Master Thesis:
Interactive Signaling Network Analysis Tool

W.J. Bos
Student: 0020699
University of Twente

August 26, 2009

Supervisors:
dr. P.E. van der Vet
dr. ir. R. Langerak
ir. J. Scholma

Abstract

When researching kinase signaling pathways in cells, molecular biologists are confronted with large experimental data sets. Evaluation of these data sets, together with the fitting of these data on possible pathway models, is a highly nontrivial task. The aim of this research is to support biologists in exploring the space of networks inferred from experimental data by means of software that is both interactive and visual, thereby considerably alleviating this task. At the basis of the software lies a quantitative modeling technique that makes use of a computer science model called timed automata. Timed automata models can be created, simulated, and analyzed with UPPAAL, which is a state-of-the-art tool for analysis and design of real-time systems. Modeling a pathway with timed automata makes it possible to decide, using UPPAAL, whether experimental data fits a specific model, or whether such a model should somehow be updated. In order to hide the technical intricacies of timed automata and UPPAAL from the molecular biologist, a prototype interface tool has been built. This interface tool lets users draw a network and add experimental data to it, and then exports this information to a timed automata model to be verified by UPPAAL. Finally, the results from this verification process are translated back to the interface and presented in a graphical manner.

Contents

Chapter 1

Introduction

Introduction

The estimated number of cells in the human body is a stunning $1 \cdot 10^{14}$. [1] [Papin et al., 2005]. Each of these cells has an identical copy of DNA, containing the genes that encode all proteins a human cell could produce. The DNA is located in the cell nucleus. When a gene is expressed, DNA is transcribed into an mRNA molecule, which is transported to the ribosomes in the cell cytoplasm. Here, mRNA is translated to a protein (Figure 1.1).

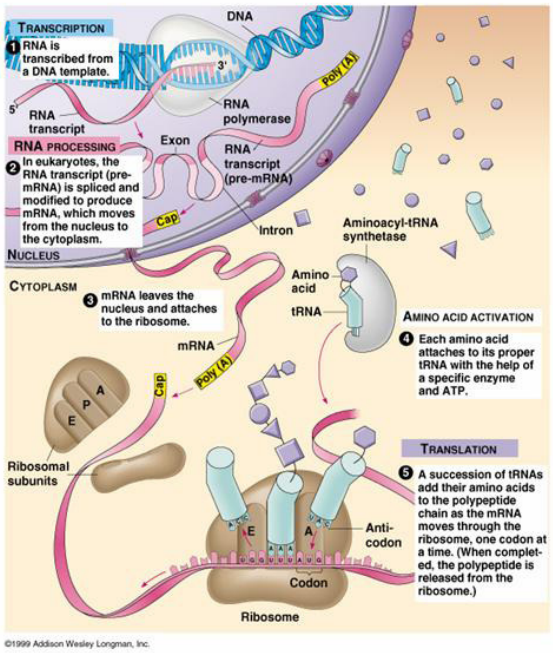


Figure 1.1: DNA transcription, picture from `http://fajerpc.magnet.fsu.edu/Education/2010/Lectures/26_DNA_Transcription.htm`

Proteins are the most versatile functional components of a cell. The three-dimensional shape and physicochemical properties of a protein together define its function. Different types of proteins can be distinguished; structural proteins provide mechanical support, enzymes facilitate

[1] All numbers in this section come from [Papin et al., 2005]

biochemical reactions, and antibodies protect the organism to infections. The functionality of a cell is defined by the specific subset of proteins that is expressed. E.g., the cells that reside in an eye contain photosensitive proteins, whereas contracting proteins are more abundant in muscle cells.

For a complex, multicellular organism such as a human to survive, detailed and extensive communication between the individual cells of the organism is necessary. In human cells, a large set of proteins is dedicated solely to this purpose. Cells are enclosed by a lipid membrane that separates cellular processes and contents from the cell environment Communication between cells is possible by secreting and receiving molecular signals (Figure 1.2). To send a signal, a cell produces signaling molecules, which are often proteins. Special transmembrane proteins function as receptors. In order to discriminate between different signaling molecules, over 1,500 different genes encode for receptors. A receptor has two functional domains, an extracellular ligand binding domain, and an intracellular domain to relay the signal. Binding of a signaling molecule to the extracellular domain, causes a conformational change and activation of the intracellular domain. Specific down-stream proteins inside the cell can subsequently be activated by the activated receptor. These proteins in turn can activate other proteins. The ultimate effect of this signal transduction process can be a change in activity of one or more of over 1,800 different transcription factors; proteins that regulate gene expression.

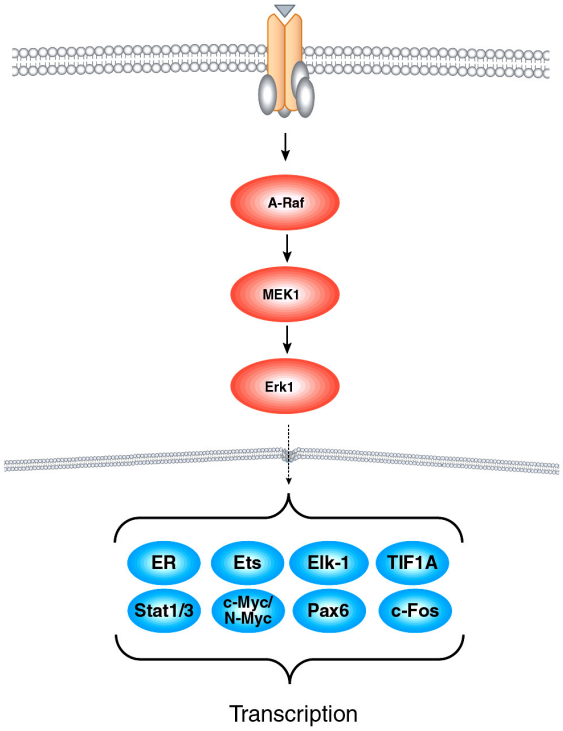


Figure 1.2: Part of a signaling pathway

In a signaling pathway, every activation step makes it possible to amplify and modulate the signal; the receptor in Figure 1.2 can activate several A-Raf proteins, and every A-Raf protein can activate several MEK1 proteins, and so on. Signaling pathways, however, are not simple signaling routes that function independent of other signaling pathways. Signaling pathways are interconnected at many points, forming a complex signal transduction network (Figure 1.3) for the processing of signals. Negative and positive feedback loops, as well as the (differences in) timing of reactions further complicate this network.

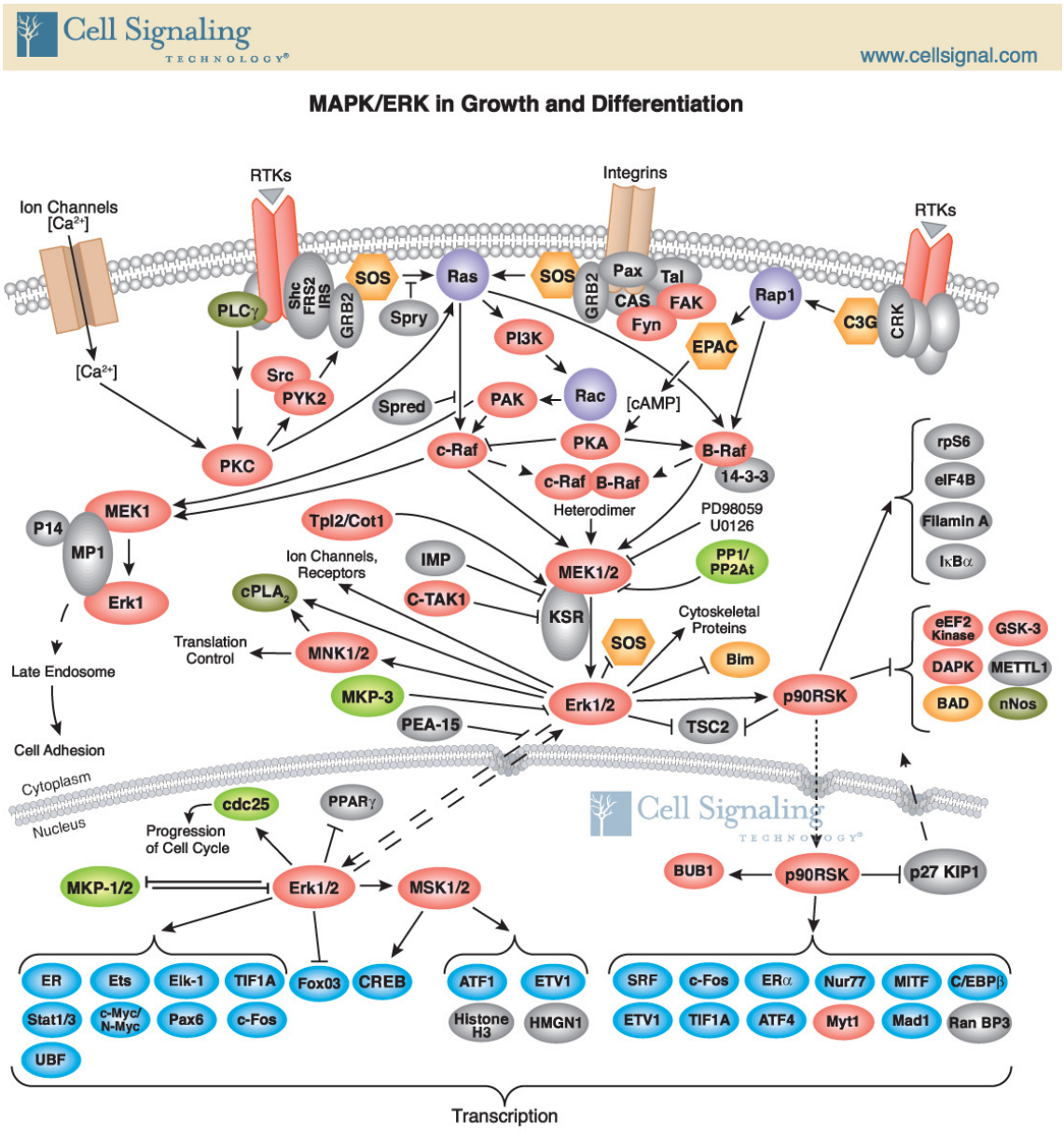


Figure 1.3: In this picture, part of a signaling network that can be found in a human cell. In this network, four types of receptors can be distinguished, each with its own signaling pathway. The cross talk is clearly visible here; Ras (pressed against the cell membrane, in the middle) for example is influenced by two receptors, and activates B-Raf, which is also influenced by yet another signaling pathway. The image is available at `http://www.cellsignal.com/pathways/map-kinase.jsp` (choose Growth Factors, Mitogens, GPCR). This is a graphical representation that is used by biologists.

Cancer is a leading cause of death worldwide: it accounted for 7.4 million deaths (around 13% of all deaths) in 2004, and the number of deaths from cancer worldwide are projected to continue rising, with an estimated 12 million deaths in 2030 [2]. Besides cancer, autoimmunity disease and diabetes can also be traced back to a malfunctioning signal processing by one or more cell

[2]Information from `http://www.who.int/mediacentre/factsheets/fs297/en/index.html`

types. Therefore, understanding how a signal is processed by a cell helps researchers to design effective treatments for these diseases. As yet, the complexity and flexibility of a signaling network prevents a thorough understanding of signal transduction processes in the cell [Marks et al., 2008]. In order to understand and work with signaling networks, researchers have to make abstractions. Modeling a signaling network is a special way of making an abstraction, with which it is possible to assess certain aspects of it. With a good model researchers can get an idea of what the effect of certain medications will be on the signal processing in a cell. In this research a prototype was developed to help researchers modeling parts of signaling networks in such a way that it is possible to simulate their behavior.

1.1 Case: Kinome profiling of human stem cells

This research is based on the Ph. D. research on the Kinome profiling of human stem cells, carried out by J. Scholma. In his research, Scholma measures the activity of multiple kinases at once with the help of a peptidemicro-array technology, the PepChip^{TM}. The results of the experiments, after postprocessing, can be seen as snapshots of the activity of a group of kinases. The research of this report originated from the wish to visually represent the activity information present in the different snapshots. Moreover, using a formal model as basis makes it possible to simulate (parts of) kinase networks. The representation in the prototype is able to give an overview of the activity of multiple kinases for different time intervals, like the information present in the snapshots" from the experiments.

