of an activation pathway can be, for example, the activation of transcription factors (the signal with respect to our point of view). IKNAT provides a graphical layer which easily allows the design of ISN. Once an ISN is designed, IKNAT generates a timed automata model of the ISN and submits it to Uppaal.

Since ISNs are, with their regulation networks, similar to GRNs in several aspects, we divert the IKNAT model from its original goal and we extend it to fit more to GRNs modelling.

In this chapter, we present this approach with first a mathematical timed model defined by ourselves for GRNs. This timed model can be used to explain and underline the mechanism of the IKNAT model. Then, the original IKNAT timed automata are described directly in the context of GRNs. Finally, we extend the IKNAT timed automata to enhance their abilities to model GRNs, and we underline some problems and observations related to the approach.

5.1 Principles

The idea behind IKNAT to drive the concentration level of a gene product can be formally exposed with what we call the timed model. The timed model is a model that we have defined from the piecewise linear ODE model introduced in Section 2.3.1. Therefore, the timed model sets up explicitly a link between the ODE model and the IKNAT approach.

5.1.1 Timed model

From the linear model stated in Section 2.3.1 where the time is implicit in the rates values, it is possible to easily obtain a model focused on the time aspect of a GRN.

Remember Formula 2.5 where the dynamics of a gene product Y regulated by n regulators is driven by a summation of linear rates delineated by threshold values:

$$dY/dt = \mathcal{B} + \sum_{k=1}^{n} \sum_{i} \beta_{ki} \Theta(R_k \geq t_{ki}) - \sum_{j} \alpha_j \Theta(Y \geq t_j)$$

where

- R_k denotes the concentration of the k^{th} regulator of Y.
- β_{ki} are the contribution rates of the R_k concentration $t_{ki} - t_{k(i-1)}$ to the influence of R_k (with $\forall i \in \mathbb{N}_{>1} : t_{ki} > t_{k(i-1)} > 0$).
- α_j are the dilution/degradation rates contributing for the Y concentration $t_j - t_{j-1}$ (with $\forall j \in \mathbb{N}_{>1} : t_{Yj} > t_{j-1} > 0$).
- $\mathcal{B}$ is the basal production rate of Y.

This model is closely related to the ODE models from which it is derived. However, it is possible to get easily from it a more abstract and useful result for IKNAT. For this, we can leave out the $\mathcal{B}$ value (for IKNAT) and we can specify that the β and α values are not rates anymore but are in fact the reciprocal of the delay ($1/delay$) expected before a change of the Y concentration level (i.e., discretized concentration in $\mathbb{N}$), with an use of negative β values for inhibiting regulations. The summation result of the activated β and α values (i.e., those with $\Theta(...) = 1$), if positive, is the reciprocal of the delay to wait before a Y increase of one concentration level, and likewise for a Y decrease if negative. This evolution, which changes the semantics and the role of the parameters in the piecewise linear ODE model, results in the timed model.

Timed model definitions

For a product Y with a range of concentration level from 0 to m_Y and regulated by n regulators $\{R_1, ..., R_n\}$ with for each regulator R_k of the set, a maximal concentration level m_k, the delay before a change of concentration level for Y is computed by

$$\delta_Y = \sum_{k=1}^{n} \sum_{i=0}^{l_k} \beta'_{ki} - \sum_{j=0}^{l_Y} \alpha'_j \tag{5.1}$$

where

- $|\delta_Y|$ is the reciprocal of the computed delay before the increase (if $\delta_Y > 0$) or decrease (if $\delta_Y < 0$) of the Y concentration level by one level if the history of Y is not taken into account.
- l_X denotes the current concentration levels of the regulator X (with $l_X \in \mathbb{N}_{[0-m_X]}$).
- α'_j is the reciprocal of the time contributing for the j^{th} concentration level of Y to the decrease of Y.
- $|\beta'_{ki}|$ is the reciprocal of the regulation time contributing for the i^{th} concentration levels of R_k. If R_k is an inhibitor, its β_{ki} values are set negative.

Note that in this formula, the $\mathcal{B}$ parameters is not conserved because IKNAT, in its original version, does not support the specification of a basal production rate.

To manage the evolution of the product Y according to the δ_Y computed and with respect to the history of Y, we use two clocks associated with Y. We denote these clocks by c_Y^{up} and c_Y^{down}. These clocks are constrained by the following rules:

- The clock c_Y^{up} is frozen iff $\delta_Y \leq 0$.
- The clock c_Y^{down} is frozen iff $\delta_Y \geq 0$.

At any time, two situations require an evolution of the Y concentration level as soon as they occur:

a) $c_Y^{up} - c_Y^{down} \geq 1/\delta_Y \geq 0$
b) $-(c_Y^{down} - c_Y^{up}) \leq 1/\delta_Y \leq 0$

In the situation a), the concentration level of Y is increased immediately by one level if $l_Y < m_Y$. In the situation b), the concentration level of Y is decreased immediately by

one level if $l_k > 0$. Finally, in both situations, the two clocks are reset to the value 0 and the updated δ_Y value is computed with Formula 5.1 according to the new concentration level of Y.

Whenever a Y regulator changes its concentration level, the δ_Y is recomputed according to the new concentration level of the regulator.

Note that since it is not the values of the clocks which are important but their difference, whenever δ_Y is equal (or diverge) to 0, it is also possible to leave both the clocks unfrozen. Indeed, in such a case the difference between the clocks stays unchanged, and since $1/\delta_Y$ diverges to infinity, neither situations a) nor b) can occur. Note also that by convention, we will use the notation ∞ to denote the reciprocal of parameters (α, β) without effect ($= 0$), although this notation is mathematically improper.

Finally, to set up the vocabulary used henceforth, we mean the α' and β' parameters when we speak about the dilution/degradation parameters and the regulation parameters. We mean the values $1/\alpha'$ and $1/\beta'$ when we speak about the positive/negative time/delay of these parameters. And we will use no more the prime symbol to refer to the α and β parameters of the timed model.

Example

Figure 5.1 shows an example of GRN where a gene X activates a gene Y and where both gene products have a maximum concentration level of 2. Table 5.1 gives the timed parameters concerning Y. Table 5.2 gives the resulting delays for Y computed with the timed model Formula 5.1 which results for our example in:

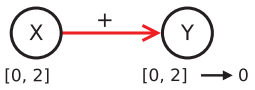


Figure 5.1: Example GRN

$$\frac{1}{\beta_{X1}\Theta(l_X \geq 1) + \beta_{X2}\Theta(l_X \geq 2) - \alpha_1\Theta(l_Y \geq 1) - \alpha_2\Theta(l_Y \geq 2)}$$

CL	$\mathbf{1/\alpha_j}$	$\mathbf{1/\beta_{Xi}}$
0	∞	∞
1	15	13
2	6	4

Table 5.1: Time values specified for the gene Y and its promotion in the example Figure 5.1. CL stands for Concentration Level.

X\Y	0	1	2
0	$\pm\infty$	-15	-6
1	13	97.5	-6.39
2	4	3.84	10.68

Table 5.2: Time values ($1/\delta_Y$) computed for Y according to the concentration level of Y (horizontally) and X (vertically) and following the values of Table 5.1.

According to Table 5.2, when the concentration level of X is 1, there is a cycle of 103.89 time units for Y between the concentration level 1 and 2. But when the concentration level of X is 2, Y has only positive delays and thus ends to be stuck to its maximal concentration level 2 (with loops of 10.68 time units).

IKNAT and the timed model

As said before, the timed model described and exemplified hereinbefore is the subjacent principle of the IKNAT modelling approach. Subjacent because IKNAT does not follow this model directly, but instead employs a mechanism of events that we describe hereunder.

5.1.2 IKNAT implementation

The way followed by IKNAT to implement the timed model with discrete timed automata uses on the one hand a mechanism of events and on the other hand a specification requirement.

Up and down events

In IKNAT, part of the time model is implicitly embodied in the use of a mechanism producing up and down events on genes. An up (down) event is an event which augments (diminishes) by one level the product concentration level of the targeted gene. For each gene, a process for the dilution/degradation of its product is created, and for each regulation of this gene, a regulation process is created. The up events for a gene are generated from the processes associated with the positive regulations of the gene. The down events are generated by the processes associated with the negative regulations of the gene and by the dilution/degradation process of the gene. Each process has its own clock and produces the events according to specific delays. At a given moment, if a gene receives one down event from its degradation/dilution process and two up events from two positive regulation processes, the resulting behavior of the gene product is an increasing of its concentration level by one level (see Figure 5.2 for an example).

For a gene, the equilibrium between the up and down events drives the dynamics of its product and corresponds in the timed model Formula 5.1 to the summation of the aggregated effect of each regulations R_k and of the aggregated effect of the degradation/dilution. But with this mechanism of event, each process need its corresponding parameters aggregated and stored as delays.

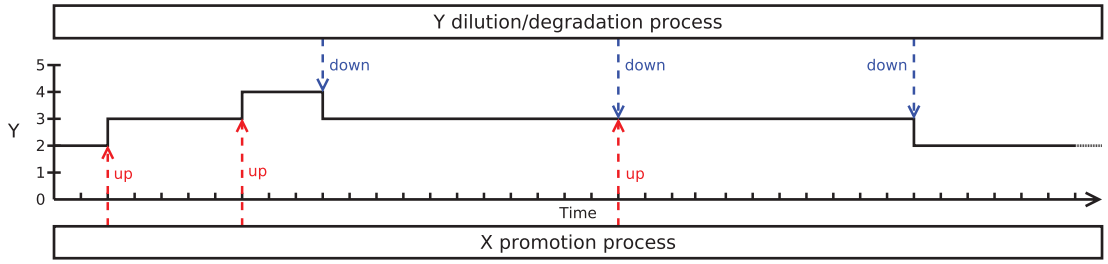


Figure 5.2: Consider the example Figure 5.1, but with a maximal concentration level for Y of 5. This chart gives the evolution of the Y concentration level (in abscissa) following the up and down events generated by the Y dilution/degradation process and by the promotion process of the regulator X. The start activity level of Y is 2.

Time specifications

For this last issue, IKNAT lets just the designer of the GRN make the aggregation himself and define directly the resulting values as α' and β' values, with $\alpha'_t = \sum_{j=0}^{t} \alpha_j$ and $\beta'_{kt} = \sum_{i=0}^{t} \beta_{ki}$. Since no more summation are necessary with this requirement, the aggregated values are directly inversed and stored as delay parameters in the processes.

Note that this implementation of the timed model is convenient to use with the discrete timed automata of Uppaal. Indeed, the only problems with the discrete aspect are left to the designer of the model when he specifies the aggregated delays ($1/\alpha'$ and $1/\beta'$) as discrete values.

5.2 Discrete timed automata

In IKNAT, the mechanism of up and down events is implemented in a structure of processes instantiating three kinds of template:

- The *species template* is instantiated by each process which represents the concentration level of a gene product and which receives the up and down events driving the concentration level of the product.
- The *dilution/degradation template* is instantiated by each process which generates down events for a gene according to the concentration level of the gene product and the delays specified for the dilution/degradation of the gene product.
- The *regulation template* is instantiated by each process which generates up or down events for a regulated gene according to the type of regulation, the concentration level of the regulator, and the delays specified for the regulation effect.

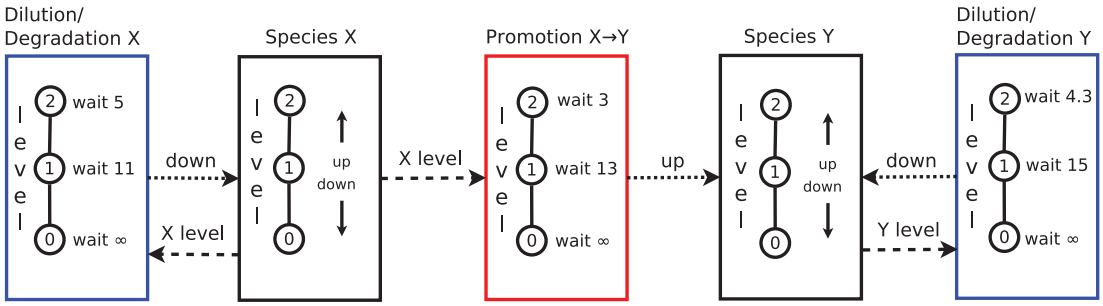


Figure 5.3: A chart with the templates instantiated (the boxes) for the example of GRN $X \to Y$ (Figure 5.1). In this example the regulation is a promotion and the maximum concentration level is 2 for both gene products. The blue box are processes producing down events. The red one produces up events. The dotted lines are channels and the dashed lines are broadcast channels.

In each template, a ladder of locations represents the different concentration levels of the related gene product. Following the current active location in the ladder of a regulation or dilution/degradation process, there is an associated delay to wait before the emission of a new event. In IKNAT, the number of concentration levels is the same for each product, allowing the instantiation of the same templates for the genes.

The event are conveyed through synchronization on channels between the processes (instantiation of the templates). In addition, broadcast channels are employed to communicate each change of activity level of one gene to the processes related to this gene (regulations and dilution/degradation processes). Figure 5.3 shows an example of templates instantiated for the GRN of Figure 5.1.

5.2.1 Species template

A species template is shown in Figure 5.4. This template is dedicated to genes with a maximum concentration level 2 for their product. The ladder structure linked to the concentration level is here evident. The signature to instantiate this species template in a process for a gene is

Species(const int startState, chan &down, chan &up, broadcast chan &br[**3**], chan &ch[**3**])

where

- **startState** is the initial concentration level of the gene product,
- **down** is the channel conveying the down events for the gene,
- **up** is the channel conveying the up events for the gene,

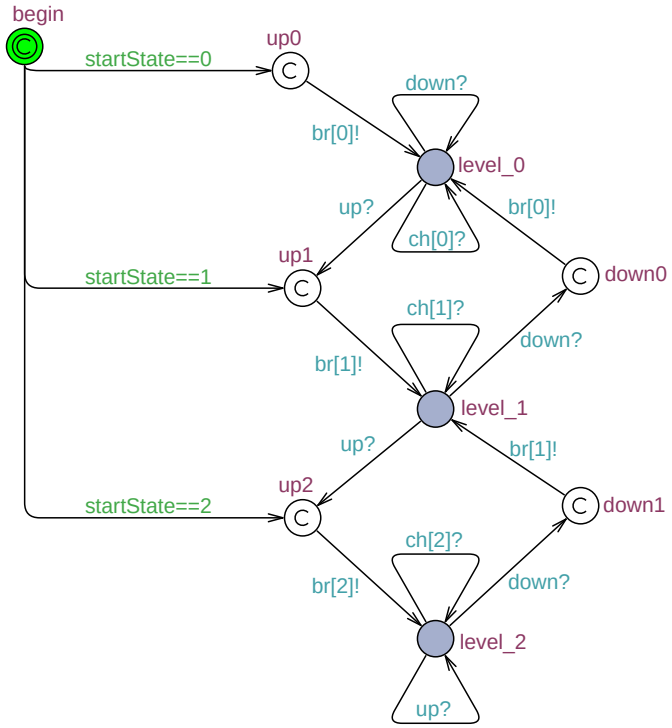


Figure 5.4: The template species for a maximum concentration level 2.

- **br**[] is the array of broadcast channels used to communicate a change of the concentration level to the depending processes. Each concentration level (0 included) has its own broadcast channel,
- **ch**[] is the array of channels employed to communicate on demand the current concentration level to the depending processes. Each concentration level (0 included) has its own channel.

In this process, each location *level_i* is active when the gene product has the concentration level i. With each change of concentration level, the process produces a broadcast synchronization on the channel dedicated to the new concentration level (*br[new_level]*). The committed location *up_i* and *down_i* are required since, in Uppaal, we cannot have two synchronizations on a same switch (here, the *up*/*down* synchronization channels and the broadcast synchronization channels *br[]*). Note that, at the initialization, the process passes by the location *up_i* to produce a first broadcast synchronization. Thanks to this broadcast synchronization, the other processes depending on the gene are informed of the start concentration level of the gene product. Finally, the synchronization channels *ch[i]* - on the loop switches in each *level_i* location - are employed to communicate on demand the concentration level.

5.2.2 Dilution/degradation template

Figure 5.5 shows a dilution/degradation template for genes with a product concentration level of maximum 2.

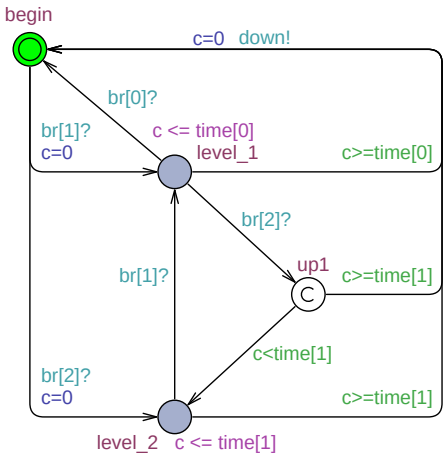


Figure 5.5: The template dilution/degradation for a maximum concentration level 2.

The element **c** in the template is a local clock. The signature to instantiate the templates in a process for a gene is

DilutionDegradation(chan &down, broadcast chan &br[**3**], const int time[**2**])

where

- **down** is the channel conveying the down events to the gene species process,
- **br**[] is the array of broadcast channels used to inform the process of a concentration level change in the gene product. Each concentration level (0 include) has its own broadcast channel,
- **time**[] is the array of delays waited before each generation of down event, according to the concentration level of the product. The values in the array has to be increasing or monotonic to avoid deadlock during a change of concentration level.

The ladder of locations following the number of concentration levels is here constituted by the *level_i* and *up_i* locations. Each *level_i* location is the location active when the concentration level of the gene product is i (except for the level 0). In these locations, the process waits the corresponding delay before producing a down event. The locations *up_i* are employed to prevent the invariant violation when the concentration level has

to increase. Indeed, in such a case, the new delay associated can be shorter than the former delay, and thus, it can violate the invariant of the new location if the clock c has a bigger time than the new delay. With the *up_i* locations, if such a situation occurs, the down event is fired immediately and the clock is reset. The location *begin* is the location used when the concentration level is 0. With such concentration level, the dilution/degradation process is "inactive" (it produces no event).

Note that the process relies only on the broadcast synchronizations (*br[]*) to be dispatched on the adequate *level_i* location, even if the change of concentration level is created by an down event coming from this process. Indeed, when the process produces a down event, the active location becomes the location *begin*. If the down event produces a change of concentration level, the species process will produce a broadcast synchronization for the new level, synchronization which will be received by the location *begin* to set active the adequate *level_i* location.

Finally, in order to improve the reality of the model, we have added on the original design the transition from the location *level_1* to the location *begin*. Indeed, the original template does not leave the location *level_1* if the concentration level of the gene product decreases to 0. Though a down event generated when the level is 0 produces no effect, if before the production of the event the product comes back to a concentration level 1, the clock c is not reset and the first down event occurs prematurely.

5.2.3 Regulation template

Finally, Figure 5.6 shows a regulation template for regulators with a maximum concentration level 2. The element **c** in the template is a local clock. The signature to instantiate the template in a process for a regulator and a regulated gene is

Regulation(chan &action, broadcast chan &br**[3]**, const int time**[2]**, chan &ch**[3]**)

where

- **action** is the channel conveying (when fired) up or down events according to the type of regulation,
- **br**[] is the array of broadcast channels used to inform the process with a change of concentration level of the regulator. Each concentration level (0 included) has its own broadcast channel,
- **time**[] is the array of delays to wait before each generation of event and according to the regulator concentration level. The values in the array has to be increasing

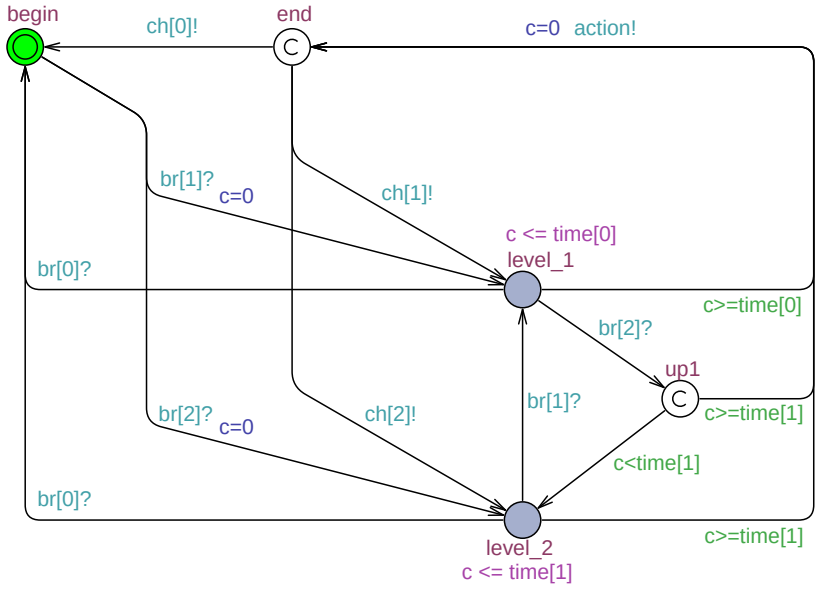


Figure 5.6: The template regulation for a maximum concentration level 2.

or monotonic to avoid deadlock during a change of concentration level of the regulator,

- **chan**[] is the array of channels used to dispatch the process in the right location according to the regulator concentration level.

This template is similar to the dilution/degradation template with however one difference. Here the target gene of the regulation, and thus of the generated events, is not the gene on which the delays of the template depend. For this reason, the production of a broadcasts synchronization on the channel *br[]* is not guaranteed after the production of an event. Therefore, after each fired event, the process needs to ask explicitly its regulator concentration level to its species process. This task is done through the location *end* and the synchronization channels *chan[]*.

Finally, note that this design and the one of the dilution/degradation template, which follow the original IKNAT design, can be simplified by not using the location *end* and *begin* after each fired event, and by dispatching directly the processes on the right location. This simplification is permitted by the committed property of the *upi* and *downi* locations of the species template which imposes to fire a broadcast synchronization (*br[]*) between each reception of a down/up event. This broadcast synchronization is then sufficient to dispatch the processes on the right location if a change has occured. The *ch[]* channels are also, with this, no more necessary. But unfortunately, this possible simplification was discovered to late and is not employed in the continuation of this thesis.

5.3 Extensions

In the current version of IKNAT designed originally to model ISNs, we can underline four features which can be added. The first two features are features which are desirable for GRNs and also for ISNs. They are the possibility to use a different maximum concentration level for each gene product and the possibility to specify intervals of time for the delays instead single time values. The last two features are important features lacking to model GRNs and, again, ISNs. They are the ability to get non-zero basal concentration levels and the possibility to use boolean gates. Note that for the latter feature, it is currently possible to specify different delays to obtain as a resulting effect a behavior similar to boolean gates. However, in practice this it is not so simple and it requires a complicated delay tuning in cases where several regulations are attached to a gate.