1.2 Outline

In Chapter 2 the problem description and related work are discussed. In Chapter 3 the modeling of a kinase network with timed automata is explained. In Chapter 4 the building and looks of the interface for biologists are made clear. In Chapter 5 the connection of the interface to UPPAAL is discussed, and in Chapter 6, a small example is elaborated. In the last two chapters the reader will find results, concluding remarks and possible directions for future research. In the appendices technical details can be found about the used formats and translations between these formats.

Chapter 2

Problem Description

2.1 Modeling cell processes

A wealth of knowledge on the interactions in small parts of pathways is available to researchers. Based on this knowledge, biologists try to hypothesize bigger and more complete networks, with respect to the global behavior of these networks. With only the local timing of small parts known, this is a trying challenge if not nearly impossible. Abstracting from the signaling network in reality can make it possible to analyze the global behavior of a network with only the local information that is available. An abstraction from a signaling network is called a model, and the process of abstracting is called modeling.

Besides analyzing the global behavior, using a model creates the possibility to (semi-) automatically process large amounts of experimental data. Nowadays, more and more data becomes available to biologists due to the use of computers and automated experiments [King et al., 2009]. To be able to handle these amounts of data, tools that process data based on models of signaling networks can help the researcher with processing.

Based on the new data, the models of signaling networks can be refined and expanded, which leads to more complex models. Understanding the behavior of complex models cannot be reached by human reasoning alone, as positive and negative feedback loops make the behavior difficult to comprehend. With a model, it is possible to simulate the behavior of a signaling network, and by doing so predict changes in its behavior based on changes in one or more of the components of the network.

One of the reasons to choose Timed Automata and UPPAAL to model signaling networks is the possibility to automatically test if a model meets a given specification, called model checking. One of the possible applications of model checking is to verify whether a model can reach a given state at a given time, where a state describes the activity of some or all of the components in a signaling network. In the case study used for this research the results of conducted experiments consist of "snapshots" of the activity. A snapshot contains information on the activity of all the kinases at a certain time, and can there be seen as a state of the system. With model checking it is possible to verify a model of a signaling network that is researched with these snapshots. This is done by checking the reachability of the states the snapshots represent, given the initial values of the model. As kinases are important components of most signaling networks, the results from model checking are valuable for the research of signaling networks.

There are more types of questions that can be answered using model checking. More research has to be done on what type of questions researchers of signaling networks are interested in, and in what way these answers can be extracted from models of such networks. For example, questions that can be translated to a "cause and effect" question (it is always/possibly/never the case that a specific state can be reached after this state) can be answered using model checking. More about the model checking used in this research can be found in Section 3.4.

2.1.1 Related Work

Various tools for modeling and visualizing signaling networks exist to aid researchers with presenting results of experiments on these networks and with the understanding of them. A rough distinction can be made between the existing solutions; those that have a formal basis and those that do not. A formal basis enables the user to simulate and calculate on models of signaling networks. The use of formal models like Timed Automata and Continuous Time Markov Chains is only one of the many ways to learn more about signaling networks. For example ordinary and partial differential equations, stochastic equations and Bayesian networks to describe signaling networks are also used [de Jong, 2002].

In the category of tools with a formal basis fall for example Bio-Pepa, Pathway Logic and Cell Illustrator. In Bio-Pepa, Continuous Time Markov Chains (CTMC) are used as formal basis, with PRISM as model checking tool [Kwiatkowska et al., 2002; Gilmore, 2008–2009]. Their approach is motivated by *the stochastic, computational and concurrent behavior of signaling pathways*, which can be modeled using a formal model with stochastic processes, in this case CTMC. The model checker tool PRISM is a probabilistic model checker, and used for formal modeling and analysis of systems which exhibit random or probabilistic behavior [Kwiatkowska et al., 2004–2009].

In Bio-Pepa, biochemical networks are modeled in a process algebra based on PEPA (Performance Evaluation Process Algebra) [Gilmore and Hillston, 1994–2009]. PEPA is an expressive formal language for modeling distributed systems. Using models described by PEPA, a system designer can determine whether a candidate design meets both the behavioral and the temporal requirements necessary [Gilmore and Hillston, 1994]. In Bio-Pepa, the networks are modeled on molecular species level with different levels of concentrations [Ciocchetta and Hillston, 2009], as is the case in this research.

PRISM can compute the probability of a state of the system for a specific point in time, but based on the imprecise data of an experiment, this probability can be (slightly) off. As PRISM computes exact answers, and uses an exhaustive analysis (check all cases), this soon leads to a state space explosion. If a researcher wants to know if a state is reachable, and the probability of which this state is reachable can be off, it might be sufficient to ensure the reachability of this state. We believe that reachability alone already helps researchers with understanding signaling networks, and to use PRISM to compute the reachability of a state without a corresponding probability seems like overkill. As no exhaustive analysis is needed, the reachability of a state can be more easily computed with another formalism; Timed Automata. Like CTMC, Timed Automata are used for modeling distributed systems with temporal information. With Timed Automata however a state can be validated for reachability without an exhaustive analysis.

Cell Illustrator [University, 2009] (formerly Genome Object Net) uses Hybrid Functional Petri Nets with extension (HFPNe) as a formal basis. HFPNes are very expressive, but because of their expression probabilities, the advantages of using a formal model may be less prominent.

It is a software tool that is built with the end user in mind, so the interaction and visual presentation of signaling networks are good [M. et al., 2004].

There are tools for the model checking of Petri Nets, the basis of HFPNe's, but most of them convert the Petri Net to some other form before model checking it, and then translate the results back to a Petri Net. As Petri Nets become larger, they are more difficult to understand and the model becomes less robust, i.e. small errors can have huge consequences. To counter this effect, some kind of process algebra is often used to create larger Petri Nets. Translating to Petri Nets as intermediate form seems to be an unnecessary step, as they are again translated to some other form before model checking. That is why in this research, we do not use Petri Nets but Timed Automata. Like Petri Nets, networks of Timed Automata are presented visually. Multiple networks of Timed Automata can interact with each other, which allows for easy compositions of systems of networks. The model checking capabilities of Timed Automata are demonstrated with the powerful tool UPPAAL, which has a good user interface to lighten the task of modeling a system of Timed Automata networks.

Pathway Logic is a symbolic systems biology approach to the modeling and analysis of molecular and cellular processes based on rewriting logic in the Maude programming language. For analysis and visualization, the models can be transformed to Petri Nets [Talcott, 2008; International, 2006–2009; Meseguer, 1992]. We only discovered Pathway Logic very late in this research, and there was no time to study it in detail. One of the things covered in Pathway Logic and not in this research is the modeling and visualization of spatial information, which is an important aspect in cell biology. With Pathway Logic signaling pathways can be found, but it is not possible (as far as I know) to express different levels of activation for the population of a molecular species.

In the category of tools with no formal basis fall, for example, the often used tool Cytoscape. Cytoscape is an open source bioinformatics software platform for visualizing molecular interaction networks and integrating these interactions with gene expression profiles and other state data [Consortium, 2001–2008]. The core of Cytoscape is easily extended by ways of a plug-in architecture, which makes it possible for plugins like BioQuali to be accessed through the Cytoscape interface [Shannon et al., 2003; Guziolowski et al., 2009]. As far is I know there is no plugin for Cytoscape, or at least not yet, that allows model checking on a formal model, the BioQuali plugin comes closest with its directed graph representation.

Pathvisio [van Iersel et al, 2008] is a tool that is aimed at displaying different types of data, including microarray and proteomics data. It mimics the workings of GenMAPP [et al, 2002–2009], another tool for visualizing gene expression and other genomic data, and is written entirely in Java. There are numerous examples of other tools and approaches to visualize signaling networks and their data, each with their own benefits, and most of them with their own file format [van Iersel et al., 2008; Dahlquist et al., 2002; Suderman and Hallett, 2007]. To write another tool for visualizing signaling networks only adds one more player to the field, making it even harder to have all information available to all researchers of signaling networks. For the future, it might therefore be a good idea to look into the possibilities of writing the solution as proposed in this research as a plug-in for on or more existing tools. The plug-in lets the users model a signaling network with a formal basis of Timed Automata, and save the results in one of the existing formats. Problems might arise however with respect to flexible interfaces present in tools that are aimed at visualizing signaling networks on the one site, and the strictness required by a formal model on the other side.

In this research instead of a plugin for a program, a standalone prototype was developed. This allowed for a quick start, as getting to know a program well enough to write a plugin for it

takes time that was not available. Besides, developing a standalone prototype gave the freedom to completely choose are own representation. Other difficulties may also arise when writing a plugin with respect to the use of UPPAAL in this research.

2.1.2 Compositionality

New information about signaling networks becomes available on a regular basis. This information might include new interactions, new kinases or other proteins that play a part in the signal transduction in signaling networks. To be able to handle these changes, the model of a signaling network should be easy to extend. To make this possible in the prototype, a compositional way of building up a model of a signaling network can accomplish this. It would be like creating a LEGO building out of small bricks of different types. In the case of signaling networks, different type of bricks that represent kinases, proteins and reactions could be distinguished. In this research, kinases and reactions between the kinases, both activation and inhibition, can be used to build op a model of a signaling network.

2.1.3 Computational problems

In the previous chapter, a picture (Figure 1.3) could be seen that shows a signaling network. This network already is of a substantial size, but it does not show all the components and reactions known in the PDGF pathway. One of the problems with modeling large networks is the computation time for model checking such a network; it increases polynomial with respect to the size of the network. Simplification of the model can reduce this computation time, but this will also have its effect on the level of detail of the simulations and verifications. An acceptable balance between the precision, or level of detail, of the model and the information that is desired should be obtained. In this research, molecular species are modeled as a whole and not as individual molecules, which simplifies the model. The activity of these molecular species is expressed by using discrete steps, and the number of these steps can be changed. This way, the precision of the model can be altered to meet the requirements of the user.

2.1.4 Visualization

The visualization of experimental results is important, as it helps with understanding the results and discussing them. The results of kinome analysis from the case show the activity of (most of) the kinases in a human stem cell. The visualization of these snapshots should give a good overview which shows at least all the components of the network, and the activity of these components, preferably over time.

In signaling networks, temporal, spatial and causal information is important for the behavior of the network, and a good way to present this information, is the *animation of algorithms* [Priami, 2009]. In this research, the snapshots can be presented in the form of a time-lapse movie, where the colors of the components in the network change over time. This animation can give insight in the behavior, global as well as local, of a signaling network.

The modeling of signaling networks with Timed Automata is done with several parallel processes that interact with each other. In UPPAAL, all the processes that describe molecular species are shown as a separate process, and the interaction between these processes is not visually presented. The signaling network in the previous chapter (Figure 1.3) gives a good overview,

which is lacking in a model of Timed Automata representing the same data. Therefore an interface is written on top of UPPAAL which shows the molecular species as simple nodes in a graph, and the interaction between these species as arrows between the nodes. Using this representation the activity levels of a complete signaling network can be easily displayed by coloring the nodes of the graph, while the underlying model is actually a system of Timed Automata networks.

2.2 Problem Statement

Molecular biologists are increasingly confronted with large experimental data sets that support or undermine hypotheses about cellular pathways. The aim of this research is to support biologists in exploring the space of networks hypothesized from experimental data by means of interactive software. A prototype will be created that is both visual and interactive, and is based on an underlying process algebra.

Ideally, biologists should be able to load, edit, and assess existing networks, and to construct new networks themselves. The functionality options for both a tool in general and for the process algebra are also subject of research. Extensibility of the tool is an important issue.

The prototype will be examined by modeling a signaling module with it as example. This model should be able to show a similar behavior as the real world equivalent. In addition the prototype should be able to give a graphical representation of the experimental results found in the case study.

Chapter 3

Model

To be able to ask questions of a signaling network model beyond simple simulation, it is necessary to use a formal basis.