We have developed an extension of the IKNAT automata for each of these features. These extensions are described hereunder.

5.3.1 Different maximum concentration levels

The ability to use a different number of concentration levels for each gene product is simply resolved by allowing the creation and instantiation of each kind of templates for each maximal number of concentration levels.

5.3.2 Delays with interval

As for the previous extension, the possibility to define interval delays can also be easily and quickly implemented. The only required modification is the use of maximal delays in the bounds of the location invariants and the use of minimal delays in the bounds of the transition guards. With this, a location representing a given concentration level can stay active as long as the associated clock is under the maximal bound. And the location can be leaved as soon as the clock is over the minimal bound.

The modifications on the signatures required by this extension affects only the *time* array. This latter become a bi-dimensional array where the first dimension has the size of the original array and where the second dimension has a size 2 to store the interval bounds.

An example of a template with such intervals is presented in Figure 5.7 of the extension described below.

5.3.3 Non-zero basal concentration levels

The extension allowing the use of non-zero basal concentration levels (or non-zero basal production rate) remodels the dilution/degradation template in a *species-activity template*. A process of this template produces up events when the basal concentration level is smaller than the current concentration level. When the current concentration level is the basal level, the process stays inactive. Finally, when the current concentration level exceeds the basal concentration level, the process produces down events. The species-activity template, as shown in Figure 5.7, is a concatenation of two dilution/degradation templates. The first produces the down events and the second, which is a horizontal symmetry, produces the up events.

The only difference for the signature except the name, is the necessity to provide the template with the channel for the up events. As for the different maximal concentration levels, for each different basal concentration level, a dedicated template must be created.

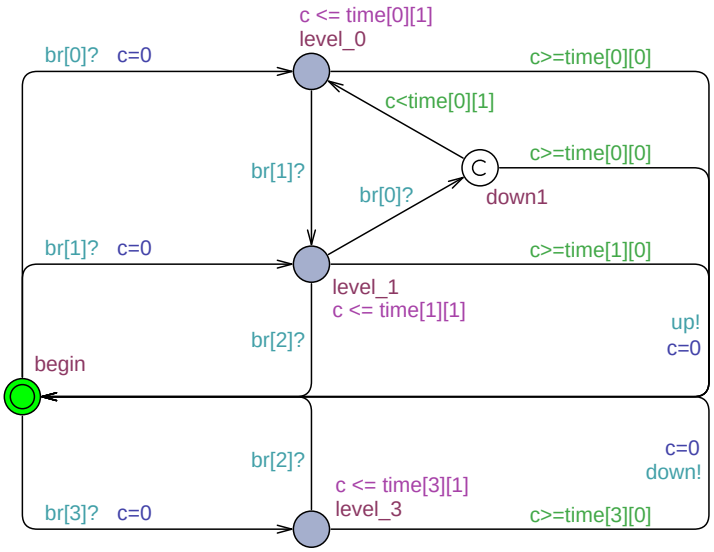


Figure 5.7: A template species-activity for genes whose the product has a maximum concentration level 3 and a basal concentration level 2. This template has also the extension which allows interval delays.

5.3.4 Boolean gates

Finally, the last extension allows the explicit creation of boolean gates in term of dedicated processes. A process for a boolean gate is a process which surveys the concentra-

tion levels of the input regulators. When the boolean formula over their concentration level is true, the process produces up or down events according to the sign of the gate and according to the delays specified. A template for such a process is presented in Figure 5.8 for an AND gate on two genes Y (threshold 8) and Z (threshold 5) which regulate the gene X. The gate produces down events with delay inside the range $[5, 8]$. Such a template is not reusable for the process of another gate. We do not have then to worry about any signature.

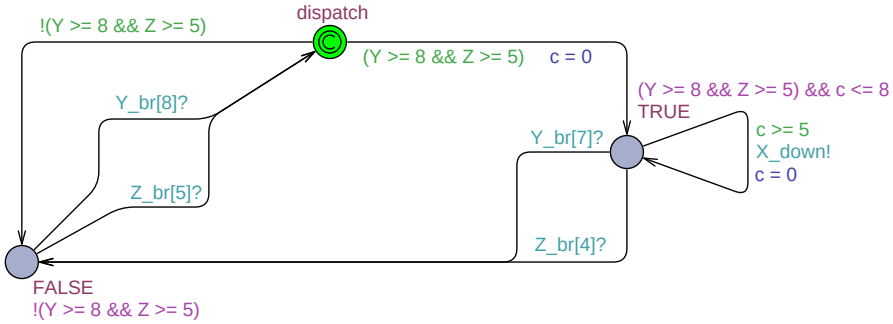


Figure 5.8: The template of an AND gate which promotes a gene X with two regulators, Y (threshold 8) and Z (threshold 5), and with a delay in the range $[5, 8]$.

The template of this boolean gate has two main locations, a location *TRUE* when the result of the gate is true and a location *FALSE* otherwise. The location *dispatch* is used to verify the logical result when one of the regulators change its concentration level around the threshold. Since it is an AND gate, the verification has only to be done from the *FALSE* location. The variables used are the following:

- The integer variables Z and Y are global variable which store the current concentration level of the corresponding regulators. This requires that when Z or Y change their concentration level, the process species updates its variable.
- *br_Z[]* and *br_Y[]* are arrays of broadcast channels employed to notify a change in the concentration level of the corresponding regulators. The process has to survey only the changes of concentration level near the thresholds value.
- The channel *X_down* communicates the down events to the species process of X.

In addition, the template has a local clock c used in the loop-switch of the *TRUE* location to produce the down events.

For boolean gate with a different configuration, the principle is the same. The difference

concerns the boolean formula, the location linked to the *dispatch* location, and the localization of the self-loop which produces events.

5.4 Stability problems

IKNAT, by its mechanism of up and down events and under certain conditions, suffers of an instability in the concentration levels which can prevent its use in some cases where the result can be affected. This instability, which is not addressed in the thesis of W.J Bos and that we underline here, has two distinct causes coming from the regulation processes and from the extreme level with the dilution/degradation process (or the species-activity process). Additionally, when oscillations between two concentration levels are expected for a gene product in a simple configuration of regulation, an unwanted stabilization can occur.

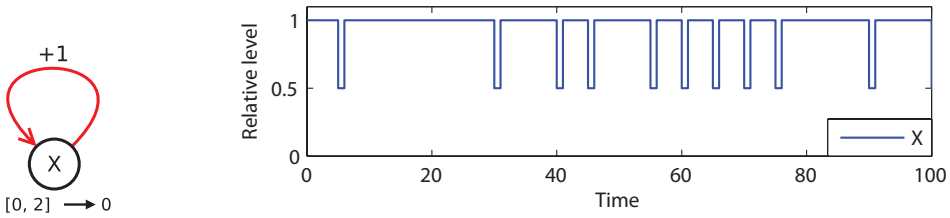
All these problems related to the stability are described in this section.

5.4.1 Extreme level instability

The first cause of instability is declared on the extreme levels with the species-activity process. Indeed, when the concentration level of a gene product is maximal and the gene is supposed to keep it maximal with the current regulations, the species-activity process still produces periodically a down event. These down events are quickly compensated by the up events of the regulations sustaining the gene product at its maximal concentration level. But by this game of infrequent down events followed quickly by up events, the gene product makes periodically short stay on the penultimate concentration level. These short stays can be seen as innocuous at first sight, but in fact they are sufficient to change the dynamics of a GRN if a threshold (of a regulation or a boolean gate) is set to the maximal concentration level.

Figure 5.9 shows a GRN which suffers of such an instability. In Figure 5.9a, a GRN where a gene X sustains its product concentration level with a self-promotion. In Figure 5.9b, a simulation of the GRN with IKNAT and with a start level 2. The delays parameters employed for the regulation process are five time smaller than the delays of the species-activity process. As we can see, the gene product supports short come back from the concentration level 2 to the concentration level 1. If, as it is required by boolean GRN, the maximum concentration level were 1, then the self-regulation would be deactivated at the first decrease of the concentration level.

Note that there is no a decrease of the concentration level at each down event of



(a) GRN **(b)** IKNAT simulation

Figure 5.9: In (a) a GRN and in (b) a simulation showing the instability associated to the extreme concentration levels.

the species-activity process in Figure 5.9b. This is a result of the non-determinism associated with the order of processing simultaneously received events. In the example, the regulation produces an up event at each time unit. When a down event occurs before the up event, the change in the concentration level introduced by the down event is canceled immediately by the up event. If the event order is inverted, the up event has no effect since the gene product cannot reach a concentration level higher than the maximal level.

Extension

By modifying slightly the species-activity template, it is possible to avoid this instability on the maximal concentration level. The trick is, when a gene product is on its maximal concentration level, to reset the clock of its species-activity process for each up event received by its species process. This reset can be done by using a new channel synchronized for each up event received in the species template by the *level_i* location associated to the maximum concentration level. These synchronizations can be received on the location *begin* and on each of the locations *level_i* in the species-activity template. In these locations, the synchronizations are received with a loop-switch which resets the clock c of the species-activity template.

Evidently, where a non-zero basal concentration level is specified, the same kind of instability occurs at the concentration level zero. A symmetric solution can be used then.

5.4.2 Regulation instability

The second cause of instability finds its source in regulations. Indeed, when a gene is regulated, the concentration level of its product can make oscillations around the

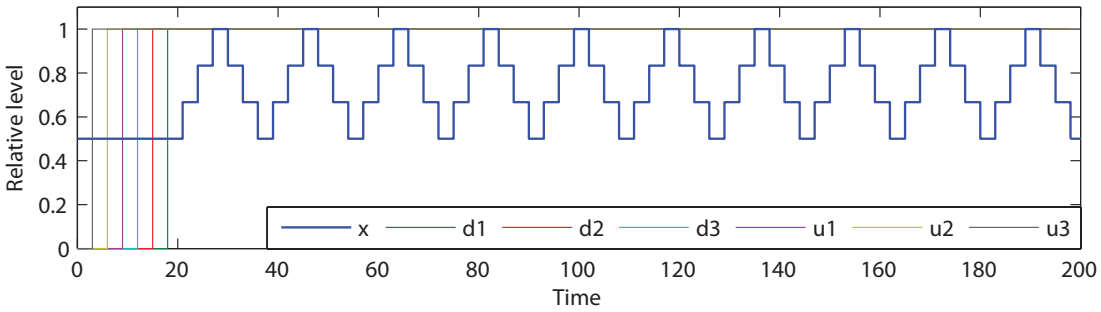


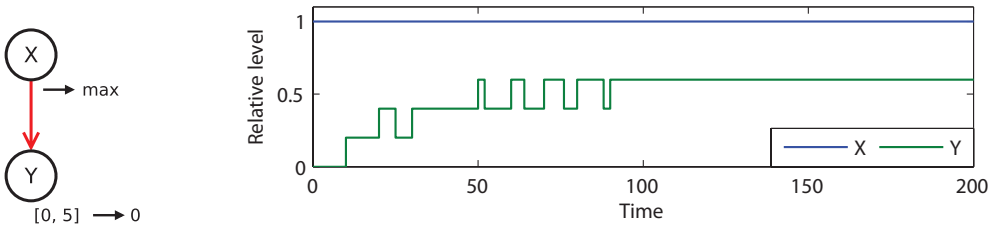
Figure 5.10: An artificial GRN tuned to produce exaggerated oscillations. x is a gene promoted by u1, u2, and u3 and inhibited by d1, d2, and d3. The range of concentration level for x is [0, 6], the delays for the dilution/degradation are ∞. As we can see, the oscillations occur between the levels 3 and 6, since the delays are set up to produce bunches of 3 ups followed by 3 downs.

expected steady state concentration level. These oscillations can have an amplitude of one if there is only one regulation. However, they can also have a larger amplitude with several regulations and following the order of the event arrival. For example, if a gene X is regulated by several promoters and several inhibitors, following the delays of the regulators, it can happen that the gene X receives the event by bunches of same type (e.g., 3 ups, 5 downs, 2 ups, ...) and therefore its product oscillates strongly around its theoretical steady state level. Such a situation is displayed in Figure 5.10 with an artificial example tuned to produce such a behavior in an exaggerated manner. In real models with realistic parameters, such strong oscillations will probably not occur, or only for a short time since the arrival frequencies of the events have little chance to stay the same for a long time.

Since the source of this instability comes from the mechanism events, which is the base principle of IKNAT, there is no easy solution to prevent it. At least the use of many concentration levels with no dense thresholds is advisable to damp the oscillation effects and to avoid as much as possible the cross of several thresholds during their occurrences.

5.4.3 Unexpected stabilization

With respect to the same subject, it is interesting to note that with a simple regulator configuration, in IKNAT, a gene product can tend to a stable concentration level whereas we expect from it oscillations between two concentration levels. Figure 5.11a shows such a configuration where X promotes continuously Y with a Y range of levels $[0, 5]$. The time for the species-activity process of Y is set such that, in the concentra-



(a) GRN **(b)** IKNAT simulation

Figure 5.11: An simple GRN (a) which produces an unexpected stabilization for Y.

tion level 3, Y increases and, in the concentration level 4, Y decreases, according to the regulation delays and the timed model. Despite the fact that there is no level in Y which equilibrates the regulation delay (i.e., $\delta_Y = 0$), Y is unexpectedly stabilized in the concentration level 3 and, moreover, never arrives to reach the concentration level 4 (Figure 5.11b).

This phenomenon stems from the way followed to handle the clocks. In the example, the regulation produces an up event after 10 time units. When Y reached for the first time the concentration level 4, the clock of its species-activity process has at least a value of 10. Since the delay associated with the level 4 in this process is 8, a down event is produced immediately and Y comes back to the concentration level 3. Subsequently, this sequence is repeated each time Y reaches the concentration level 4. The only way for Y to stay on the concentration level 4 is to begin the simulation with a concentration level bigger than 4. Then the situation is symmetrically inversed and Y ended to be stuck on the concentration level 4.

This unexpected stabilization is however presented only in very simple regulator configurations like the example of Figure 5.11a. With more regulations, the regulation instability is stronger and overwhelms this unexpected stabilization.

5.5 Observations

Similarly to the timed-boolean approach, we have made several observations about the IKNAT approach. Note that these observations are often linked to the timed model.

Time aspects

With the timed model which underlies the principle of IKNAT, the timed aspect drives the reactivity and the steady state concentration level of the gene products. It is more than in the timed-boolean approach where the logical parameters determine the steady state level and where the reactivity is constant. In the timed model, for each gene, these logical parameters are substituted by a concurrence between the active delay parameters. The equilibrium of this concurrency defines for the gene product its steady state concentration level and drives the reactivity to reach this one. Note that for the aspect of the reactivity, we can remind the link between the timed model and the ODE model with, between them, the step made with the piecewise linear ODE model (Section 2.3.1).

For the history of the gene evolution inside a same concentration level, the timed model (and thus IKNAT) does not suffer from the problem present in the timed-boolean approach. This problem was for example in the case of a situation where a gene waits to increase its product concentration level and then switches to a situation where it waits to decrease its. The time spent by the gene in the former situation is not taken into account in the required delay before a decrease of the concentration level in the latter situation. With the mechanism of two clocks by gene and their difference, the timed model does not suffer from this aspect. In IKNAT, the implementation of the timed model with the event mechanism avoids also this problem.

Finally, with the extensions of the IKNAT model, the delays can be expressed as intervals, giving here the ability to model the biological noise.

Concentration levels

In IKNAT, as in the timed model, we express the concentration of the gene products with concentration levels instead of activity levels as in the timed-boolean approach. These concentration levels can have an abstract meaning or can be linear to the concentration of their product (the concentration level 2 correspond to a product concentration two times bigger than the concentration level 1). And since they are not limited to the thresholds of the regulations as in the timed-boolean approach, their number can be arbitrarily large. It is even advisable to use large number of concentration levels to refine the result and more, to improve the robustness against the IKNAT instability problem. Note that for each gene, the scale associated to the concentration level of its product can be different to the one of the other gene products.

With the extensions proposed for IKNAT, it is possible to specify a non-zero basal

concentration level and a different maximal concentration level for each gene product. For each maximal concentration level used, a corresponding regulation and species templates are created, while for each different couple of maximal and basal concentration levels, there is a dedicated species-activity template.

Note that by leaving a location in the species-activity template where the process does not produce events, we impose for the gene products a stable concentration level. This seems natural for the implementation, but as stated before, we can envisage situations where the basal "concentration level" is in fact defined between two neighboring concentration levels, producing oscillation of the product concentration level between them where no regulator are active. This last situation could have been implemented by an independent template whose the process produces up events following the rate of the basal production rate. This implementation is theoretically more precise but in practice increases the number of events and thus the instability.

Gate

With its mechanism of up and down event implementing the timed model, the SUM gate is the implicit default gate of IKNAT. If it is possible by tuning properly the parameters to get a result somehow equivalent to the one obtained by the use of boolean gates, the extension for boolean gates allows the explicit and easy specification of such gates whatever is the complexity of their boolean formula.

For the MULT gate, if the production of the events were done probabilistically at each unit of time and not at the end of the delays, it would be somehow possible to define such a gate for x regulations. The gate would be implemented by a process requiring x events of the same kind and at the same time unit to produce another event. For example, consider that we have a situation where a gene A is regulated by two promoters B and C which produce each a new event after respectively the delay D_B and D_C. If the events were not produced at the end of the delays but at each unit of time with a probability of $1/D_B$ and $1/D_C$, the average amount of events produced by the two regulations would be preserved. In such a case, it would be then possible to create a process for the two regulations which, when it receives simultaneously an event of each regulation, produces another event for the gene A. The probability of producing an event for the process is then $1/D_B * 1/D_C$, which is equivalent to the rate computed in a MULT gate. However, since the events are not produced probabilistically but at the end of their delay, this solution is not applicable.

Multi-effects regulations

In the timed model, the number of intensity levels for a regulation is equivalent to the number of concentration levels of its regulator. By setting a regulator with several concentration levels, we get a regulation with as many intensity levels. Note that with this system, it is possible to emulate a boolean regulations from a regulator with more than one concentration level, whereas the reverse is not possible. The trick is to define infinite delays for the regulation levels below a given threshold level, and then set the same finite delays for the other levels.

Degradation regulations

In IKNAT, there is no difference in the modelling of degradation regulations and inhibitor regulations. Both have to be modeled by the production of events according only to the concentration level of their regulator. This non-distinction which is also present in the timed model, does not really fit to the ODE modelling where, unlike the inhibition regulations, the effects of the degradation regulations depend also on the concentration of the regulated gene products.

Timed automata

The timed automata generated in the IKNAT approach used abundantly the processes and synchronization channels provided by Uppaal. If these constructions keep the reachability problem decidable, they have a negative effect on the performance of the model checker by entailing a very big state space. However, these apart, the automata remain rather simple since they do not use the other high level specifications provided by Uppaal (functions, variables, ...).

5.6 Conclusion

Originally dedicated to ISNs, the IKNAT model of W.J Bos [10], that we have diverted and extended to model GRNs, provides an approach completely different from the one developed in the timed-boolean approach. Indeed, IKNAT can be related to the ODE model through our timed model developed at the beginning of this chapter. This link with the ODE allows the regulators to drive the reactivity without preventing the definition of logical gates indirectly (by tuning the delays) or directly (with the dedicated processes). This approach is thus less abstract than the timed-boolean approach where

this reactivity is fixed and where the definition of non-logical gate is not possible. However, beside these advantages, we have underlined the instability generated by the event mechanism and which can be the cause of artifacts in the simulations. The next chapter tackles this instability in a new approach developed directly from the timed model.

6

A new approach

The approach pursued with IKNAT in the previous chapter is very interesting since it allows the modelling of GRNs with logical gates, time intervals, and with reactivity driven by regulations. However, as we have underlined, the instability produced by the mechanism employed to deal with the timed model can be a problem in certain situations, especially situations where the thresholds play a prominent role. In a new approach that we have developed and that we present in this chapter, we try to show to what extent it is possible to implement the timed model more directly without using the IKNAT mechanism, and thus without getting into stability problems.

6.1 Principles

6.1.1 Timed model extensions

The IKNAT approach - with only slightly modifications of its templates - allows the specification of non-zero basal concentration level and time intervals. In this section, the timed model, created according to the original version of IKNAT, is extended to formally allow these possibilities.

Non-zero basal concentration level

To allow the specification of non-zero basal concentration level, we just need to use the basal production rate $\mathcal{B}$ of Formula 2.5. The Formula 5.1 computing δ is thus now for

the Y product extended in:

$$\delta_Y = \sum_{k=1}^{n} \sum_{i=0}^{l_k} \beta_{ki} - \sum_{j=0}^{l_Y} \alpha_j + \mathcal{B} \tag{6.1}$$

where the only difference with Formula 5.1 is $\mathcal{B}$, the reciprocal of the time contributing for the basal production rate of Y.

Time intervals

For this extension, each of the β and α parameters is defined with an interval delineated by β^{min}, β^{max} and α^{min}, α^{max}, with $\beta^{min} \leq \beta^{max}$ and $\alpha^{min} \leq \alpha^{max}$. The same for the $\mathcal{B}$ parameter with $\mathcal{B}^{min} \leq \mathcal{B}^{max}$. Note that with the constrains between the min and max values of the parameters, the β values associated with a negative value, to speak in terms of delays, are labeled for the longest bound with "min" and for the shortest bound with "max".

In the continuity of this distinction, the δ_Y value of the gene Y is now an interval delineated with δ_Y^{min} and δ_Y^{max}. This interval is computed simply by Formula 6.1 using the labeled "max" parameters for δ_Y^{max} and the labeled "min" parameters for the δ_Y^{min}. Therefore we have $\delta_Y^{min} \leq \delta_Y^{max}$ and four situations are possible:

1. Both bounds are positive: $(\delta_Y^{max} > \delta_Y^{min} \geq 0) \vee (\delta_Y^{max} \geq \delta_Y^{min} > 0)$.
2. Both bounds are negative: $(\delta_Y^{min} < \delta_Y^{max} \leq 0) \vee (\delta_Y^{min} \leq \delta_Y^{max} < 0)$.
3. One bound is negative and the other is positive: $\delta_Y^{min} < 0 < \delta_Y^{max}$.
4. Both bounds are equal to 0: $\delta_Y^{min} = \delta_Y^{max} = 0$.

With these results, the clock c_Y^{up} is frozen in the cases 2 and 4 and the clock c_Y^{down} is frozen in the cases 1 and 4. In the case 3, both clocks can be active but not at the same time. Thus, randomly, one is frozen while the other is activated.