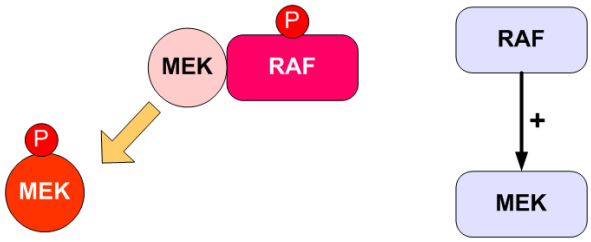
In this research, a system of Timed Automata is used as formal basis to model signaling networks. The case study focuses on measuring activity of kinases, which are important components of most signaling networks that pass on a signal, in human stem cells. The activity of kinases gives information about the signal that is processed by a signaling network. A particular example is used to explain, step-by-step, the modeling of signaling networks with Timed Automata. This example is the activation of a MEK kinase by a RAF kinase, occurring in many signaling networks. Modeling with Timed Automata is also applicable to other types of reactions in signaling networks, because of the abstraction level used. One of these types could be inactivation, the reduction of activity, also used in this research.

In reality, the activity of a population of kinases is also reduced by other inactivation processes. By kinases, the degeneration is regulated by phosphatases. These inactivation processes are modeled implicitly in this research; as a process that belongs to the population of a certain molecular species, see Section 3.3.

3.1 Abstracting from a reaction

In Figure 3.1(a) the phosphorylation of a MEK kinase by a RAF kinase can be seen. In this picture the phosphorylated form of RAF, the phosphorylated form of MEK, and the normal form of MEK are depicted. This is a typical example of a small part of a signaling network and will be used throughout this chapter. In this example, phosphorylated RAF acts as a catalysator that binds to MEK and, by doing so, enables MEK to bind a phosphate group, i.e. to get phosphorylated. This process of phosphorylation leads to activation, so in this example RAF *activates* MEK by binding a phosphate group to MEK. In Figure 3.1(b) this process is pictured in a more abstract form where the arrow with the + sign represents the activation process.

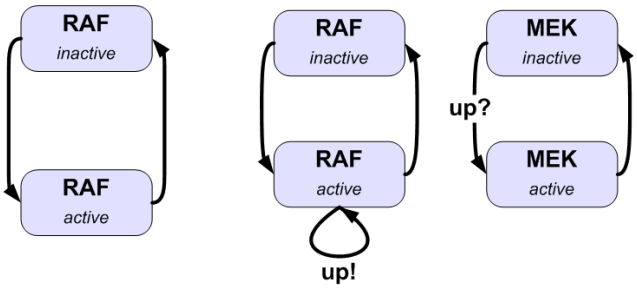
RAF is only able to activate MEK, if it is in its phosphorylated or *active* state. Therefore it is necessary to differentiate between kinases in their active and inactive state. In Figure 3.2(a) this is done for the RAF kinase. The complete process in which RAF activates MEK can now be modeled as depicted in Figure 3.2(b), where two processes interact with each other by means



(a) Biological (b) Simplified

Figure 3.1: Representation of the activation of MEK by ways of phosphorylation, catalyzed by RAF. The circles with the P inside in Figure 3.1(a) represent phosphate groups.

of synchronization on the *up* action (RAF sends a signal, and MEK acts upon this by moving from the inactive state to the active state). The synchronization of two actions makes sure these actions are always performed together and at the same time. If a process has an inactivating (or inhibiting) influence instead of an activating influence, an active kinase will move to its inactive state by means of a synchronization on a *down* signal.



(a) States (b) Synchronization on the up action

Figure 3.2: Modeling the activation of MEK with states of activity and synchronization on the up action

The synchronization described is between two components, and is also called two-way synchronization. In signaling networks however, it is possible for a protein to influence more than one other protein. To model this behavior, another way of synchronization called multi-way synchronization should be used. How this is modeled in this research will be explained later on in this chapter.

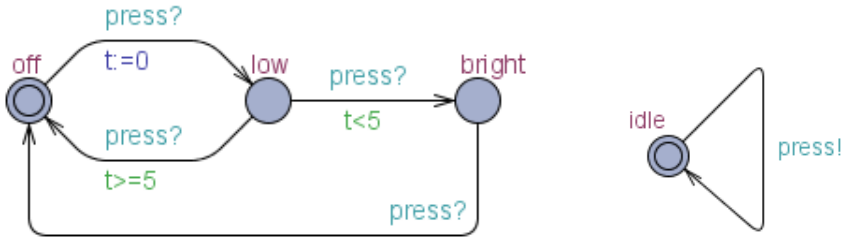
3.2 Modeling timing of reactions

Until now, the time the complete activation process takes, and how to model this, has been ignored. In Bio-Pepa a stochastic approach is used to compute the probability of an occurence of a state of a complete system, based on Continuous Time Markov Chains (CTMC) Ciocchetta and Hillston [2009]; Calder et al. [2006]. In a stochastic environment, it is common to add an exponential distribution to a transition. As models are an abstraction from reality, it is possible

to see the time representation a bit different. Instead of an exponentional distribution one can consider an interval in which an action will take place. This sounds coarser than the use of an exponential distribution, but the proposed solution is scalable with respect to the precision of the experimental results. The Timed Automata formalism uses time intervals instead of exponential distributions. For Timed Automata a state-of-the-art tool is available, called UPPAAL. Where in Bio-Pepa it is possible, by using PRISM [1], to compute the probability of a state, this is a bit different with UPPAAL. Instead of computing a probability it is possible to ask a question about the model using temporal logic expressions for queries, and the tool will return a possible trace if there is one that satisfies this query. In order to understand the power of these queries, first a basic understanding of timed automata is necessary.

In the paper by Bengtsson et. al. [Bengtsson and Yi, 2004], a Timed Automaton is described as:

> ...essentially a finite automaton (that is a graph containing a finite set of nodes or locations and a finite set of labeled edges) extended with real-valued variables. Such an automaton may be considered as an abstract model of a timed system. The variables model the logical clocks in the system. They are initialized with zero when the system is started, and then increase synchronously with the same rate. Clock constraints i.e. guards on edges are used to restrict the behavior of the automaton. A transition represented by an edge can be taken when the clocks values satisfy the guard labeled on the edge. Clocks may be reset to zero when a transition is taken.



(a) Model of the Lamp (b) Model of the User

Figure 3.3: Timed Automata model of a lamp and a user

In Figures 3.3(a) and 3.3(b) a model (or system) of Timed Automata is shown of a lamp and a user. Both these models can be seen as UPPAAL templates, as mentioned later in this section. A double circle indicates the start location of a model, in this example the lamp starts in the "off" location, and the user in the "idle" location. From the start location, the lamp can go to the "low" location via the edge, or action, in which the clock t is set to 0 ($t := 0$). Synchronization is used to link the model of the user and the model of the lamp, where the actions with the *press*! label and the *press*? label are performed at the same time. If an exclamation mark is used, as is the case in the user model (*press*!), a signal is "sent" to synchronize on. The question mark is used to indicate an action that "receives" a signal to synchronize on. The initiative of the synchronized actions lies with the action labeled with an exclamation mark. It is not possible for any of these actions to occur if their counterpart cannot be performed at the same time.

[1] `http://www.prismmodelchecker.org/`

The clock of the lamp, t, is used for the guards on the actions from the "low" location ($t < 5$ and $t \geq 5$), determining if the subsequent location is "off" or "bright". The complete system works as follows [Behrmann et al., 2004]:

> The lamp (3.3(a)) has three locations: "off", "low", and "bright". If the user presses a button, i.e., synchronises with *press*?, then the lamp is turned on. If the user presses the button again, the lamp is turned off. However, if the user is fast and rapidly presses the button twice, the lamp is turned on and becomes bright. The user (Figure 3.3(b)) can press the button randomly at any time or even not press the button at all. The clock t of the lamp is used to detect if the user was fast ($t < 5$) or slow ($t \geq 5$).

In UPPAAL, a system can be composed from models like the model of the lamp and the model of the user. These models are called *templates* in UPPAAL. To use these templates in a system, they have to be instantiated. An instance of a template is called a *process*, so an UPPAAL system consists of several processes. The composition of a system with a *lamp* and a *user* in UPPAAL is explained in Appendix B.5.

In a system of Timed Automata every network can have its own clock, and use the value of this clock as a guard on an action or as a location invariant, as was explained in the previous section. In the system that models the activation of MEK by RAF, the process of activation could for example have a duration somewhere between 5 and 10 time units ($5 \leq t \leq 10$ or $t \in [5, 10]$). This activation process can be described with the system in Figure 3.4 where the location invariant for the active state of RAF takes care of the upper bound, and the guard on the synchronization action takes care of the lower bound of this interval.

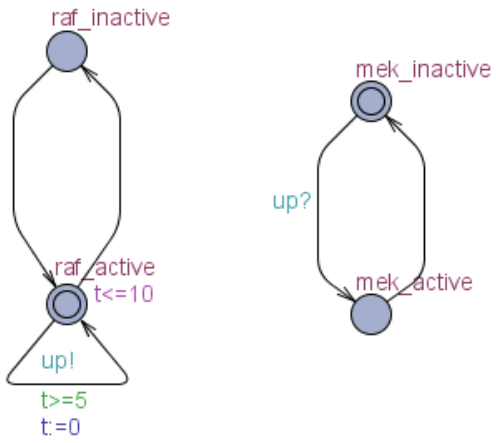


Figure 3.4: Timed automata representation of the activation of MEK through active RAF with timing constraints

As was already mentioned it is possible in signaling networks to have a single protein influence several other proteins, which calls for multi-way synchronization. In UPPAAL this can not be modeled directly, but using broadcasting signals we can achieve the same result. In UPPAAL it is not possible to put timing constraints on synchronization actions on a broadcast signal. As timing constrains are needed while modeling reactions, separate UPPAAL templates for

reactions and the implicit inactivation process of a component are used to make this possible. How the different types of synchronization are applied in this research can be found in Appendix A.1.

3.3 Modeling molecular species

The models described so far model only single kinase molecules, where each molecule can be either active or inactive. To model all kinases residing in a cell this way, the number of processes that can possibly synchronize gets enormous. This problem can be avoided by modeling on a molecular species level instead of on the individual level (molecules). On molecular species (or population) level, it is no longer possible to speak of only an active and an inactive state; some parts of the population can be active while others are not. This can be modeled using more states to express the fraction of active individuals in the population. In Figure 3.5 this is done for the RAF population in a cell. This classification of activity makes it possible to talk about the RAF population as either *inactive* (more or less all the individuals are in their inactive state), about one third active, two thirds active or maximally active (roughly all the individuals are in their active state). If modeled this way, the activity of a complete population will be changed on synchronization in discrete steps, either up or down. It is possible to change the number of states of activity of kinase by using more locations in the Timed Automata model. This way the precision of the model can be adapted to the level of detail required.

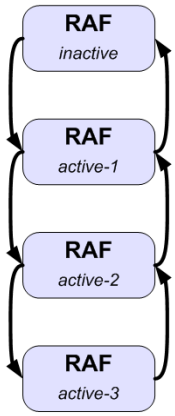
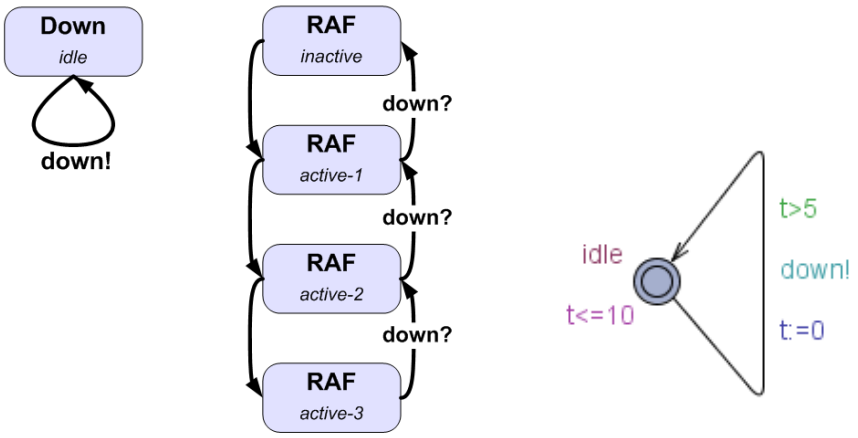


Figure 3.5: Model of a RAF population which describes the distribution of active and inactive individual RAF molecules

The degeneration is modeled as a process that belongs to a population, and synchronizes on the *down* actions of the population, or molecular species, model. In Figure 3.6(a) both the process to model a RAF population and its degeneration process are depicted. To add time to this action, the same principle as for the activation process can be used, resulting in the Timed Automata representation in Figure 3.6(b).