We can now redefine the evolution of the Y concentration level for each situation as follows:

For the first situation, the Y concentration level is allowed to increase by one unit when $c_Y^{up} - c_Y^{down} \geq 1/\delta_Y^{max}$, and it has to increase by one unit if $c_Y^{up} - c_Y^{down} \geq 1/\delta_Y^{min}$. Note that with the multiplicative inverse of the δ_Y bounds, we have to interchange their use since $1/\delta_Y^{min} \geq 1/\delta_Y^{max}$. Note equivalently that when a bound converge (or is equal) to the value zero, the associated delay diverges (is considered as diverging) to the infinity and then is unreachable.

For the second situation, the Y concentration level is allowed to decrease by one unit when $c_Y^{down} - c_Y^{up} \geq |1/\delta_Y^{min}|$, and it has to decrease by one unit if $c_Y^{down} - c_Y^{up} \geq |1/\delta_Y^{max}|$.

The third situation, is slightly more complicated. Indeed, since the interval includes zero (not on a bound), the Y concentration level can increase or decrease by one unit. This one is allowed to increase by one unit when $1/\delta_Y^{max} \leq c_Y^{up} - c_Y^{down} < \infty$ and is allowed to decrease by one unit when $1/\delta_Y^{min} \geq -(c_Y^{down} - c_Y^{up}) > -\infty$. According to the way the clocks are managed, the bound with the lower delay is the most likely to be reachable first.

For the fourth situation, nothing is required. Both clocks are frozen and the product is not expected to change its concentration level.

Other aspects related to these situations remain the same: a change of concentration level for Y is processed only if the concentration level stays inside the bound $[0, m_Y]$. The clock are reset after each of the three first situations. And whenever a Y regulator changes its concentration level or whenever Y changes its concentration level, the δ_Y value is recomputed.

6.1.2 Implementation principles

The principles of the new approach are closely related to the extended timed model presented hereinbefore. Therefore they can be directly and quickly presented:

- Each gene y with a range of concentration level $[0, m_y]$ for its product is represented by a template with a ladder of m_y locations representing each a concentration level.
- Each gene y has two clocks declared locally in the template of y. One of the clocks is called cUp and represents the clock c_y^{up}, the other is called $cDown$ and represents the clock c_y^{down}.
- The location changes for each gene y are driven by the delay $1/\delta_y$ according to the situations described in the timed model extended with intervals.
- The genes are notified of a change in their regulators concentration level with a broadcast synchronization.
- When a gene y is notified by a change of concentration level of one of its regulator or when it changes its product concentration level, the associated δ_y value is recalculated.
- The product concentration level of each gene is stored in a global variable used to communicate this one to the regulated genes.

From these simple principles, two difficulties emerge. The first one is about the calculation of the $1/\delta$ values in the discrete automata environment and can be avoided by using a model checker for real automata instead of Uppaal. The second one concerns the management of the clocks under the limitations of the time automata formalism.

For each of these difficulties, two versions or variants for the templates of the genes are proposed. These variants are for the calculation difficulties the *pre-calculation variant* and the *on-the-fly calculation variant*. For the clock management difficulties, they are the *variant with stopwatches* and the *variant without stopwatch*. Note that the variant addressing one difficulty can be combined with any variant addressing the other difficulty.

The variants are described in Section 6.2 with further explanations in due course about the two difficulties. Section 6.3 is dedicated to the implementation of the gates.

6.2 Discrete timed automata

In the new approach, each gene of a GRN has its own dedicated template conceived following its regulators and its number of concentration levels for its product. Since a template for a gene is not convenient to show, we present first an abstract template, and then, for each variant of this one, we give the complementary part related to this variant. The template presented is for a gene Z with a maximal concentration level 2 and regulated by the genes X and Y.

6.2.1 Abstract automaton

The abstract template of the gene Z according to the example described hereinbefore is shown in Figure 6.1. Several transitions are colored to ease the explanations. The guards of these transitions and the invariants of the locations *level_j* (i.e., *level_0*, *level_1* and *level_2*) are not in the figure since they depend of the followed variant.

The ladder of location *level_j* represent the concentration level of Z. Besides these locations, there is an associated ladder of location *update_j*. The goal of these locations is to prevent the infringement of the *level_j* invariants when the configuration of the regulator changes and thus entails the recalculation of the delay ($1/\delta_Z$). Between each location *level_j* and its associated location *update_j*, and for each regulator of Z, there is a switch fired when the regulator notifies a change of its concentration level through its broadcast channel (here *X_br* and *Y_br*). These switches are in grey in the figure.

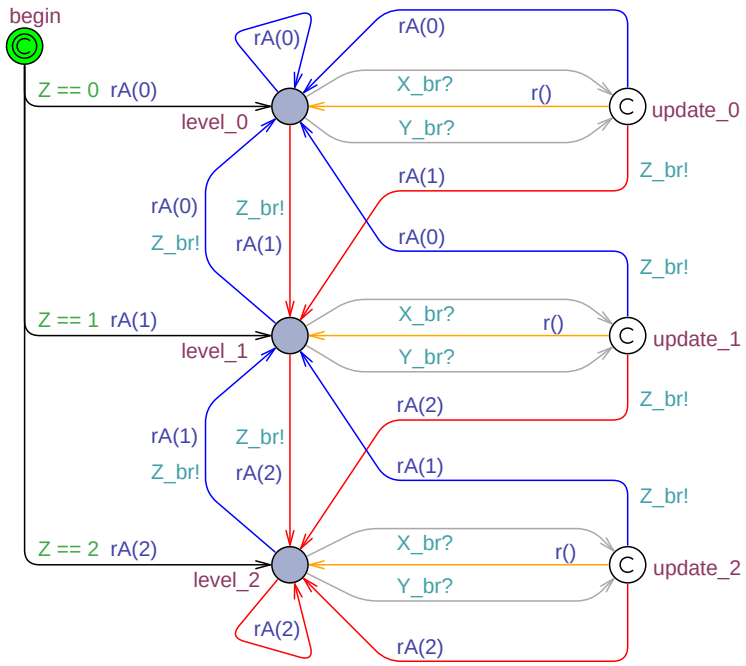


Figure 6.1: The common part of the variants for the Z template. Z is regulated by X and by Y. The maximal concentration level of Z is 2.

The blue switches are those which reply to a situation where the concentration level can or has to decrease. Similarly, the red ones are those for an increase of the concentration level. Finally, the orange ones are switches employed when, following a notification of a change in the concentration level of a regulator, no change of the Z concentration level is required.

When a switch changes the Z concentration level, the switch emit a synchronization on the broadcast channel *Z_br* to notify the change to the potential gene regulated by Z (if ever). The potential regulated genes access to the Z concentration level with the variable *Z*. This variable is also used to dispatch the template on the right location at the beginning, according to the start concentration level of Z.

The function *r()* and *rA()* are used to manage the clocks (reset, ...), the *Z* variable and the operations related to the δ_Z value according to the variant of the automaton.

Now that the common part of the template for all the variants is presented, we can introduce these variants with their particularities according to the difficulty addressed by them.

6.2.2 Calculation variants

These section is dedicated to the computational methods used to solve the problem of the δ_Z calculation in the discrete timed automata formalism. The first part of the solution displaces the $-\mathcal{B}$ value of Z to the parameters α_0 which is normally unused ($\alpha_0 = 0$).

The second part of the solutions let the designer make himself the the aggregation of the parameters of each regulations R_k and of the parameters of the degradation/dilution. Already used in IKNAT, this requirement avoids the computing of the sum $\sum$ on i and j in the formula 6.1. From this trick and the previous, each α'_t is redefined as $\sum_{j=0}^{t} \alpha_j - \mathcal{B}$ and each β'_{kt} is redefined as $\sum_{i=0}^{t} \beta_{ki}$.

The work of the automaton to compute the current delay $1/\delta_Z$ is then to make the summation of the active (i.e., where $i = l_k$) β'_{ki} values and the active α'_j value, and to multiplicatively inverse the result. To do this, two variants are proposed.

Pre-calculation in the model-builder

The first variant for the parameters calculation is called the *pre-calculation variant* because it simply pre-computes in the model-builder all the possible δ_Z values according to the possible configurations of the regulator concentration levels. The pre-computed values are then multiplicatively inversed (to get the reciprocal), discretized and stored in a multi-dimensional array directly as delays. A gene with n regulators has then a $n+2$ dimensional array of $1/\delta$ values where the n first dimensions are scaled each one on the corresponding concentration level range of its regulator. The last two dimensions are for the concentration levels of the gene product (for the dilution/degradation) and for the bounds of the interval computed.

To simplify the guards and invariants in the timed automaton, the negative δ_Z values are converted into positive values. To keep the distinction between the delays leading to an increase of concentration level and the delays leading to a decrease, we split the delay array in two arrays *time_up* and *time_down*. These arrays are declared locally in the gene template with unchanged dimensions and the value left empty by the splitting are replaced by an arbitrary large integer (denoted i_∞) which outlives the simulation duration. All values larger than i_∞ (especially when δ^{min} or $\delta^{max} = 0$) are set to this value.

For the example, in this variant, the current $1/\delta_Z^{min}$ value is directly accessible with *time_up[X][Y][Z][0]* and *time_down[X][Y][Z][0]*, where each *[...]* is the indice of a di-

mension in the array. The $1/\delta_Z^{max}$ value is accessible with *time_up[X][Y][Z][1]* and *time_down[X][Y][Z][1]*.

Note that in this approach, a self-regulated gene accounts for two dimensions in its arrays. We can merge these two dimensions in a single one since their indices (the concentration level of the gene product) have always the same value.

On-the-fly calculation in Uppaal

The second variant dedicated to the calculation of the delays is called the *on-the-fly variant* because it uses the mathematic expressions allowed in Uppaal to make the computation. Indeed, the $1/\delta_Z$ values are computed on-the-fly by the automaton when needed. To do this, the aggregated parameters (α', β') are saved in the automaton in an upscaled form depending on a constant factor. The bigger is this upscaling factor, the less is the loss of precision in the discretization of the parameters. From these upscaled parameters, Uppaal computes the δ_Z value by summing the activate ones afterwards, to get the resulting delay, the upscaling factor is divided (with an entire division) by the result.

For each gene, this calculation is done in a function *computeDelay()* called by the function *rA()* and by the switches $level\_j \longrightarrow update\_j$. Since a change in the regulators configurations can occur when the gene is in a location *update_j*, we need to add to these locations loop-switches which react to the notification of the regulators and call the function *computeDelay()* (Figure 6.2). The result of the function *computeDelay()* is stored in four local variables: *minUp*, *maxUp*, *minDown*, *maxDown*. The two former store the delay bounds before an increase, the two latter the delay bounds before a decrease. As for the first variant, an arbitrary large value i_∞ is used to represent the infinite values.

Since the function *computeDelay()* for a gene is fit to the number of regulations of the gene, it is declared locally in the gene template.

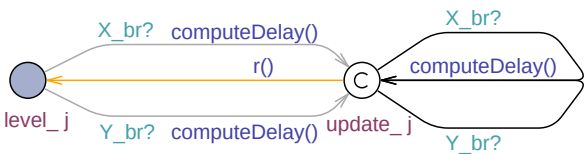


Figure 6.2: A location *level_j* and its associated *update* location with the function *computeDelay()* added to the switches $level\_j \longrightarrow update\_j$. On the left are depicted the added loop-switches in the *update_j* locations to recompute the delay when a regulator notifies a change of its concentration level.

Comparison

Both variants have advantages and disadvantages and each one seems to be more adapted to specific situations than the other. The precision of the pre-calculation variant is better since numbers can be treated as real numbers in the model-builder with a discretization only for the final results. However the generated arrays can be very large and hardly computable since their size growth exponentially with the numbers of regulators and concentration levels (add to a gene one regulator with 10 concentration levels multiplies the size of its delay arrays by 11). Moreover, a significant part of the pre-computed values can be never used since they are associated with concentration levels of gene products non-reachable by these ones.

When the configuration of regulators is too complex, the variant computing the delays on-the-fly seems to be more advisable. In this variant, the loss of precision can be limited by using a strong upscaling factor. However, the computation of the delays in the model checker increases the workload of this last. An interesting question is to determine to what extend the model checker is slowed down by this computation. Besides, an inefficient aspect of the on-the-fly variant is that a same delay has to be computed again and again each time the corresponding concentration levels of the regulators are met. Indeed, there is no efficient way to save the delays once computed.

Note that with a model checker for real timed automata, the on-the-fly calculation variant without the upscaling mechanism seems to be naturally the best way to compute the delays since the precision is no more a problem.

6.2.3 Clock variants

In this section, two different solutions to manage the clocks *cUp* and *cDown* are presented. The first one makes use of the stopwatches to get the proper behavior of the clocks. The second, to keep the reachability problem decidable, avoids the use of stopwatches by an approximation of the clock behavior.

Clock with stopwatches

The first variant, called the *variant with stopwatches*, gives the proper behavior for the clocks by using simply the timed automata stopwatches to freeze the clocks. The variant is described for the template of the gene Z in the context of the on-the-fly calculation variant. But the only difference with the pre-calculation variant is the use of the variables *minUp*, *maxUp*, *minDown*, *maxDown* in the invariants and guards

instead of references to the array *time_up* and *time_down*.

To control the activation of the stopwatches, we need two boolean variables *down* and *up* which are defined as follows in the *r()* and *rA(x)* functions:

$$up = (i_\infty > 1/\delta_Z^{max}) \wedge (1/\delta_Z^{max} \geq 0)$$
$$down = (-i_\infty < 1/\delta_Z^{min}) \wedge (1/\delta_Z^{max} \leq 0)$$

These variables *up* and *down* are computed in the function *r* and *rA()* and are true respectively when the gene Z is in a situation where the δ_Z value allowed it to increase its product concentration level and to decrease it in future (situation three). With these variables, the invariant in each *level_j* location is

```
((cUp - cDown) <= maxUp) &&
((cDown - cUp) <= maxDown) &&
(cDown' == down) &&
(cUp' == up)
```

where a clock is frozen in the two last lines when its derivative equal zero (or false).

The guards are in the orange switches:

```
((cUp - cDown)<= maxUp) && ((cDown - cUp) <= maxDown) ,
```

for the red switches: `((cUp - cDown)>= minUp)`, and for the blue switches: `(cDown - cUp) >= minDown`.

The clocks are reset in the function *rA()* which is called whenever a change of concentration level is required.

Finally, to end the implementation of this variant, we need to add some loop-switches in all *level_i* locations (Figure 6.3). These loop-switches change the evolution of the gene product when the delay is in the situation three (Z, with the current δ_Z value, is allowed in future to increase or decrease its concentration level). These switches randomly freeze one clock or the other by setting the *up* or *down* variable to false. To accomplish this, a third boolean variable *both* (defined in the *r()* and *rA(x)* functions by $both = up \wedge down$) is employed to store the fact that both variables are true.

Such a mechanism starts with both clocks unfrozen, and then oscillates randomly between the situations where only one of the clocks is frozen or where both are frozen. According to the timed model, the clock with the lower delay reaches it before the other

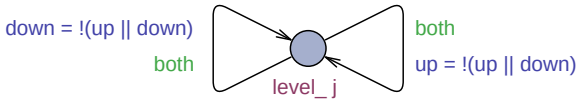


Figure 6.3: The random mechanism for each location *level_j* which drives the clock freezing when, according to the current δ_Z value, the concentration level of Z is allowed to increase or decrease by one level in future.

clock with a greater probability. Note that the start situation where both clocks are unfrozen is useless since a difference over their values is always done. Such situation can be avoided by using a slightly more complex mechanism.

Clock without stopwatch

The previous variant using the stopwatches provides a complete and direct implementation of the timed model clocks. However, the stopwatches use affects the decidability of the reachability problem. Moreover, when these lines are written, their use is only possible with a version of Uppaal 4.1 under development, version which is yet unstable and produces several bugs letting the variant with stopwatches unusable for the moment. These problems have led us to explore to what extent it is possible to get rid of the stopwatches. This result in a second variant, called the *variant without stopwatch*, which provides an imperfect solution.

Since we cannot freeze the clocks, and since there is no way to store the values of the clocks or to initialize a clock with another one in Uppaal, the only solution is to no more use the difference between the clocks and to reset a clock which is switched from a situation where the clock should be frozen to a situation where the clock is active (i.e., not frozen). This solution is imperfect but is the only one available. Figure 6.4 shows the evolution of the clock difference with this solution in comparison with the solution of the previous variant.

An issue related to this solution is where to do the reset in the template. Again here, there is no ideal solution. Indeed, the reset cannot be done when leaving a location *update_j* since the choice of the switch to fire requires the proper value for the clocks. The only solution is then to reset preventively the clock when we enter in a location *update_j* (Figure 6.5 with the function *resetClock()*) with an inversion of the updated δ sign. Moreover, since the inversion for a clock can occur when the active location is an *update_j* location, we need to add in these locations a loop-switch per regulator to make the preventive reset if necessary. The problem witch such a system is that when two changes of situation as described before occur instantaneously one after the other,

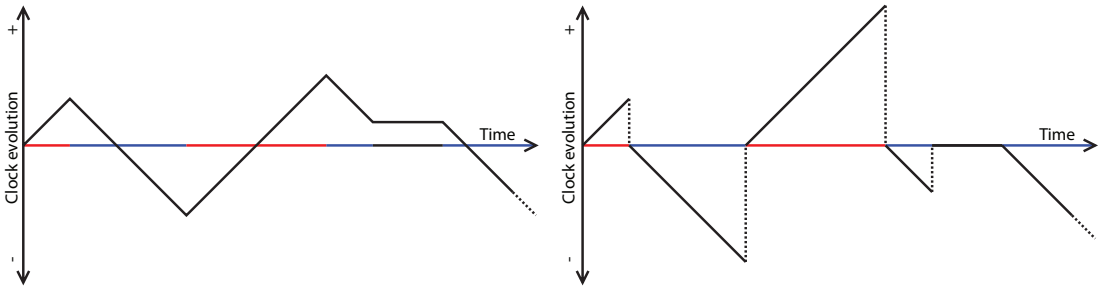


Figure 6.4: A graph which represents for a gene the normal evolution of the difference $c_Z^{up} - c_Z^{down}$ simulated with the stopwatch variant (on the left) and simulated in the variant without stopwatches (on the right). The abscissa is red when, in the timed model, only the c_Z^{up} clock is unfrozen, blue when only the clock c_Z^{down} is unfrozen, and black when both clocks are unfrozen. In the version without the stopwatch, we can see the imperfections resulting from the clock resets.

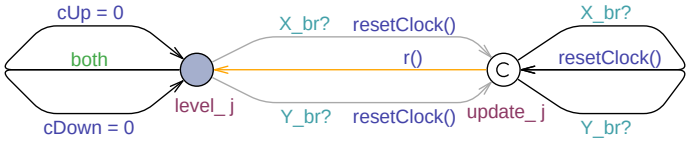


Figure 6.5: On the left, the two loop-switches added on each location level_j to randomly reset the clocks when Z can increase of decrease its concentration level at the same time. On the right, the loop-switches added for each regulator to the *update_j* location. On these switches and on the switches $level\_j \longrightarrow update\_j$, the *resetClock()* function preventively resets the unused clocks.

both the clocks are reset needlessly.

Finally, we need to add a mechanism in the location *level_j* which resets randomly one or the other clock when the gene is in situation three. As shown in Figure 6.5, this is done with two loop-switches guarded with a boolean variable *both* as used in the previous variant. If the purpose of the model is to get the trajectory of the concentration levels, this mechanism is not ideal since it can favor a situation where the gene product is considered as only stable ($\delta = 0$) if the model checker fires too promptly the switches.

For the guards and invariants, this variant is similar to the previous variant without the stopwatches and clock difference. Following the pre-calculation variant, the invariant for the *level_j* locations are:

```
(cUp <= time_up[Z][X][Y][1]) && (cDown <= time_down[Z][X][Y][1])
```

The guard for the orange switches are:

```
(cDown <= time_down[Z][X][Y][1]) && (cUp<= time_up[Z][X][Y][1]),
```

for the blue switches they are: `cDown >= time_down[Z][X][Y][0]`, and for the red switches: `cUp >= time_up[Z][X][Y][0]`.

Remember that when a gene product is not allowed to increase (decrease) its concentration level, the array *time_up* (respectively *time_down*) has the value i_∞, which outlives the duration of the simulation. Note that, in the current implementation, this aspect can appear as a problem for some state space analyses. Indeed, in assertions which require to explore exhaustively the state space, the model checker can reach states where the time of a clock is bigger than i_∞ and therefore where an improper changes of concentration level is allowed. However, such a case can be easily avoided or detected by using in the invariant two boolean variables *up* and *down* as previously defined, by bounding the exploration to a maximal time (see Section 7.5), or by analyzing the returned trace to check whether a trace is associated with a clock indicating a time equal to i_∞.

Finally, to conclude this variant, we can underline that the use of two clocks is not necessary since we can get the same result with only one clock, but then the invariants and guards would be more complicated. However, note that the use of two clocks here does not change the size of the state space for the automaton.

Comparison

As already described, the first variant with stopwatches provides the right behavior to simulate the clock system of the timed model. Unfortunately, it requires the use of stopwatches which affect the decidability of the reachability problem and which are only available (when these lines are written) on an unstable development version of Uppaal. If the latter point is only a question of time to be resolved, the first point is more worrying since it entails the undecidability of several assertion for the model checker .

The second variant which does not use the stopwatches and therefore does not suffer from such problems is however not able to implement correctly the behavior of the timed model clocks. But since, with the limitation of the timed automata formalism, there is no way to get a better result without stopwatches, and since the stopwatches version is currently unusable, we are forced to consider the second variant for the evaluation of the new approach.

6.3 Gates

This section discusses briefly for each kind of gate, its definition in the timed model and its implementation in the new approach.

Boolean gates

A boolean gates G regulating a gene Y and associated with a β_G parameter (or more precisely a β_G^{min} parameter and a β_G^{max} parameter) can be represented in the Formula 6.1 of the timed model directly with a new member $+\beta_G \times \Theta_G$ where Θ_G returns 1 if the condition over the concentration levels of the input gene products (the regulators) in the boolean gate is respected, 0 otherwise.

For instance, if we have a boolean gate regulating Y with the condition $(A < t_A \wedge B \geq t_B)$, where t_A and t_B are the thresholds for the corresponding regulators, then the formula computing the δ_Y values is extended with the new member $+\beta_{G_Y} \times \Theta(A < t_A \wedge B \geq t_B)$, where β_{G_Y} is defined as the effect of the gate on Y.

The implementation of boolean gates of any complexity can be done without difficulties in both computation variants. In the on-the-fly calculation variant, the effect of a boolean gate is computed in the *computeDelay()* function. In the pre-calculation variant, the effect is integrated directly in the delay arrays computed by the model-builder.

Discrete gates

Since the new approach compute the δ values with a summation, the default gate is a SUM gate. It is possible to extend the timed model for MULT gates by allowing the definition of clusters of regulations where the resulting effect is not the summation of the effect (the β' parameters) of all regulations in the cluster, but their multiplication. Nevertheless, it is then necessary to reintegrating the basal production rate of the inhibitions into their β parameters. Indeed, for a gene, the timed model, which is defined to formalize the IKNAT principle, reassemble in the parameters $\mathcal{B}$ the basal production rates of the gene product. In consequence of this, the semantics of the β parameters associated to the inhibition is changed and their value is negative. A MULT gate of a cooperation comporting inhibition would thus result in an improper result if it was done according to the current timed model.

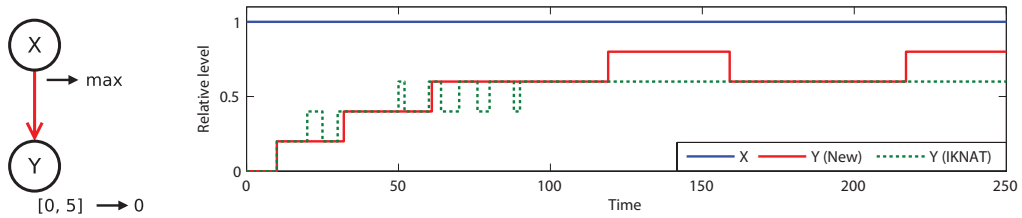
From these semantic issues with the current form of the timed model for the MULT gate, we have not proceeded to the implementation of this one.

6.4 Observations

Since the new approach implements directly the subjacent principle of the IKNAT approach, several observations made for IKNAT can be kept unchanged here. Therefore, the aspects which differ are only discussed.

Stability

The main motivation to develop this approach was to get an implementation of the timed model without the IKNAT stability problems. By following in a direct way the delays computed in the timed model, this goal is achieved and the new approach does not produce any unexpected behavior like instability or unwanted stabilization.



(a) GRN **(b)** IKNAT simulation

Figure 6.6: The example of Section 5.4.3 simulated with the new approach (on-the-fly variant without stopwatch). The red line is the result of the new simulation, the dashed green is the simulation performed with the IKNAT approach. The behavior under the new approach is more stable and produces the expected oscillation for Y.

Note that such oscillation between two levels are frequent since the computed δ values are rarely equal to zero, unless one uses expressly appropriate α and β parameters. Such oscillations can be a drawback for a boolean oriented modelling, but there is no proper way to get rid of them when we used the timed model. The IKNAT stabilization is not adequate since it concerns only simple regulations and since the level where the stabilization occurs depends on the gene antecedents. These oscillations have to be interpreted like an intermediate concentration level.

Time aspects

We can remark that the variant of the automata without stopwatch corresponds in fact in a model where the history aspect of the gene products inside a concentration

level is not handled. This non-history management is the same as in the timed-boolean approach (see Section 4.2, abstraction b).

Concentration levels

The only difference here with IKNAT is the possibility to get for a gene product a basal "concentration level" oscillating between two neighboring concentration levels. To get a stable basal concentration level for the product of a gene Y with a non-zero basal production rate, we just need to get an equilibrium between the $\mathcal{B}$ value and an α' value of the gene: $\exists k : (1 \leq k \leq l_Y) \wedge (\sum_{i=0}^{k} \alpha_i - \mathcal{B} = 0)$.

Gates

The possibility allowed by the boolean gates in this model is the same as in the IKNAT approach. However, with the better stability of the concentration levels, the use of these gates is safer and it is no more required to avoid a dense set of thresholds among the range of concentration levels.

Another difference in this approach is the technical possibility to define easily MULT gates. However, the difficulties of these constructions is to get a proper semantics and they require to refine the timed model.

Timed automata

Although simple in their conception, the automata created in this approach are the most complex comparatively to those used in the timed-boolean approach and in the IKNAT approach. Indeed, according to the variant followed, the automata use high level constructions like the stopwatches or the functions. Such constructions are not without influences on the performance and the ability of the model checking. While the stopwatches impact the decidability of the reachability problems, the functions used to compute delays in the on-the-fly calculation variant are a workload for the model checker and could slow down its exploration of the state space. These aspects incontrovertible for a direct implementation of the timed model can be nevertheless alleviated by tolerating compromises on the accuracy of the clock simulation and by delegating the delay calculations to the model-builder.

6.5 Conclusion

In new approach that we propose, we avoid the main problem of IKNAT concerning instability by implementing our timed model in a more direct way, and we obtain a more flexible model for the specification of gates. However, limitations related to the implementation of the clocks and to the calculation of the delays in the discrete timed automata formalism were raised. For each of these limitations, two solutions have been proposed (summarized in Appendix C, Table C.1), with, for each of these solutions, some advantages and disadvantages which have to be considered when using one solution over another.

7

Case studies

This chapter compares the different modelling approaches - depicted previously and summarized in Table C.2 of Appendix C - in some case studies and from the point of view of expressiveness, easiness, and performance. The selected case studies are the self-regulation motifs, the incoherent Feed Forward Loop motif, and an oscillatory pattern. The first two come from the Uri Alon book [2], and, although described briefly here, have a more complete description in Appendix A. The last one comes from [16] and is described in this chapter in due time. These case studies are small GRNs. However it makes sense to use them since in biological networks, simple structures are often used as modules linked together to build larger structures. The ability of the approaches to simulate these modules is thus a requirement before considering the simulation of bigger structures.

In addition to these case studies, some artificial GRNs with a complex dynamics are tested to compare the model checking performance in the different modelling approaches. It is necessary to use such dedicated GRNs since the dynamics of the previous case studies is too simple, entailing a processing time too short and no significant for performance results.

But first, we present briefly the *model-builder*, a tool developed to support the work of this chapter.

7.1 Model-builder

The model-builder is a command line tool which, from a description of a GRN, generates the timed automaton according to a given approach and parses the result returned by

the Uppaal model checker on this automaton. The model-builder is described in detail for the use and implementation aspects in Appendix B, however some of its aspects are introduced here.

Model-builder inputs

The inputs of the model-builder are of three types. First, there is an XML file depicting the GRN for which the model-builder has to generate the Uppaal timed automaton. Secondly, there is some arguments which specify the approach to follow, the variant of the approach to apply, and some other aspects related to the parsing operated by the model-builder on the trace returned by the Uppaal model checker. Finally, there is an assertion on the timed automaton for this model checker.

For the XML input, and for each of the approach depicted in the previous chapter, a DTD file has been created to specify the structure of the XML description. Such a describing XML file is basically constituted of various elements as *gene*, *regulation*, *gate*, ... specifying each a particular aspect (a gene, a regulation, ...) of the GRN to process. Evidently, following the target approach of the depicted GRN, there is some variations in the elements included in the XML file (see Appendix B.2).

Note that to allow a direct comparison of the new approach and the IKNAT approach, the β and α parameters are specified in the XML file with a form compatible with both approaches, as follows: $1/\alpha'_t = 1/(\sum_{j=0}^{t} \alpha_j - \mathcal{B})$ and $1/\beta'_{kt} = 1/(\sum_{i=0}^{t} \beta_{ki})$, with a $\mathcal{B}$ value compatible with the implementation made in IKNAT (i.e., entailing a stable basal activity level). In consequence, the presentation of these parameters in this chapter are under the form $1/\alpha'_t$ and $1/\beta'_{kt}$ for both approaches.

The inputs parameters of the model-builder are necessary to specify the approach to follow, its variant, and some other aspects. For the case studies of this chapter, and unless otherwise specified, we use the stability extension on the extreme levels with the IKNAT approach, the urgent mode with the timed-boolean approach, and unless otherwise specified, for the new approach, the on-the-fly calculation variant (upscaling of 1000) without stopwatch (which produce crashes in the Uppaal development version). The other relevant parameters of the model-builder are let to their default value (see Appendix B.4).

Finally, for the model checker assertion, each element of the timed automata presented before can be employed within the assertion. However, for the easiness, the level of each gene product is stored in an integer variable which the same name of the gene. These variables can be referred in the assertion. In addition, to get an absolute time

reference, a never reset clock named *globalTime* can be defined in each automaton by the model-builder. A such clock can be employed, for example, with assertions of the form `[E<> globalTime > X]` to get the trajectories of the GRN until the time value X. Several other examples of assertions are presented in Appendix B.3.

Output of the model-builder

The outputs of the model-builder are of two types. Firstly, there is evidently the Uppaal timed automaton corresponding to the GRN of the input XML file and corresponding to the approach and its variant specified by arguments. Secondly, once this timed automaton and the input assertion are processed by the model checker of Uppaal, and if this last returns a trace, the model-builder can parse this trace to return the relevant information about the trajectories of the gene products. This information about the trajectories is returned in a file with a convenient format to chart easily the trajectories (see Section B.5).

Note that since Uppaal returns the shortest trace and then used the lower bounds where time intervals are specified, we use only deterministic time values for the case studies with charted trajectories.

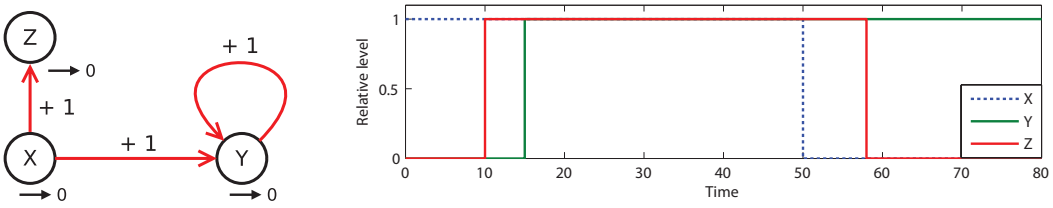
7.2 Self-regulation motif

Figure 7.1: The self-regulation

The self-regulation is a very simple motif where a gene regulate himself in a positive or a negative manner. But despite this simplicity, the effects of such a motif are important. They can be described according to two aspects relevant for our work. The first one is the influence of the self-regulation on the gene response time. The self-inhibition decreases the response time and the self-activation increases this one. The second aspect is related to the concentration level and concerns the self-activation which allows a gene to stay expressed by its product.

7.2.1 The timed-boolean approach

Following the standpoint on the time aspect, this approach is paradoxically more or less capable of properly representing the effect of a self-regulation. Less able because the regulations cannot modify the time needed for a gene to change its product activity



(a) GRN **(b)** Timed-boolean simulation

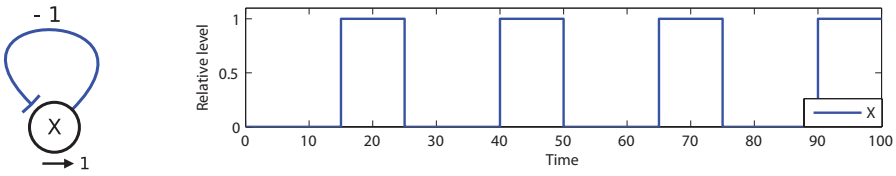
Figure 7.2: An example of self-activation (a) modeled with the timed-boolean approach (b). X, which has been just activated (start activity level of one), promotes Y and Z. When X stops its activity, only Y can sustain itself. The relevant parameters are $\delta_{Y,0+} = 15$, $\delta_{Z,0+} = 10$, $\delta_{Z,1-} = 15$, $K_{Y,\{\}} = 0$, $K_{Z,\{\}} = 0$, $K_{Y,\{X\}} = 1$, $K_{Y,\{Y\}} = 1$, $K_{Y,\{X,Y\}} = 1$, and $K_{Z,\{X\}} = 1$. The delay required by Y to increase its activity level is set to a longer value than Z to simulate the effect of the self-activation.

level, inducing then only as direct result for the study case the ability for the gene to sustain itself the activity level with the self-activation. Therefore, the influence of a self-regulation on the time aspect must be directly specified by the designer on the delays, letting the model not really involved in this important aspect for the self-regulation.

However, more capable, because these regulations are the ones whose time aspect is the best modeled in this approach. Indeed, since a change in the activity level of a gene product is always followed by a change of the self-regulation effects on the gene itself, the effects specified for the times can never be inadequate for the self-regulation. In other words, the self-regulation does not suffer from the constant reactivity imposed by the constant delays.

An example of self-activation is shown in Figure 7.2, where two genes are regulated by another. One of the two gene has a self-activation, while the other not.

Finally, as for the other approaches, note that the self-inhibition can be used to generate regular oscillations with an amplitude of one level (Figure 7.3). Such oscillations, which



(a) GRN **(b)** Timed-boolean simulation

Figure 7.3: A self-inhibition producing oscillations with the timed-boolean approach. The relevant parameters are $\delta_{X,0+} = 15$, $\delta_{X,1-} = 10$, $K_{X,\{\}} = 0$, and $K_{X,\{X\}} = 1$.

are only possible with discrete model for such a configuration, must be interpreted as a stable intermediate level.

7.2.2 The IKNAT and new approaches

On both approaches which follow the timed model, the situation is different. Indeed, a self-regulated gene cannot influence its time required to reach a concentration level under two, since its self-regulation is effective only above the concentration level 1. In consequence, we experiment here a self-inhibition with 6 concentration levels, and, as in the previous example, in comparison with a gene not self-regulated (Figure 7.4).

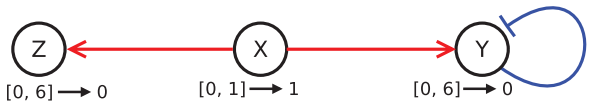


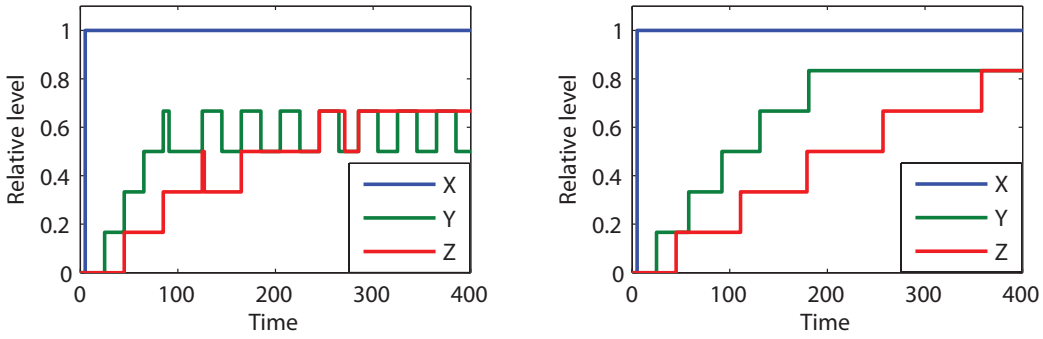
Figure 7.4: An example of self-inhibition with Y, whereas Z has only a normal regulation.

The delays for the simulation are given in Table 7.1. According to the timed model, these delays are set to stabilize the gene Z and Y on the concentration level 5 (relative concentration level 0.83) without any oscillation. Since Z is self-inhibited (with the same effect than the dilution/degradation parameters), we can set a stronger activation (*2) from X for a same steady state.

As we can see on the result of the simulation (Figure 7.5), the new approach gives the expected result. For the IKNAT approach, the situation is different. If, as expected, the Y concentration level increases quicker than the Z concentration level, both gene products do not succeed to reach the expected concentration level 5 and are stuck on the levels 3 and 4. Such results for Y are related to the same cause than the unexpected stabilization (as Z), but from the more complex configuration, the stabilization does not occur.

	Z		Y			X
CL	$\mathbf{1/\alpha'_j}$	$\mathbf{1/\beta'_{Xi}}$	$\mathbf{1/\alpha'_j}$	$\mathbf{1/\beta'_{Yi}}$	$\mathbf{1/\beta'_{Xi}}$	$\mathbf{1/\alpha'_j}$
0	∞	∞	∞	∞	∞	-5
1	100	40	100	-100	20	∞
2	96	/	96	-96	/	/
3	82	/	82	-82	/	/
4	66	/	66	-66	/	/
5	40	/	40	-40	/	/
6	10	/	10	-20	/	/

Table 7.1: Delays specified for the GRN in Figure 7.4. As for the following time tables of this chapter, the delays are without interval and CL stands for Concentration Level.



(a) IKNAT approach **(b)** New approach

Figure 7.5: Simulation of the GRN in Figure 7.4 with the delays of Table 7.1. All the genes start with the activity level 0.

Note that if the wanted steady state is the maximum concentration level, the self-activation can be used to speed up the concentration growth. But such a use of the self-activation is not realistic and relies precisely on the fact that there is a limited number of concentration level to stop the growth in these approaches.

7.2.3 Remarks

All the modelling approaches are able to simulate the self-regulation motifs on their own way. For the IKNAT approach, the stability problems influence the expected result, but with enough concentration levels, it does not prevent the use of the IKNAT approach to model these motifs.

7.3 Incoherent feed forward loop motif

The second case study of this chapter is the Incoherent Feed Forward Loop motif (IFFL). The effect of such a motif is the generation of a brief pulse in the concentration of a gene product (the product Z for Figure 7.6). The predominant aspects of this motif is the amplitude of the pulse in level and the duration of this pulse.

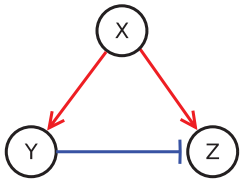
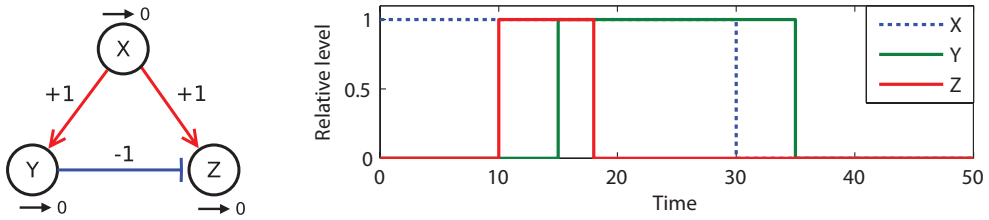


Figure 7.6: IFFL



(a) GRN **(b)** Timed-boolean simulation

Figure 7.7: An example of IFFL with in (a) the GRN and in (b) the simulation with the timed-boolean model. In this simulation, X has just been activated. The relevant parameters are $\delta_{Y,0+} = 15$, $\delta_{Y,1-} = 5$, $\delta_{Z,0+} = 10$, $\delta_{Z,1-} = 3$, $K_{Y,\{X\}} = 1$, $K_{Y,\{\}} = 0$, $K_{Z,\{Y\}} = 0$, $K_{Z,\{X\}} = 0$, and $K_{Z,\{X,Y\}} = 1$.

7.3.1 The timed-boolean approach

The simulation of a IFFL in the timed-boolean model is very easy and direct. An example of a GRN for a IFFL is represented in Figure 7.7. In this approach, the amplitude of the pulse in Z is limited to a single activity level. The duration of the pulse is the summation of the timed needed for the product Y to reach the activity level 1 and the timed needed for the product Z to fall to the activity level 0 from the level 1.

7.3.2 The IKNAT and new approaches

The modelling of a IFFL can be done in several manners in the IKNAT and new approaches. As for the next case study, we experiment two methods here: a minimal modelling with a minimal number of concentration levels for the gene product (as a sort of boolean modelling) and a modelling with more concentration levels.

Minimal modelling

The minimal modelling considered here is similar to the one of the timed-boolean approach, except that here the regulations influence the temporal aspect. The GRN used is represented in Figure 7.8. This GRN is equivalent to the GRN of Figure 7.7a but with an explicit AND gate. The delays used for the GRN are presented in Table 7.2. Figure 7.9 shows the result of the simulation on both approaches.

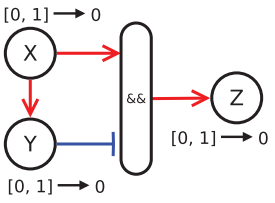
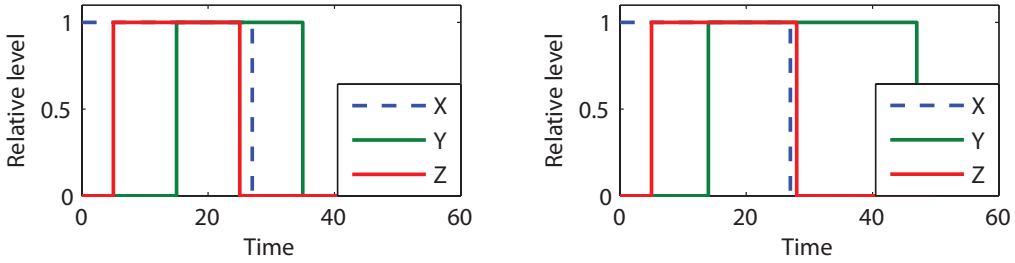


Figure 7.8: Minimal IFFL

	Z		Y		X
CL	$1/\alpha'_j$	$1/\beta^{AND}_{X\geq 1 \& Y<1}$	$1/\alpha'_j$	$1/\beta'_{Xi}$	$1/\alpha'_j$
0	∞	5	∞	∞	∞
1	15		20	15	27

Table 7.2: Delays specified for the GRN in Figure 7.8.



(a) IKNAT approach **(b)** New approach

Figure 7.9: Simulation of the GRN Figure 7.8 with the delays of Table 7.2. X starts with a concentration level 1.

As can be seen, the situation can be simulated by both approaches in a boolean sense with a minimal number of concentration levels. Note that between both approaches, several variations occur on the time required by the products to decrease their concentration level. These variations are imputable to the different implementations of the timed model.

Non-minimal modelling

For the non-minimal IFFL, the gene Y and Z have 6 concentration levels (Figure 7.10) and the boolean gate is implicitly defined with the delays (Table 7.3). As we can see in Figure 7.11, both approaches give a result rather similar if we compare them with the minimal minimal. This can be explained by the fact that the little variations in the timing of each change of concentration level in one sense are counteracted by variations in the other sense, resulting globally in an evolution similar between both approaches.