The problem of modeling the timing of the degeneration in this way, is the influence of the concentration of phosphorylated or active individuals on the time it takes for the population to change its state; The higher the concentration of active individuals, the faster the degeneration process will be. To model this effect, the degeneration process can consist of the same number



(a) Degeneration and Population process

(b) Timed Automata of the Degeneration

Figure 3.6: In Figure 3.6(a) a model of a RAF population with its corresponding degeneration process is shown. The degeneration process sends a *down* signal on which the population process synchronizes. The Timed Automata representation of the degeneration process in Figure 3.6(b) sends a signal in the interval $(5, 10]$.

of states [2] as the population process and mimic the state of this population process. Based on the state it is in, a specific interval in which the *down* signal is passed on is used.

The time it takes for a kinase to find and activate (or inactivate for that matters) another kinase also depends on the number of active individuals in the population. Therefore the same idea used for the degeneration process can be used to model the activation process. How both the degeneration and activation are modeled exactly can be found in Appendix B.1, which describes the timed automata models used in detail together with an example of UPPAAL code to compose a simple system of a part of a kinase network.

3.4 Model checking

One of the reasons for choosing Timed Automata and UPPAAL, is the possibility for model checking. Model checking is the problem of automatically testing if a model meets a given property. In order to solve this problem, both the model and the specification have to be formulated in a mathematical language. In this research, the model is formulated with Timed Automata. The specifications or *queries* used in UPPAAL, are formulated using a subset of the Computation Tree Logic language [Behrmann et al., 2004].

To test a model, there are different types of queries available, which can be divided into *reachability*, *safety* and *liveness* queries. In this research, the only queries used are of the reachability type. A reachability query asks whether it is possible to reach a specified state or not. To learn more about the queries, the reader is referred to [Behrmann et al., 2004]. To understand how a reachability query is evaluated, it is necessary to understand how this evaluation works: An

[2] To be more precise, this is the same number of states minus 1; there is no degeneration if the population is completely inactive.

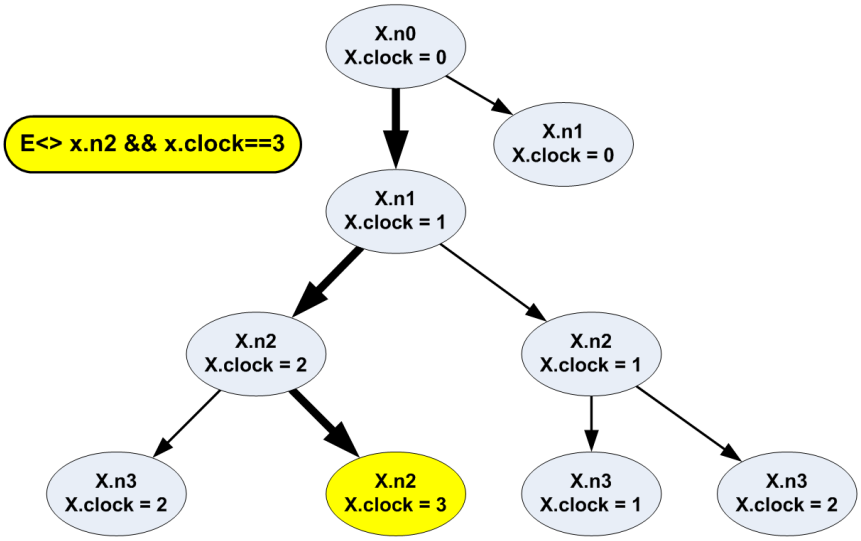


Figure 3.7: Very simple model with a query that is satisfied in the colored state. The path to this state is marked by bold arrows.

UPPAAL model has different states, where a state in UPPAAL consists of the locations of all the processes and the values of all the variables and clocks. The initial state of the system is provided by the user, and all the other states of an UPPAAL model are found via actions in the model. A reachability query consists of a description of (some of the) the features of a state. The features are given as logical statements and describe locations of the processes, clocks or variables. The identifier for a reachability query in UPPAAL is $E\Diamond$, which can be read as *there exists a state for which the following logical statements are met*. The response to a reachability query is either "no", there are no states that satisfy this query, or "yes" there is at least one such state. An example of a reachability query could be:

$$E\Diamond\ x.n2\ \&\&\ x.t\ ==3 \tag{3.1}$$

If UPPAAL now is asked if the model satisfies this query, it checks automatically if there exists at least one state in which process x is in location $n2$, and the value of the clock t of x is exactly 3. If there is at least one such state, next to the "yes" answer, UPPAAL can give a trace to this state as response. A trace consists of consecutive states, ending with a state satisfying the requirements. In Figure 3.7, the states of a simple system are displayed. This system consists of only one process, called x, which has 4 states ($n0$ .. $n3$) and a clock (t). There is only one state that satisfies the query (yellow), and the path to this state is marked by bold arrows. The trace that UPPAAL would return consists of the states along this path.

More about reachability queries as used in the prototype, can be found in Appendix B.7. For now, no safety and liveness queries are used in the prototype, but this is of course possible. More research has to be done on the desired information on a model by biologists, and how this information can be distilled from the model using either reachability, safety or liveness queries. To learn more about these queries, the user is referred to [Behrmann et al., 2004].

3.5 Summary

In this chapter, the modeling of signaling networks with Timed Automata has been explained. The components (kinases or other proteins) of a signaling network are modeled on a molecular species level. Every molecular species is modeled using different levels of activity. In this research, activation and inactivation of such a molecular species by another molecular species and degeneration are modeled. The degeneration of a molecular species is modeled as an abstract process. To model a system, three different templates are used, one for reactions, one for degeneration and one for molecular species. A model in UPPAAL, or a system of Timed Automata networks, consists of several of these templates that can interact with one another based on synchronization actions. In Figure 3.8, a simple model of a signaling network is depicted, showing two molecular species (RAF and MEK), one reaction process, and two degeneration processes.

Every reaction (activation, inactivation and degeneration) in a signaling network takes time. This is modeled using intervals that are based on the levels of activity of the molecular species involved in the reaction. The timing of the reactions is regulated by the degeneration and reaction processes, which all have their own, local, clocks; the molecular species process only changes its activity level if told so, and then broadcasts this change to all the reactions that it is involved with.

Modeling with Timed Automata makes it possible to verify certain aspects of a signaling network, if modeled correctly. One form of verification, or model checking, is checking the reachability of a state of the model given initial values. The proof of this verification can be a trace, which consists of the successive states that lead to the state described in the verification question.

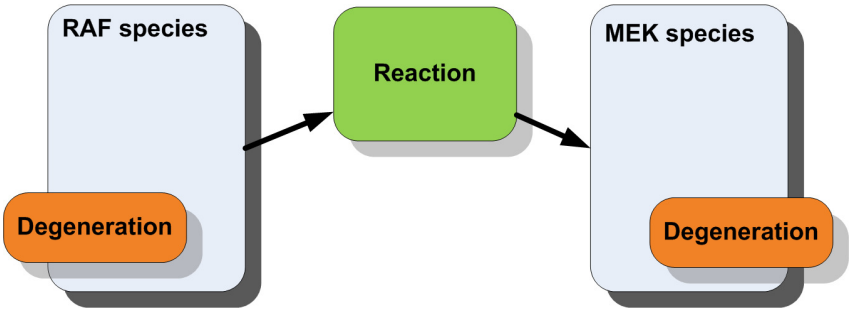


Figure 3.8: An abstract representation of the composition of templates into an UPPAAL system. The communcation between molecular species processes is done by a reaction process, and every molecular species process has its own degeneration process.

Chapter 4

Interface

In the previous chapter the modeling of signaling networks using timed automata has been described. Even with the use of templates, it is not a simple job to model this kind of networks. To support users with modeling signaling networks, a prototype with a user interface is created. The intended use of the prototype is to let biologists easily create a model of a signaling network, and either experiment with the parameters or add experimental data to it. Experimenting with the model gives a biologist more understanding of the network and how it will react to certain changes in either the network construction or the timing of one or more molecular species.

In order for the prototype to be successful, it has to give a good overview of the signaling network modeled. This overview can be created by a user with an interface in which it is possible to drag around and resize the network components. As new information on signaling networks becomes available frequently, a user should be able to change the parameters of the modeled signaling network. This information can include new components and reactions. Therefore it should also be possible to modify the structure of a network by adding, removing and updating components.

Typical actions a user may want to perform on a modeled signaling network include the loading and saving of data, automatic completion of parameters, resizing and reordering components, and updating the properties of components. For this research a user should also be able to export the model to the UPPAAL format, query this format, and to get back a trace for this query. To interpret this trace, it should be possible to visualize it with for example Microsoft Excel, using charts, or by returning snapshots of the trace, or even a time lapse movie of these snapshots.

4.1 Visualizing a network

A signaling network can be viewed as a set of molecular species that interact with each other via reactions. When biologists draw a pathway, they normally draw a graph in which the nodes stand for the molecular species involved, and the edges for the reactions between those molecular species (see for example Figure 1.3, [Marks et al., 2008] or [Alberts et al., 2003]). This was why it seemed natural to use a graph structure in the interface. The Computer Science department at the University of Twente has standardised on Java [Sun Microsystems, 1994–2009], therefore this programming language is used to develop the prototype. An excellent free Java library to visually represent graph structures was also used in the prototype, called JGraph [JGraph,

2001–2009]. The manual did provide an enormous amount of information, but there was not really a "getting started" section. Making JGraph work like you want it to work therefore took more time then was anticipated. The resulting graph structures in the prototype however are very flexible; all components can be dragged around and resized. The difference between model and view in JGraph also made it easy to have multiple tabs (explained later) with the same model structure, and with only the colors of the components changed.

Two types of data can be distinguished in a model: data that is time-independent, and data that is used to define a state of the system. This latter can be compared to the snapshots of the experimental results of the case study, and therefore consists of a specific point in time (timestamp) and the levels of activity for the components in the network. The data independent of time consists of the structure of the model, the number of activity levels, and the timings of both reactions and implicit inactivation. The number of activity levels is a global value that is used for (displaying) the activity level of a molecular species and the right timing of both reactions and implicit inactivation. In the prototype it is possible to add multiple tabs that display the same structure of the model (multiple views for one structure), with different arrangements of the activity levels of the molecular species components for a given timestamp. To keep the overview, the activity of the components is expressed as a color ranging from green (completely active) via yellow to red (completely inactive). In the prototype a tab with this data is called a "timeslice", which can be added by pressing the "New Timeslice" button. In the prototype, the first tab is used to structure the model and to add the time-independent parameters. The other tabs are used for expressing the states of this model. That the first tab also displays the state of the model for $t = 0$ obfuscates the distinction between the time-independent data and data corresponding to the states; this obfuscation grew into the prototype without much thinking and is something that should be attended to in a later version.

For now, only one global value for representing the coarseness of the complete model with respect to the activity of the components is used. In later iterations of this prototype it might be the case that different levels of detail for different species with respect to the activity are better suited for modeling a signaling network. This might reduce the state space of the underlying model on the one hand, and give more freedom of expression to the user on the other hand.

In this version of the prototype, the components all look the same, though they can be resized. As there are multiple types of components in a signaling network that all have their own representation in biological pictures (see Figure 1.3 for example), this is an interesting point of research for future versions. The modeling of a bigger example than was used in this research (see Chapter 6) will help with discovering the weaknesses of the prototype, and by helping to design a more flexible prototype that is better suited for users from the field of biology.