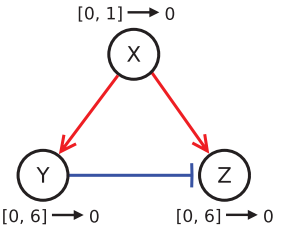
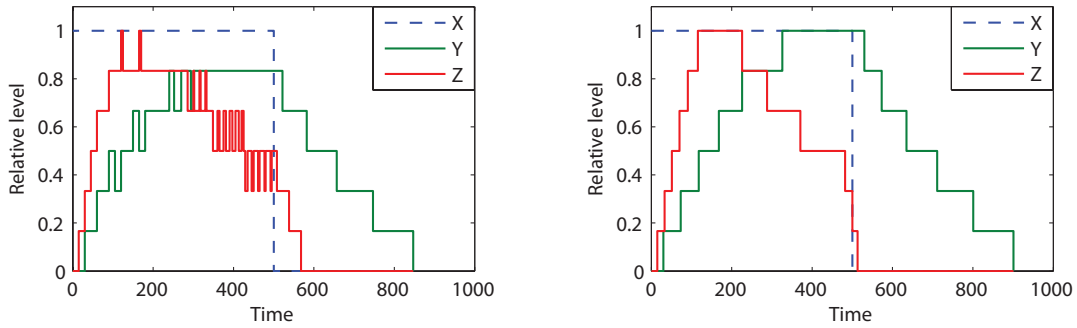


Figure 7.10: Non-minimal IFFL

Finally, note that it is also possible to use an AND gate for this IFFL experiment. Since the effect of Y on Z changes radically passed the activity level 3, the result of a gate with such a threshold would be quite similar to the current result. The strong

	Z			Y		X
CL	$1/\alpha'_j$	$1/\beta'_{Yi}$	$1/\beta'_{Xi}$	$1/\alpha'_j$	$1/\beta'_{Xi}$	$1/\alpha'_j$
0	∞	∞	∞	∞	∞	∞
1	150	-150	15	100	30	500
2	130	-130	/	96	/	/
3	105	-105	/	75	/	/
4	80	-30	/	60	/	/
5	50	-16	/	42	/	/
6	20	-15	/	30	/	/

Table 7.3: Delays specified for the GRN of Figure 7.10.



(a) IKNAT approach **(b)** New approach

Figure 7.11: Simulation of the GRN of Figure 7.10 with the delays of Table 7.3. X starts with a concentration level 1.

variation in the effect of a regulation can be found in ODE models with regulations driven by Hill functions with high exponents n.

7.3.3 Remarks

As for the self-regulation motifs, the IFFL motif does not create any difficulty with the three modelling approaches. It is nevertheless good to keep in mind that with the timed-boolean approach, the delay specified for the IFFL motifs are not necessary viable if the genes can be regulated by some other genes outside the motifs.

For the IKNAT and new approaches, we can emphasize the fact that the bigger is the number of concentration level, the more similar are both approaches. Note that the minimal modelling for the IKNAT approach is possible here because the extension which stabilizes the extreme concentration levels is used.

7.4 Oscillatory pattern

The oscillatory pattern is a couple of genes inter-regulated and generating oscillations in their product concentration. This pattern, described in [16], is involved in several biological systems like the cell cycle, the circadian rhythms, or the response of several signaling pathways. Figure 7.12 shows the GRN entailing the oscillations.

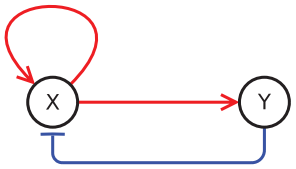


Figure 7.12: The GRN of the oscillatory pattern.

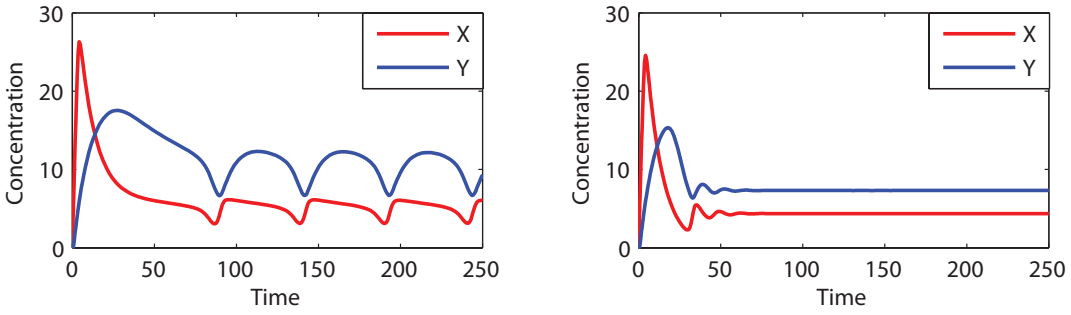
The originality and difficulty of such a pattern is that the elimination of the self-activation of X ensues the suppression of the oscillations. The behavior of such an oscillatory GRN can be described by the followings ODE formulas:

$$\frac{dX}{dt} = \frac{0.6 \cdot X^{10}}{4.6^{10} + X^{10}} + \frac{8}{1 + \frac{Y^{10}}{5.5^{10}}} - 0.1 \cdot X \tag{7.1}$$

$$\frac{dY}{dt} = \frac{2 * X^4}{5^4 + X^4} - 0.1 \cdot Y \tag{7.2}$$

Equation 7.1 drives the X concentration with first, the Hill function of the self-activation, then a spontaneous production rate moderated by the Hill function of the Y inhibition, and finally the X degradation rate. Equation 7.2 drives the Y concentration with the Hill function of the X activation and the Y degradation rate.

In Figure 7.13 the result of the ODE formulas is charted with and without the Hill function responsible of the self-activation of X. It is interesting to note that the amplitude and the frequency of the oscillations are very sensitive to the value of the parameters



(a) With the self-activation of X

(b) Without the self-activation of X

Figure 7.13: Simulation of the GRN in Figure 7.12 with the ODE formulas 7.1 and 7.2. In the second case, the Hill function for the X activation is removed. For both cases, the initial concentration of the two gene products is zero.

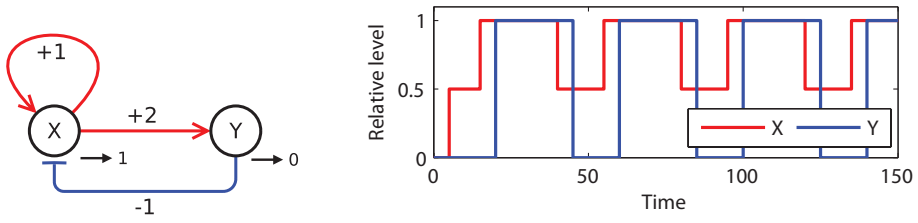
in the equation 7.1 and 7.2. Indeed, some subtle modifications of them can result in a modification of the oscillations or, more often, in their suppression.

7.4.1 The timed-boolean approach

The modelling of such a pattern with the timed-boolean approach is possible (Figure 7.14) but however with a concession on the Y activity levels. Indeed, since Y is the regulator of only one gene and since only one threshold is allowed by regulation, the maximum activity level of Y is limited to one, making Y oscillates obligatory between zero and one. This situation underlines the strong abstraction of the activity levels in the timed-boolean approach which does not model the distinct concentration of Y in the oscillation dimples.

Note that a distinct activity level for the Y oscillation dimples is possible by extending the timed-boolean model with the capability to specify several thresholds for each regulation. From this extension refining the regulations, it would be then possible to spread the regulation of Y in two activity levels, augmenting the Y maximal activity level in the same time (with, therefore, an abstraction a little bit less boolean).

Finally, notice that the self-activation is not required to get these oscillations (set the basal activity of X to 2 is sufficient), but its presence is required to get the same behavior than the pattern (i.e. when we remove it, the oscillations disappear, which is charted by the activation of X with then, nothing else).



(a) GRN **(b)** Timed-boolean simulation

Figure 7.14: A timed-boolean GRN (a) of the oscillator pattern and its simulation (b). The relevant parameters are $\delta_{X,0+} = 5$, $\delta_{X,1+} = 10$, $\delta_{X,2-} = 20$, $\delta_{Y,0+} = 5$, $\delta_{Y,1-} = 5$, $K_{X,\{Y\}} = 1$, $K_{X,\{X,Y\}} = 2$, $K_{X,\{X\}} = 1$, $K_{X,\{\}} = 1$, $K_{Y,\{\}} = 0$ and $K_{Y,\{X\}} = 1$.

7.4.2 The IKNAT and new approaches

As for the previous case study, we will try to see to which extend it is possible to model such a pattern in a minimal sense and in a non-minimal sense with the IKNAT approach and the new approach.

Minimal modelling

The minimal modelling considered here uses a minimal number of concentration levels for the gene products (no more than required), but however no boolean gate for the cooperation of the regulations in the gene X. This choice, resulting on the GRN of Figure 7.15, stemmed from the fact that a boolean gate is neither necessary nor evident to find for X.

According to the timed model, we have isolated the time values of Table 7.4 to produce the expected behavior, i.e., the oscillations with the self-activation of X and a stabilization without. Note that the times specified for the concentration level one of the two gene products, irrespective of the self-activation of X, are tuned to not produce oscillation between two levels (the products are perfectly stabilized on the concentration level one).

The result of the simulations on both approaches are charted in Figure 7.16.

For the new approach, the result is good since we have the expected evolution and more, contrary to the timed-boolean model, we have an intermediate concentration concentration level for Y and the begin of the simulation without the self-activation of X undergoes some brief oscillations before the stabilization (like in the ODE model).

For the IKNAT approach, the result is quite bad. With its natural instability inherent to the implementation of the timed model, the gene products suffer from several unexpected oscillations from one level to another. Since the number of concentration levels for both products is only of two, these oscillations really spoils the expected results.

Non-minimal modelling

We have tried here a modelling with 10 concentration levels for each product (Figure 7.17). For so many concentration levels, the tuning (by manual trials and errors) of the time values has revealed itself an hard and time consuming task. Indeed, as in the ODE model, a little change in one time value can removed the oscillations. It is thus very

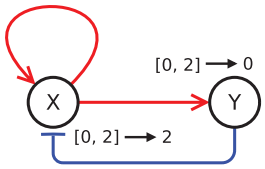
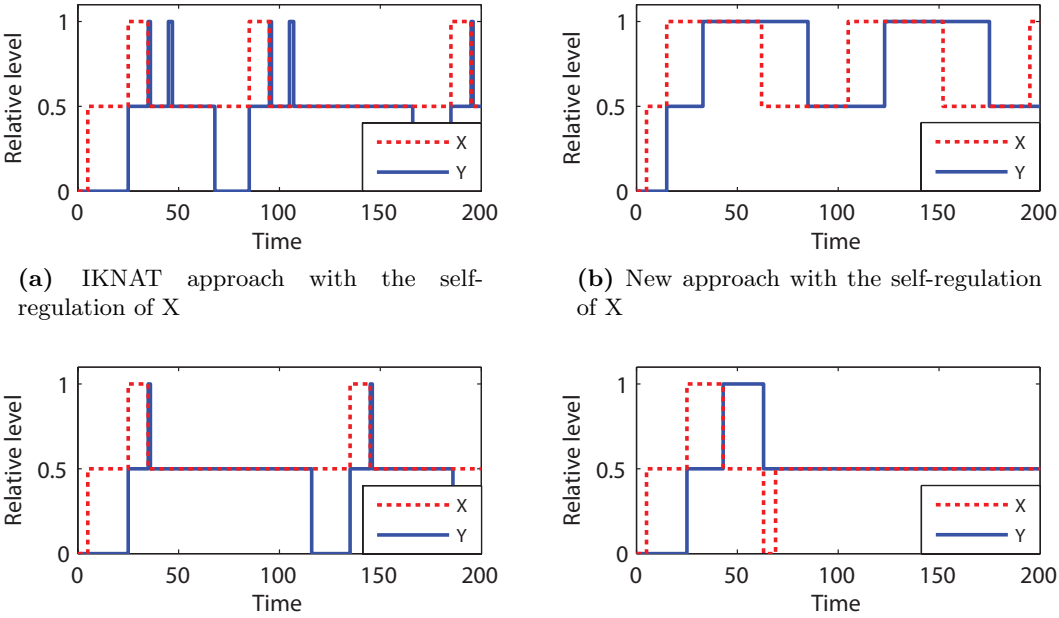


Figure 7.15: Minimal GRN for the oscillatory pattern.

	X			Y	
CL	$\mathbf{1/\alpha'_j}$	$\mathbf{1/\beta'_{Yi}}$	$\mathbf{1/\beta'_{Xi}}$	$\mathbf{1/\alpha'_j}$	$\mathbf{1/\beta'_{Xi}}$
0	-5	∞	∞	∞	∞
1	-20	-20	20	21	21
2	∞	-10	15	11	10

Table 7.4: Delays specified for the GRN of Figure 7.15.



(a) IKNAT approach with the self-regulation of X

(b) New approach with the self-regulation of X

(c) IKNAT approach without the self-regulation of X

(d) New approach without the self-regulation of X

Figure 7.16: Simulation of the GRN of Figure 7.15 with the delays of Table 7.4. On (a) and (c) the results for the IKNAT approach with the self-activation of X and without. On (b) and (d) the results for the new approach with the self-activation of X and without.

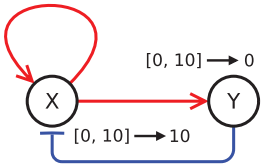


Figure 7.17: Non-minimal GRN for the oscillatory pattern.

difficult to converge to a result which induces oscillations with the self-regulation of X and a stabilization without. But despite this difficulty, once a good set of values is delineated (Table 7.5), we can show that such a pattern can be simulated with at least the new approach (Figure 7.18 (b) and (d)). At least only for the new approach because the results with IKNAT are very bad (Figure 7.18 (a) and (c)). It is probable that since the oscillations are very sensitive, the IKNAT instability disrupts their occurrences and prevents the simulation of such patterns which the IKNAT approach.

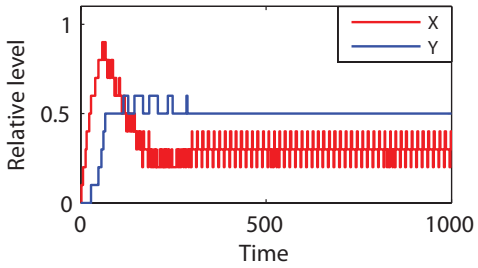
Note that the maximal number of concentration levels is chosen here to get a graphical result rather similar to the ODE model, with a stabilization of X and Y on the concentration levels two and three to follow anew the ODE model graphics. However, normally, such choices must not be driven by graphical consideration but by the requirements over the precision, the role of each gene and the relevant maximal concentration of each gene product in the modeled situation. For example, here, a lower concentration level for Y than for X could be in fact related to a higher concentration of Y than X, depending of the scales of the concentration levels defined for each gene product.

Non-minimal modelling inspired by the ODE models

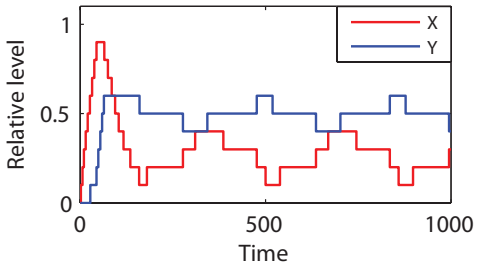
Face to the extreme difficulties to delineate by hand a proper set of time values for the previous non-minimal modelling, we have experimented the same modelling with a structuring more inspired by the ODE formulae for the oscillatory pattern. For this, with have first change the disposition of the basal production rate in the timed model. Indeed, in the ODE model 7.1 of X, the basal production rate (8) is directly moderated by the Y concentration, as it is the case for the inhibition in the ODE model. In the current timed model, which is a formalization of the IKNAT mechanism, the basal production rate $\mathcal{B}$ is integrated to the α' parameters and the X inhibition in its negative β' parameters, in consequence the link between them have to be think by the model designer. The idea here is to followed the structure of the ODE 7.1 by displacing the X basal production rate from the α' parameters to the β' parameters of the X inhibition. In such a case, the β' parameters of the X inhibition are positive and decrease with the augmentation of the Y concentration level. Secondly, if the concentration level of each gene is considered as following the same scale for the concentration and if we want to stay close to the ODE model, we have to specify the same dilution/degradation delays for both gene products since they have the same rates in the formulae 7.1 and 7.2.

	X			Y	
CL	$\mathbf{1/\alpha'_j}$	$\mathbf{1/\beta'_{Yi}}$	$\mathbf{1/\beta'_{Xi}}$	$\mathbf{1/\alpha'_j}$	$\mathbf{1/\beta'_{Xi}}$
0	-2	∞	∞	∞	∞
1	-4	∞	∞	500	500
2	-7	∞	15	350	250
3	-8	-7	14	250	50
4	-10	-6	13	100	40
5	-14	-5	12	80	30
6	-19	-4	11	40	25
7	-25	-3	10	25	22
8	-35	-2	9	10	20
9	-48	-2	8	5	18
10	∞	-2	7	1	17

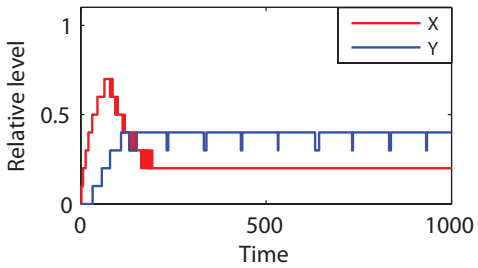
Table 7.5: Delays specified for the GRN of Figure 7.17.



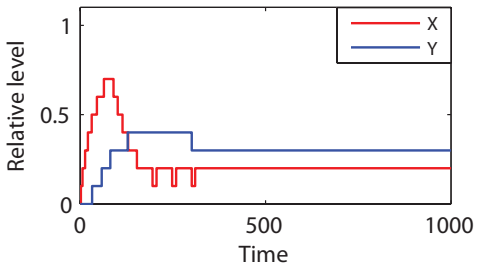
(a) IKNAT approach with the self-regulation of X



(b) New approach with the self-regulation of X



(c) IKNAT approach without the self-regulation of X

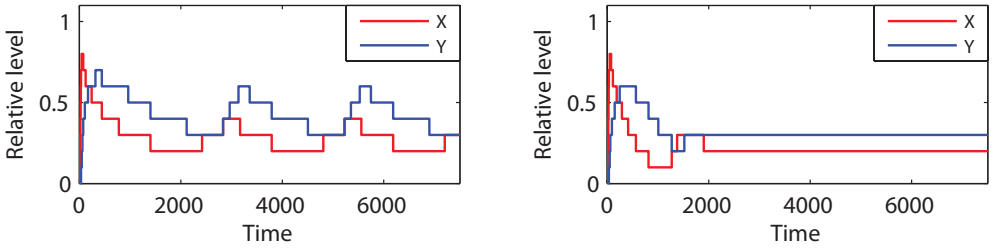


(d) New approach without the self-regulation of X

Figure 7.18: Simulation of the GRN Figure 7.17 with the delays of Table 7.5. On (a) and (c) the results for the IKNAT approach with the self-activation of X and without. On (b) and (d) the results for the new approach with the self-activation of X and without.

Such a modelling inspired by the ODE is experimented hereafter for the new approach only since the IKNAT approach requires increasing delays in the templates (templates which can be however easily adapted to removed this limitation). Note that since we use a wider range of time values for the parameters, the upscaling done here to maintain the precision is 10.000 instead of 1000.

The results charted in Figures 7.19 are better than the ones of the previous simulation for a shorter time spent in the configuration of the delays (Table 7.6). This encourages a deeper exploration of the timed model and of its link with the ODE models.



(a) With the self-regulation of X **(b)** Without the self-regulation of X

Figure 7.19: Simulation of the GRN of Figure 7.17 with the new approach and with the delays specified in Table 7.6. On (a) the result with the self-activation of X, on (b) the result without.

	X		X & Y	Y
CL	$\mathbf{1/\beta'_{Yi}}$	$\mathbf{1/\beta'_{Xi}}$	$\mathbf{1/\alpha'_j}$	$\mathbf{1/\beta'_{Xi}}$
0	5	∞	∞	∞
1	10	300	-500	350
2	75	200	-250	150
3	250	200	-150	125
4	∞	200	-125	63
5	∞	200	-100	50
6	∞	200	-75	38
7	∞	200	-38	25
8	∞	200	-25	14
9	∞	200	-19	11
10	∞	200	-10	9

Table 7.6: Delays used for the simulation of the figures 7.19. The dilution/degradation rate (α') are the same for both genes as in the ODEs 7.1 and 7.2. Note that the basal production rate of X is displaced from the parameters α'_j to the β'_{Yi} parameters.

7.4.3 Remarks

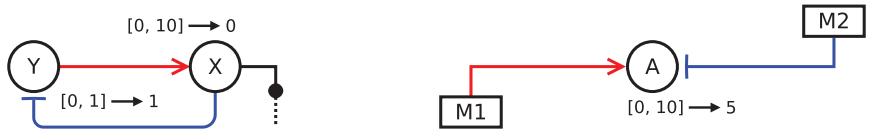
The oscillatory pattern underlines the limitation of the timed-boolean approach with its limited number of activity levels inducing a high abstraction.

But more important, the sensibility of the oscillatory pattern underlines the inability of the IKNAT approach to model sensitive GRNs. Note that the difficulties and the time needed for the configuration of the time values can be balanced by the fact that the ODE model of the oscillatory pattern was certainly not easy to define. Such difficulties have the merit to underline the inefficiency of the manually trials/errors delay tuning and to show that it is possible to refine the timed model by following the ODE structure for the role of the parameters. The idea is, besides the definition of guidelines and other automatic delay generation processes, to replace the basal production rate in the β parameters of the gene inhibitions. This parameters refinement - which improves the tractability of the timed model but can produce complication with cooperation between regulations - is technically usable on the new approach without modification and can be easily introduced in IKNAT by some slight adaptations of the templates.

7.5 Performance

The previous cases studies have underlined the limitation of the IKNAT approach in comparison to the new approach which allows the modelling of very sensitive situations with more precision. This precision itself makes the new approach more tractable to manipulate. But beside theses advantages, we could argue that the performances of IKNAT are better than the performances of the new approach. It is true that the IKNAT approach uses only automata without operations on the transitions whereas the new approach leans heavily on the calculations allowed by Uppaal. Moreover, it is interesting to know to which extend the pre-calculation of the time values alleviates the work of the model checker in the new approach. To answer to these questions, several performance benchmarks are reported in this section.

Note that we do not test here the performance of the timed-boolean approach since the state spaces of its automata, with the boolean abstraction, are not commensurable with the state spaces of the automata provided by the IKNAT and new approaches. Moreover, since the current implementation of the timed-boolean approach uses only one process for a GRN, building of the automata required an exponential complexity and thus the performance is more a matter of the model-builder than a matter of the model checker.



(a) Oscillatory module **(b)** A GRN with two modules

Figure 7.20: In (a), the structure of the oscillatory module with ten concentration levels. The bulled arrow is the outgoing regulation of the module. In (b), an example of GRN with two modules.

7.5.1 Process

GRN benchmarks

In order to conduct the benchmarks, we have created what we can call an *artificial oscillatory module*. *Oscillatory* because the goal is to get GRNs which are always in movement to avoid tractable situations where the dynamics of the GRNs is easy to browse. *Artificial* because we did it very insensitive to be simulated properly with IKNAT (when the oscillations are hampered, the state space is by far smaller). And *module* because we need the ability to construct from them scalable GRNs to test the performance as the number of modules in a GRN increases.