In Figure 4.1 a screenshot of the prototype is shown. With this prototype it is possible to open and save models, with all the accompanying timeslices. The open and save functions are accessed via the dropdown menu. Also in the dropdown menu is a function to export the graphical model from the interface to an equivalent of this model in the UPPAAL format. This is necessary to be able to query this model in order to get (simulation) traces, which can be generated by using the generate trace function. This will be explained in the following chapter. The global value that expresses the amount of activity levels used throughout the model is also changed via the dropdown menu, as well as the option to save all the tabs of the interface to image files.

If a user wants to create a model, he or she will start with pressing the button to add a vertex representing a molecular species to the drawing area. To connect two species, the user clicks

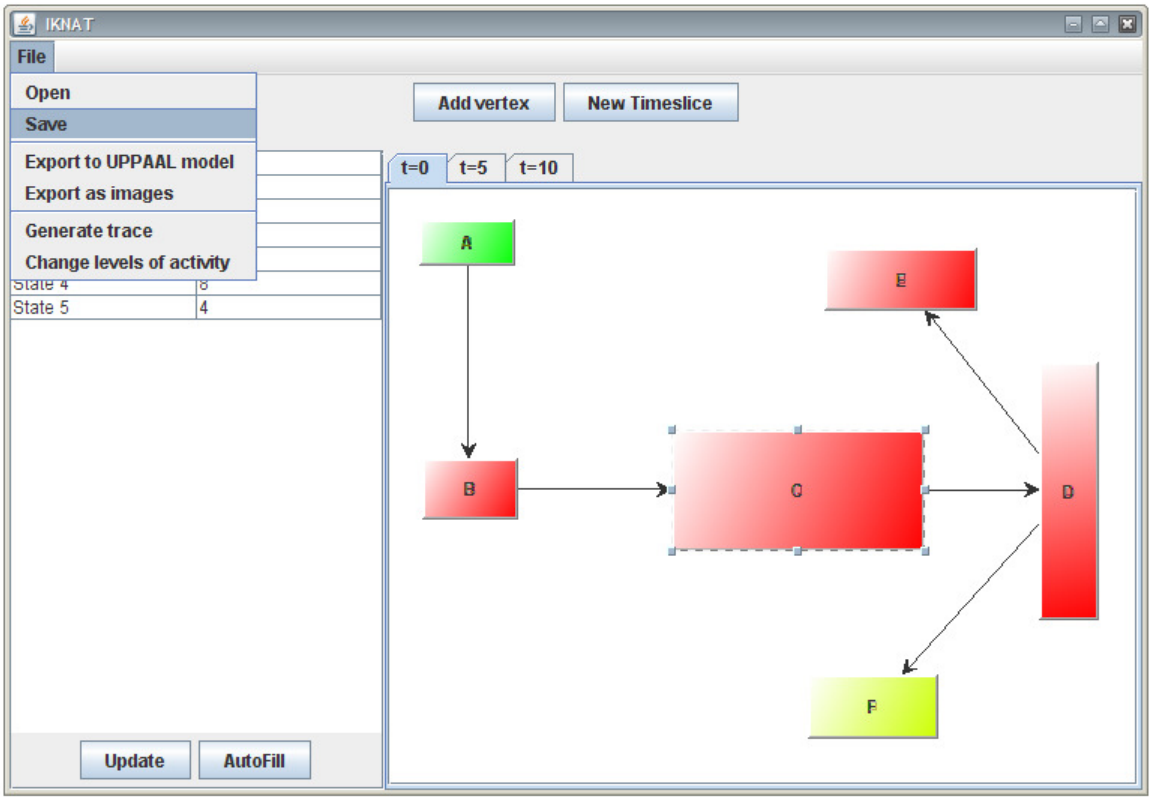


Figure 4.1: Screenshot of the prototype

the mouse in the center of the first (upstream) molecular species, and then drags a connecting line to the second (downstream) molecular species. After selecting either a vertex or an edge, the user will be presented with a small table for adding the appropriate data. To help the user with filling in the data for a model with a lot of activity levels, there is an autofill option which, given the minimum and maximum values, generates the values for all the activity levels in a linear manner. In a following version of the prototype this autofill function can be enhanced by letting the user choose from a series of mathematical formulae used in biological equations. For the prototype to act on the changes in the values, the user first has to press the update button. With the "New Timeslice" button, as was mentioned before, the user creates a new tab for a specific point in time with a (graphical) copy of the model. The user now can change (only) the activity levels of the molecular species for this timestamp.

4.2 Adding data to the model

The total number of activity levels has been entered by the user as a global parameter, and the user now has to add data to the reactions and molecular species in the model for the different levels of activity. Which data is added, and how this is connected to the reality will be explained in this section

4.2.1 Molecular species

The data needed for the molecular species consists of a name, and the parameters expressing the implicit inactivation and the current activity. The name should be unique throughout the whole graphical model, ensuring the success of translating the model to a system of Timed Automata. The activity is given as the activity level the molecular species is in. For every timestamp a molecular species can have another level of activity. In Figure 4.2 a screenshot for adding the name and a level of activity to the molecular species for $t = 10$ is shown, with a small description for the parts of the interface.

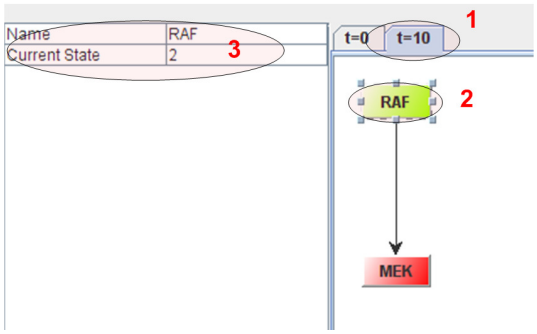


Figure 4.2: The adding of data for a molecular species for timestamp t=10. (1) The timestamp tab. (2) The selected species, the color ranges between red and green via yellowm where red (state 0) is completely inactive and green is completely active. (3) The data for this timestamp, only the activity level is changeable.

The implicit inactivation of a molecular species is modeled with a process that is not visualized in the graphical model. The data needed for the implicit inactivation consists of the timing of this reaction and the molecular species it applies to. The timing depends on the activity level of the molecular species, and describes the interval the degeneration "waits" before bringing the molecular species down one state [1]. In Figure 4.3 the adding of the timing data for the implicit inactivation of RAF is shown. This implicit inactivation is added to the model in the first tab, where the signaling network is modeled with its time-independent parameters.

4.2.2 Reaction

The data needed for the reactions consists of a name, an upstream component, a downstream component, and the timing of the reaction and the type of reaction (downstream effect). The name does not have to be unique, as it is only used in the interface. In the system of Timed Automata, a reaction is defined by the two components it connects. The timing of a reaction depends, as was the case with the implicit inactivation, on the activity level of the upstream component and describes the duration of the reaction, i.e. the interval the reaction "waits" with passing on the signal to the downstream component. The type of reaction is either activation or inactivation (inhibition). In Figure 4.4 the adding of reaction specific data is shown. As the reaction is modeled as a time-independent entity, the only place where data has to be added to it is in the first tab.

[1] If the molecular species is in its most inactive state (state 0), there is no further degeneration possible. If the molecular species now receives a signal to go to a lower state, it stays in this state.

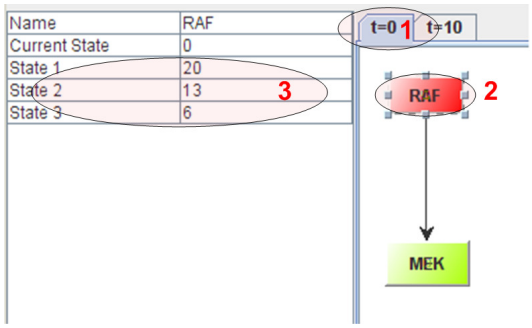


Figure 4.3: The adding of data for the implicit inactivation of a molecular species. (1) The timestamp tab, tab t=0 is the only tab on which it is possible to add degeneration and reaction times to the model. (2) The selected molecular species for which the degeneration holds. (3) Timing information of the degeneration, for every state of the molecular species except the inactive state 0.

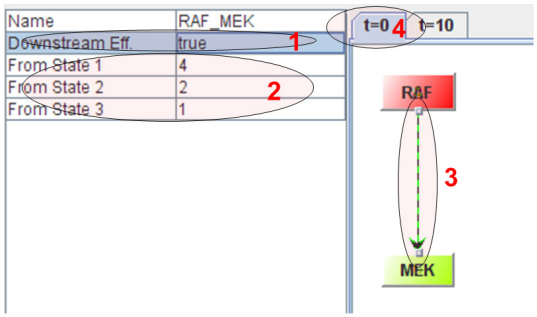


Figure 4.4: The adding of data for a reaction. (1) The downstream effect, *true* stands for activation and false for inhibition. (2) The timing information; the global value for number of states is 4 (including the inactive state 0). (3) The selected reaction. (4) Timestamp tab, this tab also contains the time-independent data for the components of the network.

4.2.3 Add tabs with specific points in time

After adding a new timeslice to a model, the user can add data to the components of the model for the specific point in time the timeslice represents. In Figure 4.2 the user is working in the timeslice for $t = 10$. After selecting a molecular species, as in this picture, the table for adding data (left) only lets the user change the value of the activity level. For the reactions between the molecular species, no state has to be added. As for now it is not possible to delete a timeslice of the model, simply because I forgot to add this functionality. In a following version, this should be added.

4.3 Representing trace data in the prototype

The resulting trace data returned from UPPAAL consists of snapshots, which can be translated back to timeslices in the prototype. When a large trace is returned, consisting of at least several hundreds of snapshots, the program will exit due to insufficient memory (traces containing more

then 500 states will dramatically effect the working of the prototype, or even result in a crash of the program) . Smaller traces are loaded directly into the prototype, and all timeslices can be saved as image files. It is possible to make an animation of all these snapshots; this is however not yet a build-in component of the prototype. As the generated trace, before it is loaded into the prototype, is saved as a system it might be a good solution to write a method to save the states as image files and as a movie directly, instead of first loading the system into the prototype. Next to this information, the data is also saved as a comma-separated value file which can be used by for example Microsoft Excel to create charts where the activity levels are measured against the time (see also Appendix E).

4.4 Conclusions on the interface

The prototype was build to show the potential of modeling a signaling network with Timed Automata, but with a separate graphical representation. The prototype only has a basic representation of components in signaling networks, and for example spatial information is not even thought of while programming it. The basic functions like adding and deleting molecular species and reactions, saving and opening a model and the graphical modeling are available to the user. A lot of things however need to be expanded, like a better and more complete autofill option, automatic query generation (see also the following chapter for more about the queries), and the representation of a resulting trace. This prototype merely shows the possibilities of an interface to graphically model a signaling network, which will be translated to a system of Timed Automata networks which can be assessed by the user. The results can be translated back to the graphical model in such a way that they can easily be shared with other researchers in the form of a movie. For this prototype to be useful to biologists, a lot of research has to be done in both the modeling of signaling networks with Timed Automata, and the graphical representation that lets users create these models intuitively.

Chapter 5

Combining the prototype and UPPAAL

In the previous chapters, both the modeling of signaling networks with Timed Automata and the graphical user interface to do this were discussed. To combine the two results, a prototype was built which uses UPPAAL for the verification of the resulting models. This prototype enables users to graphically build a model, and add (experimental) data to it. If the user is satisfied with the added data, the prototype compiles a system of Timed Automata in the UPPAAL format. Now UPPAAL is used to generate a (simulation) trace of this model, given a verification question, or query. The resulting trace can be translated back to the graphical interface, and can then be saved as an animation or as separate image files. In the prototype a trace is displayed using tabs for representing the states in the trace.

With UPPAAL comes a command line tool called verifyta, which can check a model of Timed Automata generated by the prototype. As input it needs both this model, and a query file (see Appendix B.7 for information on the queries), and as output it can generate a trace that satisfies this query. If verifyta cannot find a trace that satisfies the query, no trace is returned, which will result in an empty trace delivered to the prototype. The search algorithm used within verifyta to verify a query is a random depth-first search. Other types of searches are also possible with verifyta, but are not used in this research.