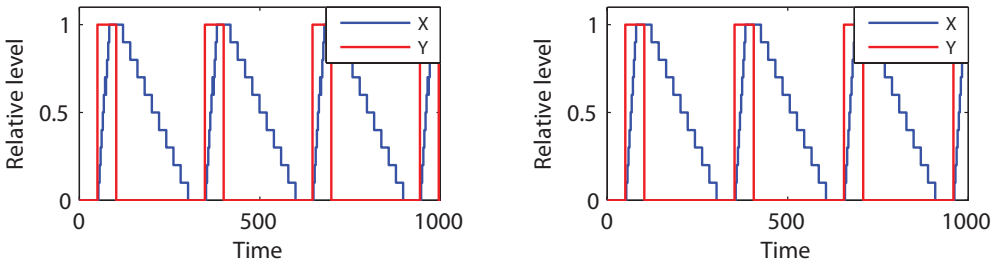
The module is a couple of gene inter-regulated to produce oscillations. One of the two gene has only one concentration level for its product whereas the other can have several ones. Such a structure, shown in Figure 7.20a, although unrealistic, allows easily the creation of oscillations in an insensitive way.

By regulating one gene with several of these modules (e.g., Figure 7.20b) configured with different delays (and thus with different frequencies for the oscillations), we obtain a complex evolution for the regulated gene.

We test here GRNs as the one described in Figure 7.20b with one, two and three modules, with 5 and 10 concentration levels and with delays or not. The delays of the module regulations are set, as much as possible, to induce changes in the product concentration levels of the regulated gene, with for this latter gene a number of concentration levels equal to the number of concentration levels in the modules.

For the version with intervals, the bounds for each delay are set to the value of the delay $\pm 10\%$, with a minimum interval size of one or two for the little delays.

Note, to simplify, that for a module, the delays specified for the dilution/degradation of X and for its regulations are the same at each concentration level (whence the constant slopes in Figure 7.21).



(a) IKNAT approach **(b)** New approach

Figure 7.21: A simulation of a module stated in Figure 7.20a with the IKNAT approach and the new approach.

Model checker assertions

Two assertions are submitted to the model checker. The first, `[E<> globalTime > 10000]`, is a reachability assertion which asks whether there is a state where the time is bigger than 10,000. The second assertion, `[A[] true]`, is a safety assertion which imposes the browsing of all the state space by the model checker to verify the property *true*. For this last assertion, a breadth first search is used whereas for the first one a random depth first is conducted.

These two simple assertions represent two extreme requests (one very easy and one very difficult) whose performance results encapsulate most of the intermediate assertions. Their results can be thus considered as representative of the performances we can expect from the assertions submitted to the model checker.

Noted that for the first one, the property *true* can be replaced by the property *no deadlock* which also imposes browsing all the state space with a result however slightly longer (but in the same scale of time).

Environment

The version of Uppaal used is 4.0.13 which is the most recent version available outside the development snapshot (which seems to improve the performance). The computer used for the performance tests has a processor Intel T7200 (dual-core) at 2 GHz, with 2 GB of Ram and Windows XP SP3 32 bits. A minimal set of programs was running during the benchmarks to avoid as much as possible the perturbations coming from them.

7.5.2 Benchmarks and results

General benchmarks

The first benchmarks are done with the stability extension on the extreme levels for IKNAT and with the on-the-fly calculation variant for the new approach. For this latter, an upscaling of 10,000 is used. The other parameters for both approaches are left to their default value. The GRNs of the benchmarks are denoted by "GM" followed by the number of their modules. The subscripted number indicates the number of concentration levels used. The subscripted star (*) denotes the benchmarks with intervals of time.

The results are given in duration (in second and including the loading time of the model checker) for the first line and in number of states explored by the model checker for the second line. Note that it is also possible to get results about the memory consumption with the number of stored states. But this information is not really relevant for us since the tests never reach exceptional values with the experimented models (the time is by far a more preoccupying aspect).

Table 7.7 shows the results for the reachability assertion. Since these results can depend on the path followed in a random mode, five occurrences of each test were used and the average results are provided. We can however mention that in our experience the standard deviation is small.

	$GM1_{10}$	$GM1^*_{10}$	$GM2_5$	$GM2^*_5$	$GM2_{10}$	$GM2^*_{10}$	$GM3_5$	$GM3_{10}$
IKNAT	0.21	7	1.35	24.7	1.08	14.5	1.44	1.33
	5,990	30,338	14,318	94,902	15,043	74,171	19,924	18,839
New	0.14	0.37	0.73	5.3	0.42	2.28	0.66	0.62
	2,491	2,681	6,059	6,442	5,985	6,685	8,050	8,022

Table 7.7: Performance results for the reachability assertion ($globalTime > 10,000$). The first line is the time (s) required by the model checker to process the assertion. The second line is the number of states explored.

From this table of results, three main observations can be done. Firstly the IKNAT approach has bigger numbers of states explored, which comes from its event mechanism. This difference led the new approach to be quicker.

Secondly, in both approaches, the models with intervals are longer with a bigger number of states explored (above all in IKNAT). This can be explained by the fact that with the intervals, there is more possibility to change or not the concentration levels, creating therefore a larger state space to manage. A complementary explanation is that Uppaal can favor little delays and thus needs to fire more transitions to reach the time 10,000.

Thirdly, using 5 concentration levels instead of 10 gives worst results. The reason for such a phenomenon could be explained by the fact that with less concentration levels, the products X of the modules reach quicker their extreme activity levels. Therefore, the oscillation frequency of the GRNs is higher and entails a bigger state space for their automaton in 10,000 units of time.

The next table (7.8) presents the results for the safety assertion browsing the whole state space. Here, the tests are performed only one time since the results are very stable from one occurrence to another (even for the shortest). Several tests implying time intervals are not given because they require a too long processing time (by far more than one hour). These tests are replaced by the GM_5 and GM_5^*.

	$GM1_5$	$GM1_5^*$	$GM1_{10}$	$GM1_{10}^*$	$GM2_5$	$GM2_{10}$	$GM3_5$
IKNAT	0.06	5.88	0.1	18.6	2.7	4.39	3,633
	407	34,778	736	113,684	42,526	71,425	6,190,335
New	0.06	0.24	0.08	76.4	3.4	3.85	1,330
	121	2,126	252	107,098	23,488	49,485	1,458,014

Table 7.8: Performance results for the safety assertion browsing the whole state space. The first line is the time (s) required by the model checker to proceed the assertion. The second line is the number of states explored.

As we can see, the assertion entails exponential increases of the times and of the state spaces relatively to the complexity of the GRNs and above all relatively to the use of intervals. The increasing complexity between the models with 5 and 10 concentration levels seems more "normal" here. And in the majority of the tests, the new approach is quicker with a lower state space. The only major exception is the GRN $GM1_{10}^*$, where we do not know why there is a so long time for the new approach. With the development version of Uppaal 4.1, this exception is still present but with only a time of 30 seconds.

Sometimes, the number of states explored is lower than with the existential assertion. This is because the *globalTime* clock is not used. Indeed, two visited states, differing with only the value of that clock in the reachability tests, are the same without and therefore are counted as only one explored state in the tests here.

Variants of the new approach

We have made several tests with the other main variant of the new approach which is the pre-calculation variant and with different parameters for the on-the-fly calculation. The results is that, with equal state space, the performances do not change significantly.

	On-the-fly calculation	Pre-calculation
$GM2_{10}$	3.85	3.98
	49,485	50,899
$GM1^*_5$	0.24	0.25
	2,126	2,142
$GM1^*_{10}$	76.4	77.32
	107,098	111,660

Table 7.9: Performance results for the test $GM2_{10}$, $GM1^*_5$, and $GM1^*_{10}$ with the on-the-fly calculation variant and the pre-calculation variant of the new approach, and with the safety assertion. The first line is the time (s) required by the model checker to process the assertion. The second line is the number of states explored.

Table 7.9 presents the results for the comparison between the on-the-fly calculation variant (upscaling of 10,000) and the pre-calculation variant. As it can be seen, there is no significant difference between both variants. At most, there are a number of state explored a few percents bigger in the pre-calculation variant, with a time in correspondence with this. From that, and if we assume that the number of states explored can be representative of the precision of the approach, we can consider that an upscaling of 10,000 with the on-the-fly calculation variant is sufficient to almost reach the precision of the pre-calculation variant. In order to confirm this, we have tested the on-the-fly calculation variant with an upscaling of 10,000,000, and, effectively, there is not significant change compared to the upscaling of 10,000.

From these results, it seems that the power consumed by the time calculations in the model checker is derisory and does not legitimate the use of the pre-calculations in the model-builder, and this especially when the configuration of large upscaling is not a problem and allows a precision almost equivalent to the precision reached by the pre-calculation variant. Therefore, this last variant seems very useless.

Note that in the continuity of this questioning, it would be interesting to see in what extend the incontrovertible rounding operated in both variants affects the trajectory precision of the timed model. Such an analyses can be done by a comparison with a real automaton or with the Uppaal discrete automata where the time is slowed down for all the delays (i.e., increase the delays proportionally) and for the bounds in the assertions of the model checker.

7.5.3 Clock reduction

In Uppaal, an optimization called clock reduction is used to avoid to discriminate uselessly the states with an unused clock. For example, a clock is considered unused

in a state when the active locations reset the clock on each of its outgoing switches (without guard expression over the clock). In such a case, the state signature does not contain the clock value. This optimization leans on static analysis. It is interesting to consider this optimization in order to explain some performance results.

In the IKNAT approach, the locations *begin* of the species-activity template and of the reaction template lead to such a situation where the clock c is considered as unused. Any attempt to bound the evolution of the clock in these locations - in order to limit the state space - will only result in a reduction of the performance. Indeed, unless to use stopwatches, the only way to limit the value of the clock is to put a self-switch on these locations which resets the clock periodically. But actually this supposed optimization discards the clock reduction on c since the clock need to be tested before being reset. This result in a worse performance caused by an expansion of the state space and by more transitions being fired.

More generally, for the three approaches, the safety assertions which do not involve the *globalTime* clock, allows also the inactivation of the clock by the clock reduction optimization. Therefore, the size of the state space is the same with or without the *globalTime* clock. The only difference is an overhead in the processing time of about 20 % produced by the management of the clock by the optimization.

Finally, this optimization provides the ability to bound efficiently the size of the state space browsed for an assertion. Indeed, by creating a process with only one location where the invariant is "$globalTime < K$", the state space of the automaton is limited by the time value K. If the *globalTime* clock is not mentioned on the assertion, the clock reduction still considers that the *globalTime* clock is unused. From this, the state space explored is the same than without the process but cropped by the time limit K.

	Normal	Reset every 5 units	Deadlock at 10,000
$GM2_{10}$	4.39	32.4	2.3
	71,425	116,845	48,882

Table 7.10: Performance results for the test $GM2_{10}$ with the IKNAT approach and the safety assertion. The first line is the time (s) required by the model checker to process the assertion. The second line is the number of states explored. The first column of results is the normal situation. The second column are the results when the clocks c are reset every five units of time in the locations *begin*. The last column are the results where a deadlock is created after 10,000 units of time. The limit of time before deadlock which allows the model to express all the 71,425 states of the normal result seems to be just below 15,000.

7.5.4 Remarks

The second series of benchmarks with the safety assertion underlines very well the difficulties associated to the explosion of the state space. But it is important to keep in mind that the assertion used is the worst possible and that the GRNs experimented, by their oscillations, are themselves not easy to manage.

For the comparison between the new approach and IKNAT, the former is often (although not always) quicker with a lower state space.

Finally, it is worth to note that Uppaal does not make use of any parallelism which can reduce the exploration time on many-core processors. This ability which can strongly accelerate the exploration of the state space is currently considered by the Uppaal developers.

7.6 Conclusion

The examples developed in this chapter show to what extent we can use one approach or another, i.e. on the one hand the convenient boolean approach enabling easy and quick modelling, and on the other the more expressive IKNAT and new approach, both based on the timed model. Compared to IKNAT approaches, the new approach stands out with its ability to model more sensitive situations and with better performances on average.

For the timed-boolean approach, neither the incapacity of the regulations to influence the reactivity nor to handle the history of a gene evolution in an activity level were a problem to model the three modellings case studies. But our case studies do not prove that it will be always like this. A situation with more regulators for a gene can be a problem to the extent that it is not possible to determine proper delays for each configuration of the regulators.

For the new approach, the pre-calculation variant for the delay computing seems to be useless.

Finally, with these case studies, it was also demonstrated that it is possible to model more complex GRNs by being more inspired by their ODE model for the specification of the delays in the IKNAT and new approaches. This opens the door for new possible refinements of these approaches and more generally of the timed model.

8

Conclusion and future work

In this research two methods to model GRNs on timed automata were exposed. The first one, the timed-boolean approach, is well known and relies on a boolean abstraction (i.e., the Thomas's formalism) where the time aspect is introduced in the form of constant delays between the changes of activity levels. This approach, by its high abstraction, suffers from several limitations to express some situations and aspects of the GRNs. However, by this abstraction, the approach is easy to parametrize and produces automata with a limited state space. This limited state space is a strong advantage for the efficient application of parameter synthesis, a process which, from a described situation, delineates a set of parameters which can induce the situation in a given model. The result of such a process in a timed-boolean model can be used as a starting point for further researches on less abstract model like ODE models.

The second method, initiated by the IKNAT approach to model ISNs, was extended in this thesis to the modelling of GRNs and was formalized in the timed model. The principle of the IKNAT approach leans on the delays to drive the dynamics of the GRNs. This aspect induces a better expressiveness than the timed-boolean approach, but also a bigger state space for the automata and a more complicated parametrization of the delays. Moreover, the implementation followed by IKNAT raises some stability problems for the concentration levels of the gene products. To solve these stability problems, we have developed and experimented with success the new approach which implements more directly the timed model. In addition, the new approach produces automata with smaller state spaces and is more flexible for prospective evolutions.

Future work

Although the new approach seems to be promising, it really needs to be further explored before claiming that it can be useful to model GRNs. A first practical task to be achieved in the continuity of this thesis and when Uppaal allow it, is the exploration of the variant with stopwatches. The point is to determine to what extend the use of the stopwatches for the history accuracy is important with regard to the decidability of the reachability problem. Another interesting practical task is to test the new approach on real timed automata, which are more adapted since - with real timed automata - we do not need the apparatus to manage the discrete aspect.

More importantly, the specification of the delays is off-putting and time consuming. To tackle this aspect, the first task is to achieve the exploration of the possible timed model refinements. As it was introduced in the case study of the oscillatory pattern, it is possible to facilitate the specification of the delays by reintegrating the basal production rate to the β parameters of the inhibitors. Another improvement would be to create guide-lines for the specification of timed models. These guide-lines could, from well-known formalism used to model GRNs as ODEs, facilitate or drive the choice of the number of concentration levels, the use of gates, and the specification of delays. We can even think of an automatic generation of delays from simple ODE formulas. With this automatism we can imagine for example that the new approach could be used to quickly assess the relevance of an ODE model with noise, before going deeper with more complicated and accurate models, or it could be used to verify some properties thanks to the decidability of the reachability problem.

Finally, in the other direction, it could be very interesting to apply the parameters synthesis to the new approach. The idea, as already explained, is not to use the model of a GRN anymore to verify the GRN dynamics induced by some parameters, but to find the parameters compatible with a description of the GRN dynamics. This process, which is already applied to the timed-boolean approach in [1], can however be (really) less efficient with the new approach since the state space of its automata is by far bigger than the state space of the automata in the timed-boolean approach.

Appendix A

GRN motifs

A motif is a pattern of regulations which is often encountered in various GRNs because of the advantages it brinks. These advantages can be, for example, a better resistance of the GRN function against mutations or a better reactivity of the GRN in reaction to changes in the environment, allowing the cells to be more fitted to the surrounding conditions. From these advantages, the corresponding motifs are selected by the evolution and are therefore more present in GRNs than random structures. Three basic and well-known motifs are presented here with trajectories created from ODE models (see 2.2.1). For these motifs, a more complete presentation is available in the book of Uri Alon [2].

A.1 Self-regulation motifs

A self-regulation (or auto-regulation) motif is a motif where a gene regulates itself directly in a positive manner (self-activation) or in a negative manner (self-inhibition).

Self-inhibition

The self-inhibition for a gene improves the reactivity of its product by diminishing its response time. As stated before for ODE models, the response time can be modeled for a product by the logarithm of 2 divided by the degradation rate of the product ($T_{1/2} = log(2)/\alpha$). By setting a self-inhibition, the resulting effect looks like if the gene product was increasing itself its degradation rate, and therefore decrease its response time. This situation which appears ludicrous at first time can be easily grasp by the fact than, if the gene product "actively increases" its degradation rate when it

increases its concentration, then it can have a bigger rate of production for the same steady state concentration. More precisely, this can be described as follows: the product concentration begins to grow quicker, and then slows down more strongly its production rate when it reaches the steady state concentration.

Another related advantage is that a self-inhibition brings a better robustness to the steady state concentration of a gene product against the fluctuations of its production rate. Indeed, if the gene is less expressed, it is less repressed by its production and then more expressed, this closing the full circle.

Self-activation

The self-activation for a gene augments the response time of its product (for the reverse reasons than for the self-inhibition) and allows the gene to stay expressed by itself independently of its promoters. This motifs is very common in cell involved in a differentiation or specialization process. Indeed, once the cell begins a process of differentiation, several genes with a self-activation are expressed to prevent a reversion to the initial situation and then kept the cell stuck in the new configuration.

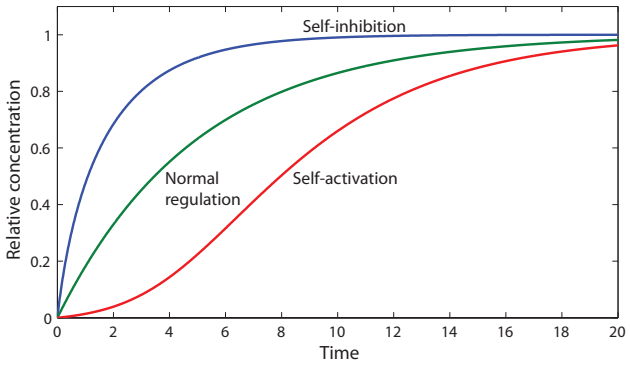


Figure A.1: Graph with the dynamics of the self-regulation motifs. The gene is promoted in a normal way in the green line. In the red line, a self-activation is added with tuned parameters to slow-down the speed of the increase. In the blue line, a self-inhibition is added with tuned parameters to accelerate the speed of the increase. For both cases, $K = 0.5$ and $n = 1$.

A.2 Feed forward loop motifs

A Feed Forward Loop (FFL) is a motif which involves three genes in a triangular structure. In this kind of structure, there is a gene X which regulates another gene

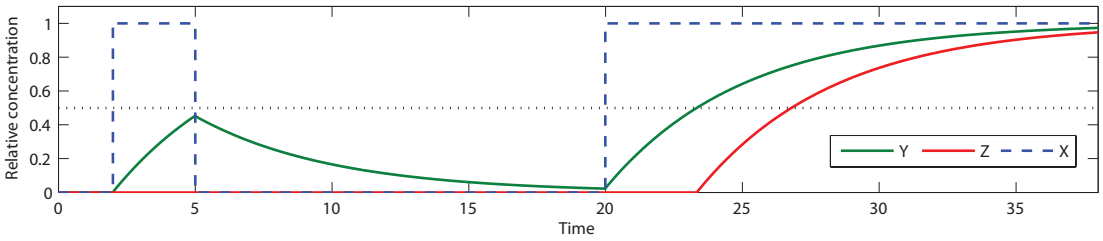


Figure A.2: Graph with the dynamics of a C-FFL AND. The activity of X is a boolean abstraction. The threshold of Y for the AND gate is set to 0.5. In the first pulse of X, Y does not have the time to reach a concentration of 0.5 and then the production of Z is not started. The second activation of X is sufficiently long to start the production of Z.

Y. And both gene regulate a third gene Z. According to the kind of regulation, the behavior of the motif can be different. We present here two useful kinds of FFL often present in GRNs by the advantages they bring. The first kind is the *coherent FFL* (C-FFL) and the second is the *incoherent FFL* (I-FFL).

Coherent FFL

In coherent FFL motifs, all the regulations are positive (X promotes Y and Z, Y promotes Z). There are two C-FFL which differ by the cooperation used in Z to deal with the two incoming regulations. In the first case, it is an AND gate. This means that Z needs to be promoted by X and Y simultaneously to be synthesized. The purpose of such a structure is to prevent the expression of Z when X is only expressed for a period lower than a given delay δ. This period is determined with the time required by the regulator Y to switch from an inactive state (the Y product is under the threshold of the gate) to an active state (the product is over the threshold) when its promoter (X) is active. So, when X is active, the concentration of the Y product increases, after a delay δ without accident, Y becomes in turn active and the production of Z can begin. But if before this point, X became no more active, Y stops its production and there is not a "false start" in the production of Z. This motif can be used if for example the Z product launches a very important and heavy process in reaction to the signal activating X, therefore the delay ensures that the situation of the signal is durable and thus that it is worthy to launch the process.

The C-FFL with OR gate produces the same kind of delay but for stopping the promotion of Z. So, if X is no more active for a moment, this moment has to be sufficiently long to stop the expression of Z.

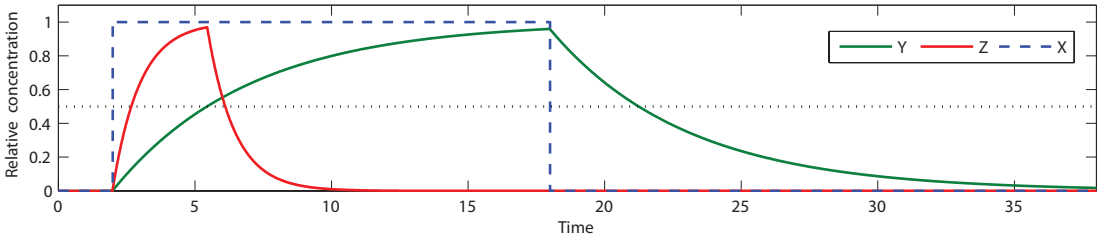


Figure A.3: Graph with the dynamics of an I-FFL. The activity of X is a boolean abstraction. The threshold of Z for the AND gate is set to 0.5. Once X is activated, Z increase quickly its concentration. When Y reaches the threshold 0.5, Y inhibits Z which then falls.

Incoherent FFL

For the second kind of FFL, the Incoherent FFL, we retrieve the same regulations as in the C-FFL with an AND gate but with one difference: here the regulation between Y and Z is an inhibition instead of an activation, and thus the gate *X AND Y* becomes a gate *X AND NOT Y*. The dynamics allowed by such a structure is the generation of brief activity pulses for Z. Indeed, when X is active and not yet Y, Z is only promoted. But after a given delay δ, Y which is promoted by X becomes active and thus inhibits Z by turning off the result of the gate. During the delay δ, Z was promoted without inhibition, reaching therefore a high concentration during a brief period.