5.1 Work-flow

First a user has to make a model and add the time independent parameters to it, see also the previous chapter. If this is done, there are two possibilities to continue: add timeslices to the model, or generate a simulation trace.

In order to generate the simulation trace, the model first has to be exported to the UPPAAL file format (see Section 5.2), see Appendix C.3 for more information on this format. Then, the user has to make a query file, which for now is done manually by making a .q file with a text editor with in it the query, see Appendix B.7 for more information on queries. The file has to reside in the same directory as the UPPAAL (.xml) file, and has to have the same name (test.xml and test.q for example).

Now verifyta is used to generate a trace that satisfies the query. See Section 5.3 and Appendix

A.2 for more information on how this is actually done. The generation of the trace will take some time, depending on the number of activity levels, the number of components and the query used. The result from the trace will be a comma separated value file [1], which is build up as follows: The first column represents the value of the time, or the specific point in time, for which the state holds. All the columns thereafter contain the activity levels of the molecular species components in the model (E). This can be used to generate a scatter chart, using for example Microsofts Excel or OpenOffices calc, displaying the activity levels of several components over time. It is also possible to load the states into the prototype, but for large traces (over 500 states), this method will become unstable as all the tabs in the prototype use up memory. The intermediate result, before it is loaded into the prototype, is saved as a (temporary) .ikn file (see Appendix C.2 in the working directory. In the future, a method to generate separate image files for all the states in this file has to be added to the prototype. Sadly there was not enough time to implement this during this research.

With all the image files, it is possible to generate an animation. The code to do this was already developed, but not yet included in the prototype. The resulting trace contains gaps, i.e. it does not have a state for every specific point in time (see Section 5.3 as to why). To be able to get a good animation, these gaps have to be filled. This can be done using the fillGifs java class. After this, an animated gif can be generated of the image files using the writeAnimatedGif java class; given a directory containing gif files (format is based on a timestamp, an underscore and a suffix, for example 1251119790727_9.gif), and a frame rate (in 1/100s of a second).

Instead of simulation, the user can also decide to add more timeslices to the model. This way a user can visualize experimental data. The initial idea was that the user was able to automatically generate a reachability query given the time independent parameters of the model, and the timeslices as points in the trace that should be reached. Due to time limitations, this was not possible to add to the prototype in this version. Still, it seems to be a good feature, as it makes it possible to verify if a model can produce specific states at certain moments that are measured in experiments. The results from experiments however are not applicable (yet) to use in this prototype, and therefore this feature did not have priority.

5.2 From graphical model to UPPAAL

This section will describe how the model in the prototype is exported (or translated) tot system of Timed Automata that can be loaded into UPPAAL. This file format is described in Appendix C.3. To understand this section, the reader first has to have an understanding on how an UPPAAL system is build up (see Appendix B for more information). Together with the global value representing the number of activity levels used throughout the model, the data in the first tab of the prototype will be used. This describes the structure (which components, and how connected) and the time-independent data (reaction and implicit inactivation timings).

Three templates will be generated for every model that is exported; one for molecular species, one for reactions between molecular species, and one for the implicit inactivation of molecular species. These templates always look similar, but are based on the amount of activity levels used throughout the model. One of the advantages of using a fixed value for the activity levels of all components is that only three templates are needed for building a signaling network model. In Appendix B.1 the templates that are used for a model with four levels of activity are depicted.

[1] `http://en.wikipedia.org/wiki/Comma-separated_values`

In the graphical model, nodes and edges can be distinguished. A node will result in a molecular species template and an implicit inactivation template in the exported system. An edge will become a reaction template. The resulting system is build up from several instantiated templates in parallel, which communicate with each other using "channels". Before instantiating the templates with the parameters that belong to the molecular species (after which they become processes), these channels have to be declared. The following channels are used:

- The *up* and *down* channels that are used for the regulation of the activity level of the molecular species.
- Broadcasting channels along which the molecular species process can tell the other processes (implicit inactivation and reaction) what its new activity level is.
- Channels used by the reaction processes to find out in what state a molecular species process is.

The technical details of how this is done exactly are omitted in this report. The actual translation is done in the SaveToUppaal method in the SaveResults class. After the templates are generated, this method loops over the nodes in the model and then over the edges. During this process, the code for the channels and instantiation of the templates is collected. At the end, the method makes sure that the code snippets are placed correctly and the file will be saved to the UPPAAL format.

5.3 From trace to prototype

When a user chooses the "Generate trace" function from the dropdown menu, he or she has to select an UPPAAL file for which a trace has to be generated. For this purpose two small command line tools are used in this project; verifyta and tracer. Verifyta is the command line tool for verifying properties of UPPAAL models, and is included in the UPPAAL distribution [2]. Tracer is a command line tool not yet included in the UPPAAL distribution, which can generate a symbolic trace from an UPPAAL specific trace [3]. In this research the Windows variants of these two tools are used.

In the generation of a trace, several steps can be distinguished. More information on these steps can be found in Appendix A.2. The first step is the generation of a trace in the UPPAAL trace format. After this, a second output is generated that together with the trace from the first step are used by the tracer tool to create a symbolic trace file C.1. This result is not saved as intermediate result, but directly parsed by the prototype.

In the prototype, the states presented visualize the activity levels of the modeled network. In the tracefile every change in the UPPAAL model is present. A lot of states however only reflect changes that deal with the communication between the parallel processes. The parsing of the file makes sure that no unnecessarily large symbolic trace files have to be saved. More information on the parsing of the symbolic trace file can be found in Appendix A.2.

[2] `http://www.it.uu.se/research/group/darts/uppaal/download.shtml`

[3] After contacting Alexandre David, one of the developers of UPPAAL, he sent me the following two links for downloading the tracer: `http://www.cs.aau.dk/~adavid/tracer.exe` (win) or `http://www.cs.aau.dk/~adavid/tracer`(linux)

The result of the parsing is a set of states in which the levels of activity are described for the molecular species components. As an intermediate result, these states are written to a file, together with the model in the prototype (only the structure of the model). The file format used is the IKNAT file format (Appendix C.2), representing a complete system. The states from this result can be loaded into the prototype, but this is switched off by default. (This can be switched on in the actionPerformed method in IKNATPrototype.java, in the parseTrace action.) As was mentioned earlier in this chapter, the states of the trace are also saved as a comma separated value file. In the following chapter, a chart generated by Microsofts Excel of such a file can be viewed (Figure 6.3).

5.4 Conclusions

This section showed that it is possible to write an interface on top of UPPAAL, or more precise to generate an UPPAAL model which can be verified and interpreted using verifyta and tracer. The initial idea was to load the states of the trace in the prototype as tabs, but with larger traces (>500 states) this becomes troublesome. Therefore a feature has to be added to the prototype to generate image files from the intermediate result in the IKNAT format. The comma separated value files proved to be a good addition with which it is possible to easily assess a model, as was the case while creating the example in the following chapter. As I am not an UPPAAL expert, I do not know if better ways exist to generate and parse (simulation) traces for this prototype. This is something that could be looked into by someone with more knowledge on UPPAAL.

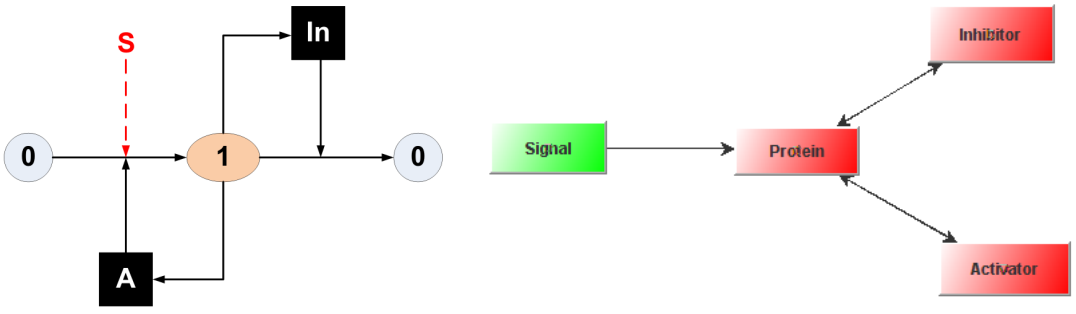
Chapter 6

Modeling an Oscillator

The understanding of signaling modules, or small functional parts of bigger signaling network, contributes to the understanding of complex signaling networks. The prototype is suited to model a signaling module rather easily, and then vary the parameters to observe the changes in which these parameter variations result.

To demonstrate the functionality of the prototype, an (undampened) oscillator is modeled in this chapter. The behavior of this type of signaling module is difficult to understand and predict intuitively, despite the small number of proteins involved. Oscillating signals carry a message that depends on the frequency of the output signal. Although few oscillating biological signaling processes have been described so far, *frequency-dependent modulation of endogenous signals is expected to be the rule rather than the exception, and like the brain, a cell may be understood as an ever-oscillating system* [Marks et al., 2008].

A biological example of an oscillator is the oscillating cAMP signal, which induces cell aggregation and differentiation of the slime mold Dictyostelium discodeideum. cAMP stimulates its own production (positive feedback) and at higher concentrations it inactivates its receptor and simultaneously promotes its own degradation [Marks et al., 2008].



(a) Biological representation (b) IKNAT representation

Figure 6.1: In 6.1(a) the biological representation as can be found in [Marks et al., 2008] is showed, and in 6.1(b) this same model in the representation used in the prototype can be seen.

The modeled oscillator only uses a single negative feedback. It consists of a protein of which the activity is measured, an inhibitor, an activator and a signal. The activity of the measured

protein is considered to be the output of the oscillator. Figure 6.1(a) shows these components and how they are linked together. In this picture, the measured protein is shown as three separate states, where the signal (S) and activator (A) both influence the activation of the measured protein; the protein goes from state 0 to state 1. The inhibitor (In) inactivates the protein; the protein goes from state 1 to state 0. In Figure 6.1(b) the model as used in the prototype is depicted, where the signal is added as a component.

The model contains fifteen levels of activity for all the components. To simulate a continuous signal, the degeneration of the Signal components highest level of activity is very high (10,000, see also D.1 in Appendix D). From this state the Signal activates the Protein. To make sure the degeneration of the Protein itself does not influence the results, the degeneration of the Protein is set to very high, like the Signal.

It sounds reasonable that one component in the oscillating system could account for a modulation of the output frequency, and we speculated the inhibitor would be a suitable candidate. The inactivation of the inhibitor (modeled implicitly in this example) can be influenced by other parts of the signaling network in a cell, and the difference in the timing could account for a change in the frequency of the oscillating signal.

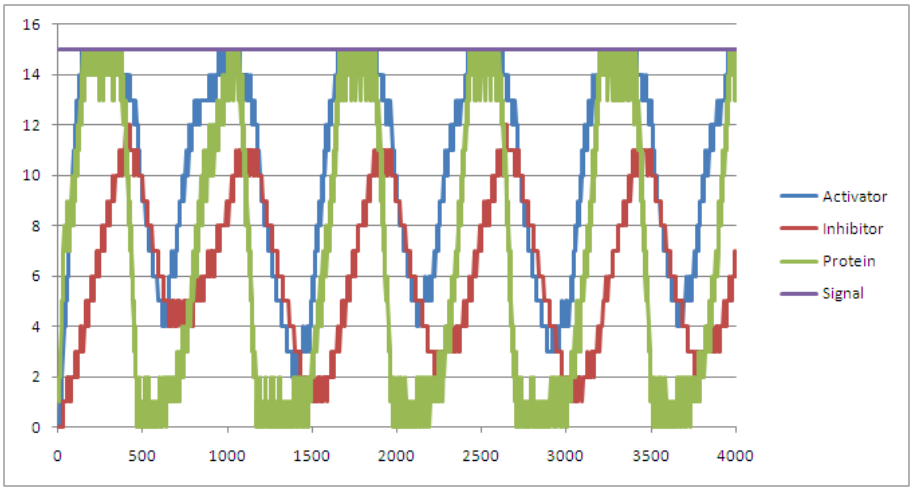


Figure 6.2: Resulting chart for the components in the signaling network, on the first run. See Tables D.1 and D.2 for the used values for reactions and degenerations.

In Figures 6.2 and 6.3 the activity levels of the components in the model can be seen. The activity signal of the Protein component in the charts represents the frequency of the oscillator. The charts show that this frequency is different between for the two setups, which can be traced back to a difference in the inactivation of the Inhibitor. This was exactly what we thought was possible with the oscillator model, but did not know beforehand how exactly this change would influence the frequency. While trying different parameters, we gradually came to understand the workings of an oscillator. Playing around with other signaling modules, and trying to simulate the observed behavior, can help researchers with understanding how the signaling modules work, and how they can be influenced.

The values used in the model are only a possible set of parameters. By no means have we pretended this to be the only parameters that could lead to an oscillating system with frequency modulation. The reader is encouraged to try out this model, and use different parameters with which it might be possible to generate a similar output of the system.

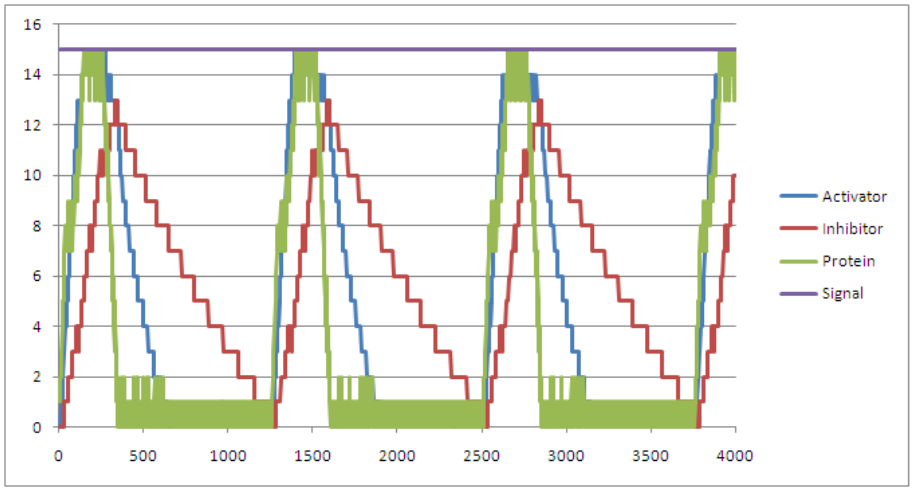


Figure 6.3: Resulting chart for the components in the signaling network, on the second run. See Tables D.1 and D.2 in Appendix D for the exact values used as parameters for the reactions and degenerations.

For realistic models of signaling processes, detailed experimental data of various biological processes has to be available. At this moment the availability of data is still a problem. For this reason, the data used in this model are fictitious. The prototype is used in one of the two ways it was designed for: playing with the parameters of a model to see what happens. The parameters used can be found in Appendix D.

The prototype also makes it possible to export the states of a trace to image files. With a little standalone Java class, these image files can be added together as a time lapse movie. This will further help understanding and visualizing the activity of the components in a signaling network (module) [Priami, 2009]. In Figure 6.4 this animation is visible, or at least in the digital format of this report. (It might be possible that you have to click twice for the animation to start.)

(Press to load movie.swf)

Figure 6.4: This is the resulting movie for the first 1,000 time units. This animation should express the same information as can be seen in 6.2

Chapter 7

Results

The goal of this research was to build a working prototype with which biologists can graphically model signaling networks with Timed Automata. This was done by modeling populations of molecular species, with levels of activity. The activity is regulated by reactions that activate or deactivate, and the degeneration of molecular species. The prototype can only model "simple" signaling networks, with activation and inhibition reactions, but can be easily extended to support other types of reactions because of the abstraction used. The spatial information of signaling networks is not taken into account during this research.

The graphical model can be translated in a one-to-one manner to a model of Timed Automata in the UPPAAL format. With UPPAAL, it is possible to check such a model for certain specifications. For now, no safety and liveness queries are used in the prototype, but this is possible. After model checking, the results can be loaded back into the prototype to visualize them. This visualization is based on the model entered, and has snapshots that show the course of the activity in the model. These snapshots can be saved as an animation. It is possible to get the activity for individual components of the network, and this activity can be drawn in a chart showing the activity level over time for one or more components.

In Table 7.2 some test results are presented. The tests are run on a laptop with an Intel Pentium M 1,60GHz processor, having Windows XP as operating system. The results reflect the time and number of states visited with the UPPAAL tool [1]. As can be seen in the table, only a few second are needed to verify a reachability query; one of the reasons for this is that the state specified in the query is always reachable. The reachability query used for testing asks for a state of the system where the first and the last component are in their inactive state.

The models used for testing do not have loops in them; they consist of a line of components. The only type of reaction used in the results is the activation of the downstream component. At $t = 0$, the first component is in its highest state of activity, and the rest of the components is inactive. Therefore, the trace that leads to the state where the first and the last component are in their inactive state is also the final state of system. The timing of the activations is always between 1 and 5, where the value of 5 is used for the lowest state(s), and 1 for the highest state(s). The timing of the degeneration is always between 1 and 20, where the value of 20 is used for the lowest state(s), the inactive state excluded. And the value of 1 is used for the

[1] The number of states visited by UPPAAL is not the same as the number of states for the model after parsing the trace. Every synchronization step in the Timed Automata model is described in a new state in the UPPAAL trace file. In the prototype, only the states that result in a new activity level of one of the components are taken into account.

highest states. The timing of all the components in the model with 3 components and 3 levels of activity (A activates B activates C) can be found in Table 7.1.

Table 7.1: Values for the timing of the model with 10 components and 3 levels of activity, as used for the testing results. For A and B, the timing parameters for the activation of their downstream components (B and C respectively) are also given.

State	Degeneration	Activation
A and AB		
0	-	-
1	20	4
2	13	2
3	6	1
B and BC		
0	-	-
1	20	4
2	13	2
3	6	1
C		
0	-	-
1	20	-
2	13	-
3	6	-

If a non-reachable state is specified in the query, it takes really long to verify this. For comparison, the time listed in the table for a model that consists of 10 components, each with three levels of activity is 1,4 seconds. The verification of a non-reachable query on the same model can easily take over two hours, as all possible states and traces have to be explored before UPPAAL knows for sure that no proof exists for the given query. If instead of a random dept-first search, a breadth-first search is used for model checking, it will take a lot longer to verify a query, if it is reachable.

The results in Table 7.2 clearly shows a linear increase of time with respect to the number of states in the trace. The number of states in a trace however depends for a great deal on the query. The oscillator from the previous chapter for example, and the first run described in particular, resulted in a trace consisting of over 50,000 states. It took less then 6 seconds to generate this trace. The parsing of these 50,000 states (both executing the tracer output and piping and parsing the trace) took about 20 seconds.

One of the possible uses of the prototype was to play around with parameters in order to get to understand a small part of a signaling network. This usage will pose no problem with respect to the time it takes to generate a trace, provided that a query is used in which the specification given describes a reachable state. More serious models, covering bigger parts of signaling networks, might take a long time to verify. When the information necessary for adequate modeling of these parts becomes available, it might be worth to look into the possible usage of clusters of computers for verification [2]

[2]I think it will take some time (years) before the information necessary for the modeling of big parts of

Table 7.2: Model checking results for reachability queries on networks with different components and levels of activity. Settings used in UPPAAL: Random depth first search with some resulting trace. "Components" refers to the number of components used in a simple setup of a network. "Activity Levels" is the maximum level of activity, levels start at 0. "Number of States" refers to the number of states this particular run returned as trace. "Time Elapsed" is the time that UPPAAL took to process the reachability query. "Size of Trace" is based on the form of the trace used by UPPAAL (.xtr format).

Components	Activity Levels	Number of States	Time Elapsed	Size of Trace
2	3	106	0,3 sec	11 KB
3	3	316	0,2 sec	44 KB
4	3	636	0,2 sec	110 KB
5	3	1,075	0,6 sec	225 KB
5	5	1,675	0,3 sec	359 KB
5	10	4,124	0,6 sec	913 KB
5	25	10,874	1,6 sec	2,436 KB
10	3	5,082	1,4 sec	2,048 KB
10	5	8,472	3,0 sec	3,509 KB
10	10	21,438	5,5 sec	9,201 KB
10	25	60,625	15,3 sec	26,075 KB
15	3	12,222	7,3 sec	7,251 KB
15	5	19,990	10,6 sec	12,213 KB
15	10	52,932	28,6 sec	33,519 KB
15	25	150,406	73,2 sec	94,754 KB

signaling networks becomes available. In the meantime insights regarding the modeling of signaling networks might have changed.

Chapter 8

Conclusions and Future work

The goal of this research was to make a prototype with which a biologist can graphically model a signaling network which can be translated to a system of Timed Automata networks. This is now possible for "simple" signaling networks; only activation and inhibition as reactions between components can be modeled. There are however more types of reactions possible in biochemical networks. More research should be done on how these reactions can be modeled using the Timed Automata approach of this research. For some components in a signaling network, the rest state is completely active, and after inhibition this component can fall back to its most active state. This is also some kind of "degeneration", which cannot be expressed in the prototype yet.

As the prototype only shows a tip of the iceberg with respect to the possible gains this type of modeling might give, it would be nice to work out a more complex biological example and compare results of this model with experimental results, or use the experimental results to verify a model of a signaling network.

The representation of the all the types of interaction in the prototype is now exactly the same; it is displayed as an arrow between two nodes in the graph. To make a visual different between the types of interaction will enhance the information that can be relayed by the visual representation of a network. One way to do this is by making the representation of an interaction be more then just an arrow. This can be done by how the nodes of the graphical model are placed, how they are connected with each other, or maybe even by using a simple animation.

The prototype was built with the intended use to model parts of signaling networks that consist of kinases, but in theory the same abstraction can be used for other types of networks. The user should be able to distinguish between the different types of networks easily, which can be done by using different types of representation. How these representations should look, is a point of interest.

The timings of the interactions between the components of a signaling network are now modeled with time intervals, where the user inputs the value that represents the middle of the interval. In the prototype, the interval consists now only of this value, but this can be changed rather easily to an interval that encompasses more values. As most information on signaling networks does not express the timing of the reactions as time intervals, research can be done on how to use existing data and existing equations as input.

In the prototype the queries that can be posed to the UPPAAL model cover the reachability of states of the system. There exist however more model checking questions that can be used with

Timed Automata, which open the door for other knowledge that can be extracted from models of signaling networks. What kind of model checking questions should be used depends on the type of information a user wants to obtain from the model. The information that is desired as well as how to get this information using model checking questions both are interesting subjects that need to be researched.

As this prototype is "yet another tool" to visualize biochemical networks, it might be fruitfull to write a plugin for a tool that is already widely used by biologists, for example Cytoscape. A lot of plugins already exist for Cytoscape that model regulatory networks but not yet with respect to the timing of a complete network. For an example plugin, the user is invited to look at the BioQuali plugin by Guziolowski [Guziolowski et al., 2009].

The prototype uses an xml format as input, hereby making it easier to write some small piece of software that can translate the results of experiments to a model that can be read by the prototype. How such an xml file should be constructed, can be found in Appendix C.2.

For the future, it might be interesting to look into the possibilities of matching the parameters of a model automatically to experimental results.

Bibliography

Bruce Alberts, Dennis Bray, Karen Hopkin, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Essential Cell Biology*. Garland Science, 2 edition, September 2003. ISBN 0815341296. URL `http://www.garlandscience.com/textbooks/081533480X.asp`.

Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on uppaal, 2004. URL `http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf`.

Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools, 2004. URL `http://www.springerlink.com/content/2agtg3kjfle73j5a/`.

Muffy Calder, Vladislav Vyshemirsky, David Gilbert, and Richard J. Orton. Analysis of signalling pathways using continuous time markov chains. 4220:44–67, 2006. URL `http://dblp.uni-trier.de/db/journals/tcsb/tcsb6.html#CalderVGO06`.

Federica Ciocchetta and Jane Hillston. Bio-pepa: A framework for the modelling and analysis of biological systems. *Theoretical Computer Science*, In Press, Corrected Proof, 2009. ISSN 0304-3975. doi: DOI:10.1016/j.tcs.2009.02.037. URL `http://www.sciencedirect.com/science/article/B6V1G-4VT14KW-1/2/d75a955fd6da7cf9a3eb25abf3b8ebbf`.