A.3 Cascade motif

The cascade motif is a very simple pattern where there is a succession of promotions from one gene to another. For example, if we have a cascade with the genes g_1, g_2, g_3, and g_4, with g_i promoting g_{i+1}, once g_1 is active (in a sufficient concentration), it actively promotes g_2, and the activation is propagated on the cascade with a delay since it takes time for a gene to switch from an inactive state to an active state. When the promotion of g_1 ceases, the concentration of the g_1 product decreases and all the genes in the cascade, one after another, stop their activity. We can find this kind of structure, for example, in the creation process of the flagella in some cells. The genes at the begin of the cascade promote the synthesis of proteins for the root of the flagella, and the genes at the end of the cascade promote the synthesis of proteins for the end of the flagella. With the temporal delay and the order induced by the cascade, the different pieces of the flagella are ready in time and in the good order to be put together.

Appendix B

Model-builder

The model-builder is the software developed to essentially generate, from a GRN, the Uppaal timed automaton. In addition to this main function, the model-builder can submit the automata created to the Uppaal model checker and can parse the returned traces. This appendix documents the use of the model-builder and gives some explanations about its implementations.

B.1 General process

The general process of the use of the model-builder is as follows. First, an XML file describing a GRN is created according to the syntax of the modeling approach selected (see Section B.2). Beside this XML file, an assertion for Uppaal is specified (see B.3). Then, the XML file is given to the model-builder (for the parametrization see B.4) which generates an Uppaal file with the corresponding timed automaton.

Once the automaton file is created, the model checker of Uppaal is launched on it and with the specified assertion. The model checker of Uppaal is embodied in the process Verifyta provided with Uppaal.

When the model checker has finished its job, the result is printed in the console and, if a trace is returned, a file is generated with the trace. This trace file is parsed by the model-builder to generate a trajectory file (.fin) containing only the information relative to the concentration level of the gene products (see B.5). Once the parsing is finished, and if specified, the model-builder can call MATLAB to chart the trajectories.

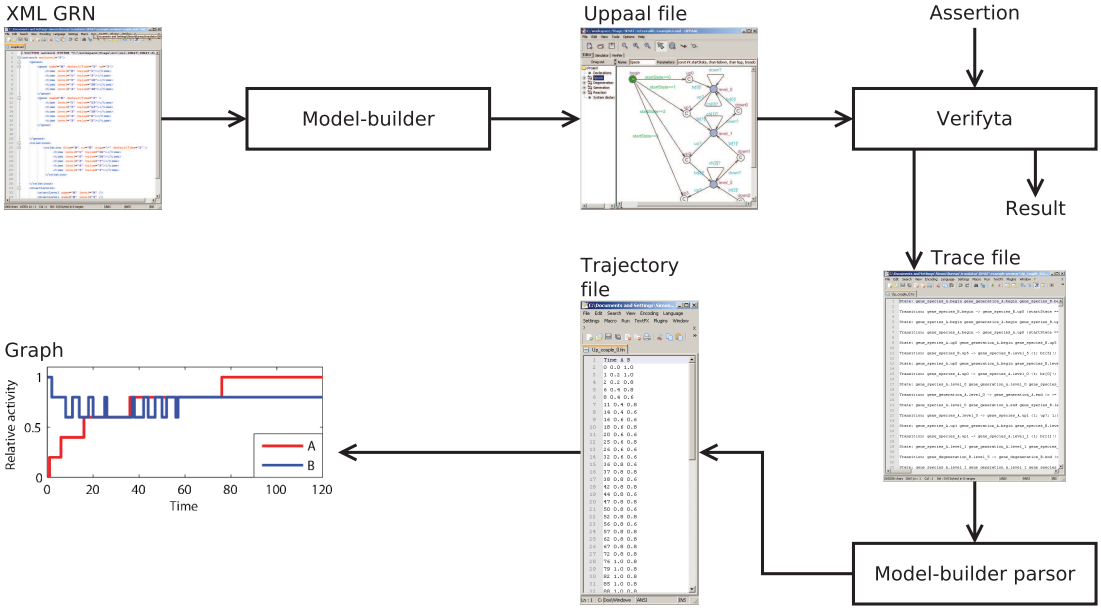


Figure B.1: The general process of the model-builder

B.2 XML formats

Each GRN submitted to the model-builder is described in an XML file. For each approach, there is a DTD file which specifies what is required for the approach. These DTD files are described subsequently.

B.2.1 Timed-boolean approach

A GRN for a timed-boolean model is defined by a *network* element constituted of at least one *gene* element. The *defaultMaxLevel* attribute of the network is the default maximum activity level used for each gene product.

A gene is identified by a *name* constituted with numbers and letters (the underscore and the hyphen are not allowed). Moreover, a name cannot contain the word "DOWN" and "UP". The *targetLevel* of a gene product is the default value assigned to the logical parameters of the gene. The maximum activity level of a gene product can be specified in *maxLevel* to override the value of the default maximum activity level assigned to the network. The attributes *dUp* and *dDown* specify the default time value used to increase or decrease the product activity level of the gene by one unit. To specify an interval of time, one can use *dUpMin*, *dUpMax*, *dDownMin*, and *dDownMax*. With the element *time*, it is also possible to specify some particular time values depending on an

timedBoolean.dtd

```
<!ELEMENT network (gene+, startLevel*) >
<!ATTLIST network defaultMaxLevel CDATA "1">

<!ELEMENT gene (set|regulation|and|or|time)*>
<!ATTLIST gene name ID #REQUIRED>
<!ATTLIST gene targetLevel CDATA "0">
<!ATTLIST gene maxLevel CDATA #IMPLIED>
<!ATTLIST gene dUp CDATA "1">
<!ATTLIST gene dDown CDATA "1">
<!ATTLIST gene dUpMin CDATA #IMPLIED>
<!ATTLIST gene dUpMax CDATA #IMPLIED>
<!ATTLIST gene dDownMin CDATA #IMPLIED>
<!ATTLIST gene dDownMax CDATA #IMPLIED>

<!ELEMENT set (resource)* >
<!ATTLIST set targetLevel CDATA #REQUIRED>

<!ELEMENT resource EMPTY>
<!ATTLIST resource name IDREF #REQUIRED>
<!ATTLIST resource inv CDATA "0">

<!ELEMENT time EMPTY>
<!ATTLIST time level CDATA #REQUIRED>
<!ATTLIST time up CDATA "1">
<!ATTLIST time down CDATA "1">
<!ATTLIST time upMin CDATA #IMPLIED>
<!ATTLIST time upMax CDATA #IMPLIED>
<!ATTLIST time downMin CDATA #IMPLIED>
<!ATTLIST time downMax CDATA #IMPLIED>

<!ELEMENT regulation EMPTY>
<!ATTLIST regulation from IDREF #REQUIRED>
<!ATTLIST regulation sign CDATA "+">
<!ATTLIST regulation threshold CDATA #REQUIRED>
<!ATTLIST regulation targetLevel CDATA #IMPLIED>

<!ELEMENT and (resource|and|or)+>
<!ATTLIST and targetLevel CDATA "1">
<!ATTLIST and inv CDATA "0">

<!ELEMENT or (resource|and|or)+>
<!ATTLIST or targetLevel CDATA "1">
<!ATTLIST or inv CDATA "0">

<!ELEMENT startLevel EMPTY>
<!ATTLIST startLevel name IDREF #REQUIRED>
<!ATTLIST startLevel level CDATA #REQUIRED>
```

activity level of the gene product. The attribute *level* of this element *time* specifies the activity level of the gene product for which the specified value is used. For example, an up time of 5 for the level 1 specifies that the delay to switch the gene product from the level 5 to the level 6 is of 5 time units.

A gene can have several *regulation* elements, where the *from* attribute is the regulator name, the *sign* attribute ("+" or "-") specifies whether the regulation is an activation or an inhibition, and where the threshold of the regulation is specified in the *threshold* attribute.

Note that the value specified for the maximum activity level of a product is arbitrary, but the Thomas's formalism requires normally minimal numbers of activity level.

Each logical parameter for a gene is specified in the gene by an element *set* and is constituted of *resource* elements where the attribute *name* is a name of a gene regulator.

The attribute *inv* of a resource inverse the resource and is used only with the boolean elements (see after). The attribute *targetLevel* of a set specifies the target level of the gene product when the resources of the set are present.

The logical parameters can also be generated by specifying the attribute *targetLevel* in the regulations or by using the boolean elements (*and* and *or* elements). In the former case, a logical parameter is created only for the resource of the regulation. In the latter case, it is possible to construct a logical formula over the resource identified by the name of the element *resource*. The target level of the logical parameters created from this formula is the value of the *targetLevel* attribute of the first boolean element (the one which contains the others). The *inv* attribute of a boolean element, if set to 1, act as a NOT over the boolean result of the element, as the normal rules of the logic. For example $\neg(R_1 \wedge (R_2 \vee \neg R_3)) = (\neg R_1 \vee (\neg R_2 \wedge R_3))$, where R_x is a resource and $\neg R_x$ denotes the absence of the resource. To specify the absence of a resource, the attribute *inv* of the element *resource* can be set to the value 1.

To specify for a gene product a start activity level different from 0, one can use the element *startLevel* with the name of the gene and the desired start level.

At last, two tricks: to specify a target level which is the maximum of the gene product, use “max”, and to specify an infinite time value, use “oo”.

B.2.2 IKNAT approach and new approach

The IKNAT and new approaches share the same DTD. This DTD is similar in several points in its syntax and in its semantic to the DTD of the timed-boolean approach. So, only the parts which differ are described here.

The first difference is that the time is specified in the *gene* element only for values resulting from the summation of its degradation and its basal activity level (the $1/\alpha'$ values). The time values specifying the effect of the regulations (the $1/\beta'$ values) are specified in the same *time* element as for the gene but in the *regulation* elements. Note that as all the values are positive, the model-builder changes itself their sign according to the type of their regulation (for the $1/\beta'$ values) and their position relative to the target concentration level (for the $1/\alpha'$ values).

The second difference is related to the threshold attribute in regulations. This one is used only if the regulation is in a boolean element (*or* element or *and* element). However, it is also used to configure the default value of the regulation times to the infinity value if the level of the regulator is under the threshold.

IKNAT.dtd and new.dtd

```
<!ELEMENT network (gene+, startLevel*) >
<!ATTLIST network defaultMaxLevel CDATA "1">

<!ELEMENT gene (time|regulation|and|or)*>
<!ATTLIST gene name ID #REQUIRED>

<!ATTLIST gene targetLevel CDATA "0">
<!ATTLIST gene maxLevel CDATA #IMPLIED>
<!ATTLIST gene dTime CDATA "1">
<!ATTLIST gene dMinTime CDATA #IMPLIED>
<!ATTLIST gene dMaxTime CDATA #IMPLIED>

<!ELEMENT time EMPTY>
<!ATTLIST time level CDATA #REQUIRED>
<!ATTLIST time value CDATA "1">
<!ATTLIST time minValue CDATA #IMPLIED>
<!ATTLIST time maxValue CDATA #IMPLIED>

<!ELEMENT regulation (time*)>
<!ATTLIST regulation from IDREF #REQUIRED >
<!ATTLIST regulation sign CDATA "+">
<!ATTLIST regulation threshold CDATA "1">
<!ATTLIST regulation dTime CDATA "1">
<!ATTLIST regulation dMinTime CDATA #IMPLIED>
<!ATTLIST regulation dMaxTime CDATA #IMPLIED>

<!ELEMENT and (regulation|and|or)+>
<!ATTLIST and sign CDATA "+">
<!ATTLIST and inv CDATA "0">
<!ATTLIST and time CDATA "1">
<!ATTLIST and minTime CDATA #IMPLIED>
<!ATTLIST and maxTime CDATA #IMPLIED>

<!ELEMENT or (regulation|and|or)+>
<!ATTLIST or sign CDATA "+">
<!ATTLIST or inv CDATA "0">
<!ATTLIST or time CDATA "1">
<!ATTLIST or minTime CDATA #IMPLIED>
<!ATTLIST or maxTime CDATA #IMPLIED>

<!ELEMENT startLevel EMPTY>
<!ATTLIST startLevel name IDREF #REQUIRED>
<!ATTLIST startLevel level CDATA #REQUIRED>
```

The last difference is that here the *and* and *or* elements are not used to configure logical parameters with a corresponding formula but they are used to specify boolean gates. Each boolean element directly nested in a *gene* element represent a boolean gate. The effect of a boolean gate ($1/\beta$) is specified by the attributes *time*, *maxTime* and *minTime*, the last two being for an interval. The attribute *sign* of a gate is used to define the kind of effect associated with the gate. To refine the boolean formula of a gate, some additional boolean elements can be defined inside a gate element in an imbricated structure. Note that for these imbricated boolean elements, the attribute *time* and *sign* have not to be specified. Finally, the attribute *inv* of a boolean element can be used to inverse the boolean result of the element.

The only word not allowed in the names of the genes is here the word “level”. Note that here the target level of a gene means the basal concentration level of its product.

B.3 Uppaal assertions

There are currently two kinds of properties easily specifiable in the assertions (or requests) for the Uppaal model checker: properties over the concentration (activity) level of gene products and properties over the absolute time.

Concentration level properties

The concentration level of each gene product is accessible in each approach by an integer variable whose name is the same as the gene name. With the IKNAT and new approaches, it is also possible to use the locations dedicated to the concentration levels of each gene product. Such locations are accessible by "*species_NAME.level_i*" where *NAME* is the name of a gene and i is a number of concentration level. But there is no advantage to use these locations since there are not the only activated in their process and since we cannot specify binary comparisons ($\leq, \geq, =, ...$) over them.

Absolute time properties

The properties on the absolute time are only available by launching the model-builder with the option *ref* which creates automatically a never reset clock as a reference point. The name of this clock is "*globalTime*". Because the clock is never reset, it increases the size of the state space since two states need to have the same value for the clock to be considered equal. The reduction clock optimization of Uppaal can prevent this, but not with certain assertions referring the clock. With such assertions, it is inadvisable to refer the clock in them if they require the exploration of a big part of the state space, the risk being to get an endless exploration of an infinite state space.

Examples

- `[E<> X > 5 && globalTime > 1000]` states that it is possible that the concentration level of X is bigger than 5 after the time 1000.
- `[A[] X > 5 && Y = 0 ]` states that the concentration level of X is always over 5 and Y is always absent.
- `[A[] (X > 0 && Y = 0) || (X = 0 && Y > 0)]` states that both X and Y are never present simultaneously.

- `[(X > 0 && Y = 0) --> (X = 0 && Y > 0) && (X = 0 && Y > 0) -- > (X > 0 && Y = 0)]` states that there is an infinite cycle between the situation where X is present and not Y, and the situation where it is the reverse.

From these examples, we can see that it is possible to inquire a GRN with assertions about several properties like the presence of cycles, the bi-stability, ...

B.4 Model-builder arguments

In this section are described the arguments specified to launch the model-builder, with first the arguments which are the same for the three approaches, and after the particular arguments of each approach. The italic value after the name of an argument is the default value of the argument. An argument with a default value is not mandatory.

Arguments for all approaches

source		The path of the GRN XML file.
query	*query.q*	The path of the file where the assertion for Uppaal is saved.
verifyta	*verifyta.exe*	The path of the Verifyta executable. If the *frozen* option is used, Verifyta has to be the version for Uppaal 4.1.
model		Specifies the model to create. Set 0 for the timed-boolean model, 1 for the IKNAT model, 2 for the new model, or any combination of these numbers (e.g., "12" launch the model-builder for the IKNAT model and the new model).
todo	*0*	Specifies what to do. Set 1 to generate the Uppaal automaton, 2 to proceed the automaton in Verifyta, 0 to do both.
ref	*1*	If set to 1, a never reset clock *globalTime* is created to be used as a reference in the assertion.
block	*0*	If set to a value x bigger than 0 and if *ref* is set to 1, creates a deadlock on the automata when the *globalTime* clock is equal to x.
nb	*1*	Specifies the number of runs of the model checker on the model (to test the performance or to get more than one trace).
keep	2^{31}-*1*	Specifies the maximum time value of the events retained from the trace returned by Verifyta.

comp	*0*	If set to 1, the time gaps in the trajectories are filled (for each time unit, there is a tuple for it in the trajectory file).
until	*1*	If the time of the last point in the trajectories (see Section Result) is lower than this value, an extrapolated point is created to reach this value. The value specified has to be used consistently with the assertion to avoid an incorrect extrapolation. If the *globalTime* is not used ($ref = 0$), this option is inadvisable.
trace	*1*	If set to 1, a trace is asked to Verifyta and, if Verifyta returns a trace, a trajectory file is generated from this one.
show	*0*	If set to 1, and if *trace* is set to 1, MATLAB is launched automatically to show the trajectories generated from the traces. The libraries of MATLAB has to be installed for this option.
print	*0*	If set to 1, a textual summary of the loaded GRN is printed in the console.
o	*2*	Indicates the search order followed by Verifyta. 0 for breadth first, 1 for depth first, or 2 for random depth first.

Argument for the timed-boolean approach

mode	*1*	Specifies the mode used to deal with the improper change of activity levels : 0 for normal, 1 for urgent, 2 for overriding.

Arguments for the IKNAT approach

stability	*1*	Set to 1 to improve the stability of the gene products in the extreme concentration levels.
limClock	*0*	When set to a value x bigger than 0, the clocks are constantly reset after x units of time when they are not used in the species activity processes and in the reaction processes (inefficient option).

Arguments for the new approach

noTab	*1*	Set to 1 to do not compute the delays in the model-builder but to compute the delays on-the-fly in Verifyta.

slow	*1*	If set to a value bigger than 1, slows down the time to keep a better accuracy when the delays are computed and discretized.
scaleUp	*1000*	If *noTab* is 1, sets the value of the upscaling factor used for the calculation of the delays in Verifyta. Has to be set to a value equal or bigger than 1.
bestRound	*0*	If set to 1, and if *noTab* is set to 0, makes the best round in the model-builder when it rounds real values in integer values. This option uses the floor or ceiling rounding according to the type of rounded bound (min or max). Since a minimum bound is rounded to x while the upper bound is rounded to $x+1$, it can entail the creation of small intervals where none are expected .
zeno	*0*	Set to 0 to prevent Zeno-behavior in the automaton. If set to 0, all the lower bounds of delays equal to 0 are set to 1 (with a shift of the upper bounds to the value 1 if necessary). If set to 1, only the upper bounds of delays equal to 0 are set to 1.
frozen	*0*	Set to 1 to use the variant with stopwatches.

B.5 Results

The result coming from Uppaal for a model and a given assertion is, *yes*, *no* or *maybe* (only with the argument *frozen* for the new approach). Besides this result, some information about the time and number of explored and stored states are displayed in the console (for each occurrence and the average when several runs are done, see argument *nb*).

When a trace is returned by Verifyta, it can be parsed (option *trace*) and the trajectories of the gene products are saved in a file with a *.fin* as extension. This file consists of a header line with the word "Time" followed by the name of the genes. After this, each line contains a tuple with a time value followed by the relative concentration levels of the products.

```
Time X Y Z
0 0.0 0.0 0.0
10 1.0 0.0 0.0
13 1.0 0.0 0.16666667
14 1.0 0.16666667 0.16666667
16 1.0 0.16666667 0.33333334
```

```
18 1.0 0.33333334 0.33333334
19 1.0 0.33333334 0.5
22 1.0 0.5 0.6666667
25 1.0 0.5 0.6666667
26 1.0 0.5 0.6666667
29 1.0 0.5 0.8333333
30 1.0 0.6666667 0.8333333
```

Listing B.1: The beginning of a results file for a GRN with three genes: X, Y, and Z.

Note that such trajectories cannot be generated with liveness assertions including the *lead to* operator ($--{>}$). Indeed, Verifyta on such assertions can only provide symbolic traces. A symbolic trace is a trace which stands for several concrete traces. In symbolic traces, the values of the clocks are represented with inequality formulae. The parsing of these inequalities to retrieve the evolution of the clocks is very complicated and was not implemented. Note also that the presence of traces is conditioned by the temporal quantifier used. For example, $A[]$ entails only a trace if a counter example to the checked assertion is found.

From such listings of trajectories, with the option *show* it is possible to generate directly a graph with MATLAB if this last is available.

B.6 Implementation

In this section, some aspects of the implementation are briefly described. These aspects are the class structure and the interactions between the model-builder and Verifyta.

B.6.1 Inheritance and implementation class diagrams

The implementation of the model checker is done in Java 1.6. Since all models manipulate a common structure of genes and relations, there is an strong usage of abstract classes and interfaces with, from them, many implementation and inheritance links. The common part of the three approaches is implemented in a package *Model*. The content of this package is, for the important part, the following:

- The abstract class *Model* to define the primitive operation that all approaches have to provide (create the time automaton, print a summary of a loaded GRN in the console, parse the result of a trace, ...).
- The abstract classes *Gene* and *Regulation* to store and to handle the corresponding element of a GRN.

- The abstract classes *Genes* and *Regulations*, extending the class *ArrayList*, to store and handle all the objects *Gene* and *Regulation* of the GRN.
- The abstract class *Gate* to store and handle a GRN gate.
- The interface *Node* and *Link* which are implemented by, respectively, the *Gene* and *Gate* classes and by the *Gate* and *Link* classes. These interfaces are implemented by classes employed in some algorithms which need to store and browse objects of the different types.

For each approach, a dedicated package is created for its classes. These packages inherit and implements massively the classes and interfaces of the *Model* package. Figure B.2 shows the inheritance and implementation links for the NEW and IKNAT packages. Figure B.3 shows the inheritance and implementation links for the timed-boolean approach (*boolMod* package). In these packages, several intermediate classes refining and implementing the *Gate* class can be used depending on the needs of the approaches. Note that to avoid the casting of the objects in these inheriting classes, we massively make use of the class parametrization allowed since Java 1.5.

Note that the previous class diagrams give the most important part of the class structure for the three approaches, although there are other classes unmentioned and dedicated to the algorithms creating the automata.

B.6.2 Interactions between the model-builder and Verifyta

This section details the interactions between the model-builder and the model checker of Uppaal which is Verifyta.

Creation of the Uppaal file

Each automaton submitted to Verifyta has to be stored in an XML file respecting a specific DTD. Therefore, the first step of the interaction begins with the creation of the XML file describing the timed automaton to be processed. To generate the automaton and then its XML file, a package *uppaal* is used (Figure B.4). This package contains classes to represent, each, an important elements in the DTD. For each of these classes, some operations are provided to handle them and to handle their contents. Once a timed automaton is defined, a simple operation can save it in an XML file with respect to the DTD.

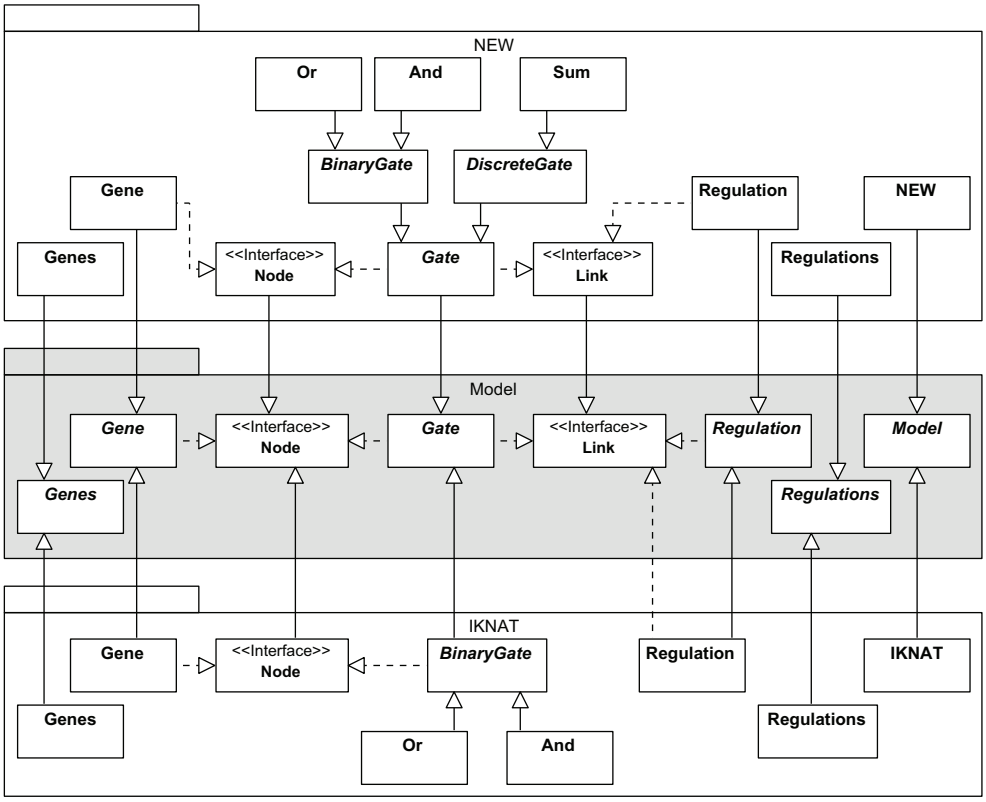


Figure B.2: A class diagram for the implementation and inheritance links of the *IKNAT* and *NEW* packages with the *Model* package.

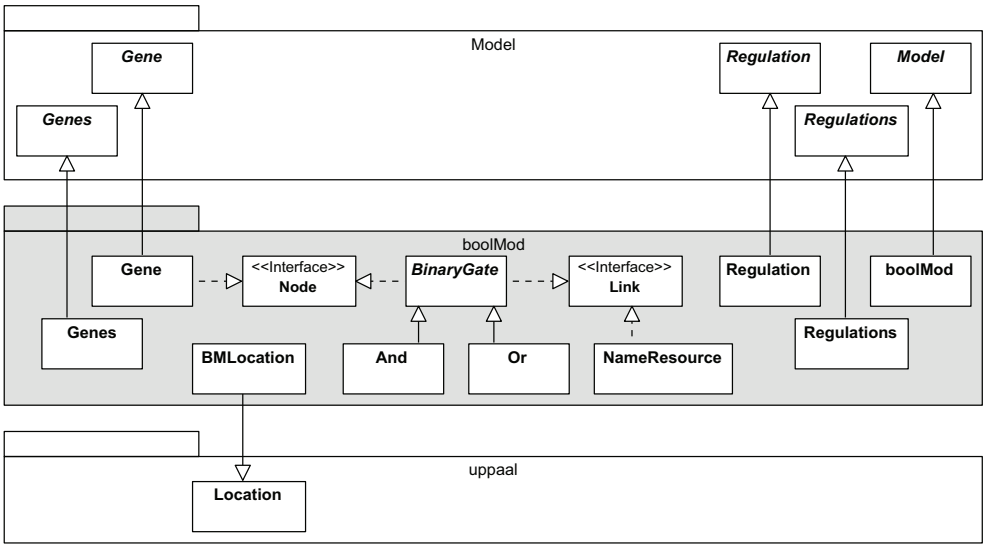


Figure B.3: A class diagram for the implementation and inheritance links of the *boolMod* package with the *Model* and *uppaal* packages.

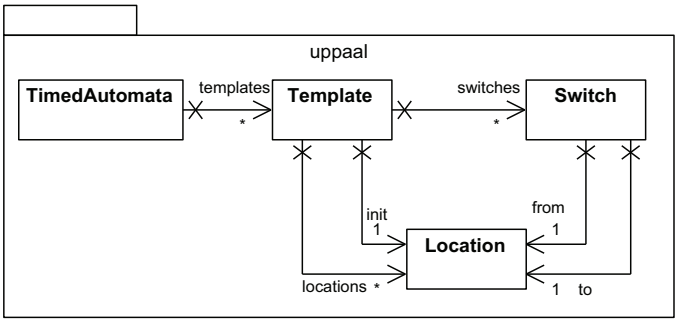


Figure B.4: The main part of the *uppaal* package.

Verifyta command line

Once the file of an automaton is created, the model-builder can call Verifyta. The command line used in Windows is the following:

```
verifyta.exe -oX -t0 -u timedAutomatonFile queryFile
```

Listing B.2: Verifyta command line

where

- **oX** specifies, with the number X, the order of the search in the state space of the automaton (see Section B.4, argument *o*).
- **t0** requests the generation of a trace (not the shortest (t1) or the fastest (t2)).
- **u** induces the printing in the console of some information about the number of explored and stored states.
- **timedAutomatonFile** is the address of the file containing a description of the timed automaton to be processed.
- **queryFile** is the address of the file with the assertion for Verifyta.

Note that since nothing about the trace format is specified, Verifyta provides by default a concrete trace (and not a symbolic trace).

Verifyta traces

There are two ways to get back a trace from Verifyta. The first way uses the argument *-f* to make Verifyta store the trace in a file. However, the trace stored in the file is

in an unusable numerical format. This format is dedicated to be loaded in the GUI of Uppaal. To convert the numerical trace into a textual trace, it is necessary to use a software called Tracer (provided with Uppaal). But to make the conversion, the Tracer requires a compiled version of the automaton associated with the trace. This compiled version of the automaton can be generated by Verifyta if the environment variable *UPPAAL_COMPILE_ONLY* is initialized before. Once this step is done, it is finally possible to convert the numerical trace into a textual trace with the Tracer. This method to get the trace has however a disadvantage: the only traces available at the end of the process are symbolic traces which are very difficult to parse when time intervals are specified in the approach. This difficulty come from the fact that the temporal information are saved symbolically as a result of inequalities with the clocks.

The second way to get back the trace is the only available to get a concrete trace. This method does not use the argument *-f*. In such a case, a textual trace is printed directly on the error stream of Verifyta with in preamble the sentence "*Showing example trace.*". This sentence is used to detect the beginning of the trace and to redirect then the error stream in a file. Afterwards, this file can be parsed following the specificity of each approach.

```
State: species_X.begin species_x.begin species_Y.begin species_y.begin species_A.level_0 X =
    0 x = 0 Y = 0 y = 0 A = 0 species_X.up = 0 species_X.down = 0 species_X.both = 0
    species_X.minUp = 0 species_X.maxUp = 0 species_X.minDown = 0 species_X.maxDown = 0
    species_x.up = 0 species_x.down = 0 species_x.both = 0 species_x.minUp = 0 species_x.
    maxUp = 0 species_x.minDown = 0 species_x.maxDown = 0 species_Y.up = 0 species_Y.down = 0
     species_Y.both = 0 species_Y.minUp = 0 species_Y.maxUp = 0 species_Y.minDown = 0
    species_Y.maxDown = 0 species_y.up = 0 species_y.down = 0 species_y.both = 1 species_y.
    minUp = 0 species_y.maxUp = 0 species_y.minDown = 22 species_y.maxDown = 28 species_A.up
    = 50000000 species_A.down = 50000000 species_A.both = 0 species_A.minUp = 0 species_A.
    maxUp = 0 species_A.minDown = 0 species_A.maxDown = 0 t(0)-globalTime<=0 t(0)-species_X.
    cUp<=0 t(0)-species_X.cDown<=0 t(0)-species_x.cUp<=0 t(0)-species_x.cDown<=0 t(0)-
    species_Y.cUp<=0 t(0)-species_Y.cDown<=0 t(0)-species_y.cUp<=0 t(0)-species_y.cDown<=0 t
    (0)-species_A.cUp<=0 t(0)-species_A.cDown<=0 globalTime-species_X.cUp<=0 species_X.cUp-
    species_X.cDown<=0 species_X.cDown-species_x.cUp<=0 species_x.cUp-species_x.cDown<=0
    species_x.cDown-species_Y.cUp<=0 species_Y.cUp-species_Y.cDown<=0 species_Y.cDown-
    species_y.cUp<=0 species_y.cUp-species_y.cDown<=0 species_y.cDown-species_A.cUp<=0
    species_A.cUp-species_A.cDown<=0 species_A.cDown-t(0)<=0

Transition: species_x.begin -> species_x.level_0 {x == 0; 0; resetAll(0);}

State: species_X.begin species_x.level_0 species_Y.begin species_y.begin species_A.level_0 X
    = 0 x = 0 Y = 0 y = 0 A = 0 species_X.up = 0 species_X.down = 0 species_X.both = 1
    species_X.minUp = 0 species_X.maxUp = 0 species_X.minDown = 45 species_X.maxDown = 55
    species_x.up = 50000000 species_x.down = 50000000 species_x.both = 0 species_x.minUp = 0
    species_x.maxUp = 0 species_x.minDown = 0 species_x.maxDown = 0 species_Y.up = 0
    species_Y.down = 0 species_Y.both = 0 species_Y.minUp = 0 species_Y.maxUp = 0 species_Y.
    minDown = 0 species_Y.maxDown = 0 species_y.up = 0 species_y.down = 0 species_y.both = 1
    species_y.minUp = 0 species_y.maxUp = 0 species_y.minDown = 22 species_y.maxDown = 28
    species_A.up = 50000000 species_A.down = 50000000 species_A.both = 0 species_A.minUp = 0
    species_A.maxUp = 0 species_A.minDown = 0 species_A.maxDown = 0 t(0)-globalTime<=0 t(0)-
    species_X.cUp<=0 t(0)-species_X.cDown<=0 t(0)-species_x.cUp<=0 t(0)-species_x.cDown<=0 t
    (0)-species_Y.cUp<=0 t(0)-species_Y.cDown<=0 t(0)-species_y.cUp<=0 t(0)-species_y.cDown
    <=0 t(0)-species_A.cUp<=0 t(0)-species_A.cDown<=0 globalTime-species_X.cUp<=0 species_X.
    cUp-species_X.cDown<=0 species_X.cDown-species_x.cUp<=0 species_x.cUp-species_x.cDown<=0
    species_x.cDown-species_Y.cUp<=0 species_Y.cUp-species_Y.cDown<=0 species_Y.cDown-
    species_y.cUp<=0 species_y.cUp-species_y.cDown<=0 species_y.cDown-species_A.cUp<=0
    species_A.cUp-species_A.cDown<=0 species_A.cDown-t(0)<=0
```

Listing B.3: Part of a symbolic trace given by Verifyta. The trace is generated with the new approach (on-the-fly variant without stopwatch) and with the GRN $GM2^{*}_{10}$ of the performance case study (Section 7.5).

```
State:
( species_X.level_0 species_x.level_0 species_Y.begin species_y.level_0 species_A.level_0 )
globalTime=0 species_X.cUp=0 species_X.cDown=0 species_x.cUp=0 species_x.cDown=0 species_Y.
    cUp=0 species_Y.cDown=0 species_y.cUp=0 species_y.cDown=0 species_A.cUp=0 species_A.cDown
    =0 species_X.up=0 species_X.down=0 species_X.both=0 species_X.minUp=50000000 species_X.
    maxUp=50000000 species_X.minDown=50000000 species_X.maxDown=50000000 species_x.up=1
    species_x.down=0 species_x.both=0 species_x.minUp=45 species_x.maxUp=55 species_x.minDown
    =50000000 species_x.maxDown=50000000 species_Y.up=0 species_Y.down=0 species_Y.both=0
    species_Y.minUp=0 species_Y.maxUp=0 species_Y.minDown=0 species_Y.maxDown=0 species_y.up
    =1 species_y.down=0 species_y.both=0 species_y.minUp=31 species_y.maxUp=40 species_y.
    minDown=50000000 species_y.maxDown=50000000 species_A.up=1 species_A.down=0 species_A.
    both=0 species_A.minUp=22 species_A.maxUp=28 species_A.minDown=50000000 species_A.maxDown
    =50000000

Transitions:
  species_Y.begin->species_Y.level_0 { Y == 0, tau, resetAll(0) }

State:
( species_X.level_0 species_x.level_0 species_Y.level_0 species_y.level_0 species_A.level_0 )
globalTime=0 species_X.cUp=0 species_X.cDown=0 species_x.cUp=0 species_x.cDown=0 species_Y.
    cUp=0 species_Y.cDown=0 species_y.cUp=0 species_y.cDown=0 species_A.cUp=0 species_A.cDown
    =0 species_X.up=0 species_X.down=0 species_X.both=0 species_X.minUp=50000000 species_X.
    maxUp=50000000 species_X.minDown=50000000 species_X.maxDown=50000000 species_x.up=1
    species_x.down=0 species_x.both=0 species_x.minUp=45 species_x.maxUp=55 species_x.minDown
    =50000000 species_x.maxDown=50000000 species_Y.up=0 species_Y.down=0 species_Y.both=0
    species_Y.minUp=50000000 species_Y.maxUp=50000000 species_Y.minDown=50000000 species_Y.
    maxDown=50000000 species_y.up=1 species_y.down=0 species_y.both=0 species_y.minUp=31
    species_y.maxUp=40 species_y.minDown=50000000 species_y.maxDown=50000000 species_A.up=1
    species_A.down=0 species_A.both=0 species_A.minUp=22 species_A.maxUp=28 species_A.minDown
    =50000000 species_A.maxDown=50000000

Delay: 22
```

Listing B.4: Part of a concrete trace given by Verifyta. The trace is generated with the new approach (on-the-fly variant without stopwatch) and with the GRN $GM2^*_{10}$ of the performance case study (Section 7.5). Note here the line "*Delay ...*" which gives a concrete information about the time evolution.

Note that the generation and parsing of large traces can require a certain time.

Appendix C

Modeling approach summary

Delay calculation variants		
Pre-calculation	Principle	- For each configuration of the regulators of each gene, the delays are pre-computed in the model-builder and stored in arrays.
	Advantage	- Best precision with use of real number for the calculation.
	Drawbacks	- Delays computed for configuration which are never reached. - Generate exponential arrays.
On-the-fly calculation	Principle	- The parameters are upscaled to allow the calculation of the delays directly in Uppaal.
	Advantage	- Only the used delays are computed.
	Drawbacks	- Less precise. - Computation could slow down the model checker. - A delay can be computed several times since it is not stored.

Clock variants		
With stopwatches	Principles	- The clocks of the timed model are implemented directly in the automata with stopwatches and clock differences.
	Advantage	- The timed model is simulated perfectly.
	Drawbacks	- The stopwatches affect the decidability of the reachability problem. - Only available with an unstable development version of Uppaal.
Without stopwatches	Principles	- The clocks of the timed model are simulated without stopwatch and difference. Instead, they are reset when there is a change of direction in the evolution of the product concentration level.
	Advantage	- The decidability of the reachability problem is preserved.
	Drawback	- The product history inside a concentration level is not handled.

Table C.1: A summary of the different variants existing for the new approach.

Characteristic	**Timed-boolean approach**	**Extended IKNAT approach**	**New approach**
Model implemented	Thomas's model extended with time	Timed model	Timed model
Constant reactivity	Yes	No	No
Gene product concentration	Activity level	Concentration level	Concentration level
History in one conc./act. level	No managed	Managed	Managed with the stopwatch variant
Delay intervals	Yes	Yes	Yes
Maximal conc./act. level	Depending of the outgoing regulations for each gene	Arbitrary for each gene	Arbitrary for each gene
Non-zero basal conc./act. level	Yes	Yes	Yes
Boolean regulations	Yes	Yes	Yes
Multi-effect regulations	No (but possible by extension)	Yes	Yes
Boolean gates	Yes	Yes	Yes
SUM gates	No	Yes (default gate)	Yes (default gate)
MULT gates	No	No	No (technically possible, semantically to be adapted)
Hight level structures used in the automata	Nothing (or, for another implementation: templates, broadcast channels)	Arrays, templates, channels and broadcast channels, logical statements.	Arrays, templates, broadcast channels, logical statements, function. With the stopwatch variant : stopwatches, clock differences.
Number of processes	One (one per gene)	Two per gene, one per regulation, one per boolean gate	One per gene.
Building complexity	Exponential with the number of genes and their number of activity levels (linear)	Linear	Linear for the on-the-fly calculation variant. Exponential for the precalculation variant according to the number of regulators for a gene.

Table C.2: A summary of the different approaches according to certain characteristics.

References

[1] Jamil Ahmad, Olivier Roux, Gilles Bernot, Jean-Paul Comet, and Adrien Richard. **Analysing formal models of genetic regulatory networks with delays**. *Bioinformatics Research and Applications*, **4**(3), 2008. 2, 31, 38, 40, 104

[2] Uri Alon. *An Introduction To Systems Biology Design Principles Of Biological Circuits*. Chapman & Hall/CRC, 2007. 6, 10, 79, 105

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. h. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. **The Algorithmic Analysis of Hybrid Systems**. *Theoretical Computer Science*, **138**:3–34, 1995. 24, 25

[4] R. Alur and D.L. Dill. **A theory of timed automata.** *Theoretical Computer Science*, **126**, 1994. 21

[5] Rajeev Alur and P. Madhusudan. **Decision Problems for Timed Automata: A Survey**. In *Lecture Notes in Computer Science*, **3185**, 2004. 21

[6] Grégory Batt, Ramzi Ben Salah, and Oded Maler. **On timed models of gene networks**. In *Proceedings of the 5th international conference on Formal modeling and analysis of timed systems*, FORMATS'07, pages 38–52, Berlin, Heidelberg, 2007. Springer-Verlag. 2, 31, 40

[7] Gerd Behrmann, Alexandre David, Kim G. Larsen, Oliver Möller, Paul Pettersson, and Wang Yi. Uppaal **- Present and Future**. In *Proc. of 40*th *IEEE Conference on Decision and Control*. IEEE Computer Society Press, 2001. 19, 25

[8] B. Bérard and L. Sierra. **Comparing verification with HyTech, Kronos and Uppaal on the railroad crossing example**. 2000. 29

[9] Hamid Bolouri. *Computational Modeling of Gene Regulatory Networks*. Imperial College Press, 2008. 11

[10] W.J Bos. *Interactive Signaling Network Analysis Tool*. Master's thesis, University of Twente, August 2009. 2, 41, 61

[11] Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, and Joseph Sifakis. **Tools and Applications II: The IF Toolset**. **3185** of *Lecture Notes in Computer SCience*, pages 237–267, Bertinoro Italy, 09 2004. Springer. 25

[12] B. Bérard and Catherine Dufourd. **Timed Automata and Additive Clock Constraints**. In *Information Processing Letters*, pages 75–1, 2000. 25

[13] Franck Cassez and Kim Larsen. **The Impressive Power of Stopwatches**. In *In Proc. of CONCUR 2000: Concurrency Theory*, pages 138–152. Springer, 2000. 25

[14] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. **The tool Kronos**. In *Hybrid Systems III, Verification and Control*, pages 208–219. Springer-Verlag, 1996. 25

[15] A. Gonzalez, A. Naldi, L. Sénchez, D. Thieffry, and C. Chaouiya. **GINsim: A software suite for the qualitative modeling, simulation and analysis of regulatory networks**. *Biosystems*, **84**, 2006. 15, 39

[16] Raúl Guantes and Juan F Poyatos. **Dynamical Principles of Two-Component Genetic Oscillators**. *PLoS Comput Biol*, **2**(3):e30, 03 2006. 79, 88

[17] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-toi. **HyTech: A Model Checker for Hybrid Systems**. *Software Tools for Technology Transfer*, **1**:460–463, 1997. 25

[18] Hidde De Jong. **Modeling and simulation of genetic regulatory systems: A literature review**. *Journal of Computational Biology*, **9**:67–103, 2002. 5

[19] Heike Siebert and Alexander Bockmayr. **Temporal constraints in the logical analysis of regulatory network**. *Theoretical Computer Science*, 2008. 2, 31, 36, 40

[20] Adrian Silvescu and Vasant Honavar. **Temporal Boolean Network Models of Genetic Networks and Their Inference from Gene Expression Time Series**. *Complex Systems*, **13**:2001, 2001. 5

[21] El Houssine Snoussi. **Qualitative dynamics of piecewise-linear differential equations: a discrete mapping approach. Dynamics and Stability of Systems**. *Dynamics and Stability of Systems*, **4**, 1989. 11

[22] René Thomas. **Boolean formalization of genetic control circuits.** *Journal of Theoretical Biology*, **42**, 1973. 2, 12

[23] René Thomas and Richard D'Ari. *Biological Feedback*. CRC Press, 1990. 2, 31, 37, 40

[24] René Thomas, Denis Thieffry, and Marcelle Kaufman. **Dynamical behavious of biological regulatory network - I. Biological role of feedback loops and practical use of the concept of the loop-characteristic state**. *Bulletin of Mathematical Biology*, **57**(2), 1995. 15

Glossary

CFFL Coherent Feed Forward Loop; one of the FFL motifs which can filter the expression or the non-expression of a gene following the duration of the presence or absence of a regulator.

FFL Feed Forward Loop; a GRN motif with several variants. Following the variant, the motif can bring a resistance against regulators whose the presence or absence is only transient.

GRN Gene Regulatory Network; network of interactions between genes to achieve a given goal or a given metabolism.

IFFL Incoherent Feed Forward Loop; one of the FFL motifs which produces a brief pulse of expression for a gene.

ISN Interacting Signaling Network; network of chemical interactions in cells used to collect and convey the information about the environment of the cells or about the other cells.

ODE Ordinary Differential Equation; equation which described the evolution of a variable according to the current state of some variables. A set of ordinary differential equations can be used to describe the evolution of a GRN according to its current state.

XML A data format dedicated to the transport and the storing of information.