Cytoscape Consortium. Cytoscape: Analyzing and visualizing network data, 2001–2008. URL `http://www.cytoscape.org`.

Kam D. Dahlquist, Nathan Salomonis, Karen Vranizan, Steven C. Lawlor1, and Bruce R. Conklin. Genmapp, a new tool for viewing and analyzing microarray data on biological pathways. *Nature Genetics*, 31:19–20, 2002. doi: 10.1038/ng0502-19. URL `http://dx.doi.org/10.1038/ng0502-19`.

Hidde de Jong. Modeling and simulation of genetic regulatory systems: a literature review. *Journal of Computational Biology*, 9(1):67–103, 2002. ISSN 1066-5277. doi: 10.1089/10665270252833208. URL `http://dx.doi.org/10.1089/10665270252833208`.

Bruce Conklin et al. Genmapp, 2002–2009. URL `http://www.genmapp.org`.

Stephen Gilmore. Bio-pepa, 2008–2009. URL `http://www.dcs.ed.ac.uk/home/stg/software/biopepa/`.

Stephen Gilmore and Jane Hillston. The pepa workbench: A tool to support a process algebra-based approach to performance modelling. In *In Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, number 794 in Lecture Notes in Computer Science*, pages 353–368. Springer-Verlag, 1994.

Stephen Gilmore and Jane Hillston. Pepa, 1994–2009. URL `http://www.dcs.ed.ac.uk/pepa/`.

Carito Guziolowski, Annabel Bourd, Francois Moreews, and Anne Siegel. Bioquali cytoscape plugin: analysing the global consistency of regulatory networks. *BMC Genomics*, 10 (244), May 2009. doi: 10.1186/1471-2164-10-244. URL `http://dx.doi.org/10.1186/1471-2164-10-244`.

SRI International. Pathway logic, 2006–2009. URL `http://pl.csl.sri.com/`.

Ltd JGraph. Java graph visualization and layout, 2001–2009. URL `www.jgraph.com`.

Ross D. King, Jem Rowland, Stephen G. Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N. Soldatova, Andrew Sparkes, Kenneth E. Whelan, and Amanda Clare. The automation of science. *Science*, 324(5923):85–89, April 2009. doi: 10.1126/science.1165620. URL `http://dx.doi.org/10.1126/science.1165620`.

M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002.

Marta Kwiatkowska, Gethin Norman, Dave Parker, and Mark Kattenbelt. Prism: Probabilistic symbolic model checker, 2004–2009. URL `http://www.prismmodelchecker.org/`.

Nagasaki M., Doi A., Matsuno H., and Miyano S. Genomic object net:i. a platform for modeling and simulating biopathways. *Applied Bioinformatics*, 2:181–184, 2004.

Friedrich Marks, Ursula Klingmüller, and Karin Müller-Decker. *Cellular Signal Processing - An Introduction to the Molecular Mechanisms of Signal Transduction*. Garland Science Publishing, November 2008. ISBN 978-0-8153-4215-1.

José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/0304-3975(92)90182-F. URL `http://dx.doi.org/10.1016/0304-3975(92)90182-F`.

J.A. Papin, T. Hunter, B.O. Palsson, and S. Subramaniam. Reconstruction of cellular signalling networks and analysis of their properties. *Nature Reviews — Molecular Cell Biology*, 6, February 2005. URL `http://www.ncbi.nlm.nih.gov/pubmed/15654321`.

Corrado Priami. Algorithmic systems biology. *Communications of the ACM*, 52(5):80–88, May 2009. ISSN 0001-0782. URL `http://mags.acm.org/communications/200905/`.

P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res*, 13(11):2498–2504, November 2003. ISSN 1088-9051. doi: 10.1101/gr.1239303. URL `http://dx.doi.org/10.1101/gr.1239303`.

Matthew Suderman and Michael Hallett. Tools for visually exploring biological networks. *Bioinformatics*, 23(20):2651–2659, 2007. doi: 10.1093/bioinformatics/btm401. URL `http://dx.doi.org/10.1093/bioinformatics/btm401`.

Inc. Sun Microsystems. Developer resources for java technology, 1994–2009. URL `java.sun.com`.

Carolyn Talcott. Pathway logic. In *Formal Methods for Computational Systems Biology*, volume 5016 of *LNCS*, pages 21–53. Springer, 2008. 8th International School on Formal Methods for the Design of Computer, Communication, and Software Systems.

Tokyo University. Cell illustrator 4.0, 2009. URL `https://www.cellillustrator.com/home`.

Martijn P van Iersel, Thomas Kelder, Alexander R Pico, Kristina Hanspers, Susan Coort, Bruce R Conklin, and Chris Evelo. Presenting and exploring biological pathways with pathvisio. *BMC Bioinformatics*, 9(399), 2008. doi: 10.1186/1471-2105-9-399. URL `http://dx.doi.org/10.1186/1471-2105-9-399`.

Martijn van Iersel et al. Pathvisio, 2008. URL `http://www.pathvisio.org`.

Appendix A

Modeling difficulties

A.1 Broadcast

In UPPAAL there are several ways for synchronizing on actions; (urgent) channels and broadcasts. In this research, a molecular species process should be able to influence, or synchronize with, one or more other such species (in biology: the reactions). To have a modular solution that can be easily extended, a mixture of both normal channels and broadcast channels is used for synchronization. One of the important factors to choose this mixture of both is because no guards can be placed on broadcast synchronization actions, which is necessary to model the timing of reaction.

In the solution, for every reaction between two molecular species processes a reaction process is added to the UPPAAL system. This reaction process listens constantly to broadcast signals from the sending species process, which convey changes in the activity level of this sending process. The reaction process then acts upon this broadcast signal, by delaying for a certain time interval and then passing on a signal to the receiving species process. In this research, only activation and inactivation reactions are modeled, so the signal to the receiving process will be either a *down* or an *up* signal.

In Figure A.1 this way of modeling reactions is illustrated by a (simplified) system consisting of one sending process, Process X, and two receiving processes, Y and Z. The reaction processes that connect the species processes, XY and XZ, are listening for the broadcast signal $X.DO!$ that is generated by process X. After synchronizing on this broadcast, both reaction processes can pass on a signal to their respective receiving species processes Y and Z. In this example, both reaction processes present a activation process that will change the level of activity of the receiving species processes to a higher level, in this case from inactive to active. How this is done exactly in the research can be found in Appendix B.1, where the actual templates for the molecular species and the reactions are explained.

A.2 Parsing the result from UPPAAL

It is possible to generate simulation trace with the prototype that can be parsed to a visual result. In this section, the commands used to generate the trace and a short description of the parsing of this trace will be explained.

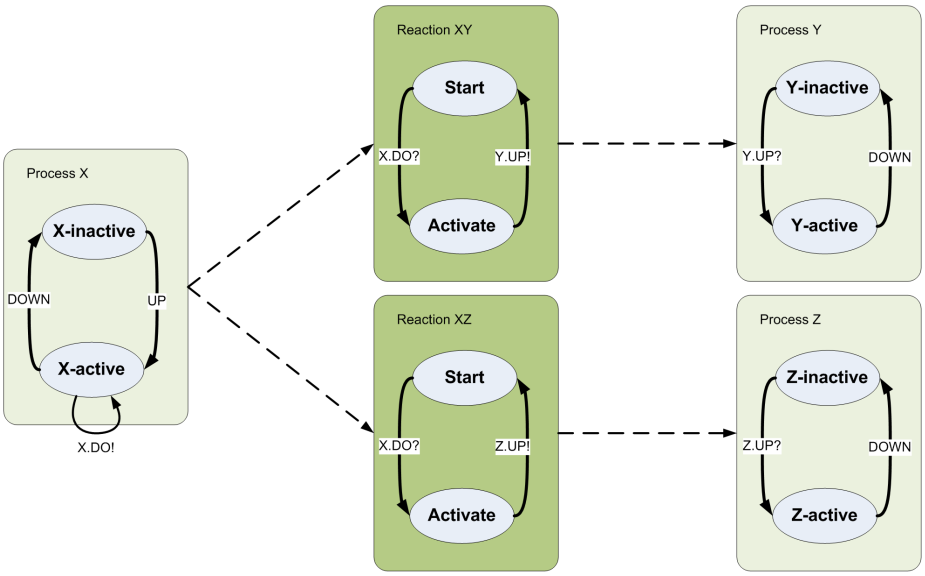


Figure A.1: Broadcasting with seperate reaction processes to handle the timing of the reaction

With UPPAAL comes a command line tool, verifyta, which takes care of finding a proof for model checking questions with regard to a specific UPPAAL system. Normally, verifyta results are traces that are within an obscure and optimized format that is used within UPPAAL, but it is possible to create trace results that can be parsed by a third party. First, a normal trace is generated. Then another output is created which is necessary to generate the readable trace with the command line tool tracer. To create a readable trace for a system called test, which will be named test.xml, and its queries, called testqueries.q, the following command line commands will be issued (Windows example):

To generate a normal trace, with the name testresult-1.xtr (-ftestresult tells verifyta to write the output to testresult-1.xtr), the following command is exectued. The −o2 in this command stands for the way the reachability query is processed; a random depth first search. The −t0 tells UPPAAL to not only verify the query but also to generate some trace to a state that satisfies the query. The −Y tells UPPAAL to generate a symbolic trace.

```
verifyta.exe -o2 -t0 -Y  ftestresult test.xml testqueries.q
```

To be able to parse the resulting trace from the previous step to a readable trace, the next steps are performed. First, a variable called UPPAAL_COMPILE_ONLY is set which is read by verifyta. After this, verifyta is run again with the same parameters as in the previous step; this produces the needed output for the tracer. The output is saved as testtrace.out.

```
SET UPPAAL_COMPILE_ONLY=true
verifyta.exe -o2 -t0 -Y test.xml testqueries.q > testtrace.out 2>&1
```

The last step is to parse the result from the previous step and the normal trace to a readable trace called testtrace.trc, with the help of the tracer command line tool. How this resulting trace is build up can be found in Appendix C.1.

```
tracer.exe testtrace.out testresult-1.xml > testtrace.trc 2>&1
```

To translate this trace back to the graphical model in the prototype, the locations of the molecular species processes that are not committed (with the names n0, n1 and so on) are filtered out, together with the value of the globalTime variable. Together they form the states of the graphical model at certain timestamps. Not for all the timestamps is a state available, as only states that differ from a previous state can be found in a trace file; it is possible to have a timestamp x and y that differ 10 time units, or only 1. If they differ 10 time units, nothing changes in the activity levels of the molelar species between time x and y.

As the .trc files generated in the last step can become very large, a parser is build in Java that immediately parses the trace by means of "piping". This Java code is also Windows specific and can be found in the file TraceParser.java. It takes every state line in the trace file and filters out the activity levels of the molecular species and the globalTime value, and if a change in time and/or activity is detected, a new timestamp state is written to a model that can be opened by the prototype.

The parsing of the file is done in the saveTracefileAsSystem method in SaveReults.java; a seperate StateParser thread is used to translate the symbolic trace lines.

Appendix B

UPPAAL

B.1 UPPAAL templates

Modeling in UPPAAL can be simplified by the use of templates. A template is a predefined model (or network of timed automata) where some or all of the variables can be given as parameters on instantiation. This way a single template can have a lot of instantiations with different parameters that together form a system of networks of timed automata. In the prototype three templates are generated for each system, based on the number of states used. In this appendix templates for a system that uses four states of activity (including the inactive state) are described in detail.

In the templates locations (circles) and actions (arrows) can be seen. If a location contains a C it is a committed location. If a state of the system is committed, i.e. contains a committed location, there is no delay possible and the next transition must involve an outgoing edge of at least one of the committed locations. The committed locations in the templates are used to deal with the state changes.

Every template should start in a given location, which is the double lined circle.

The instances of the templates are glued together with broadcast channels and normal channels on which they synchronize. These are passed on to the templates in the form of parameters, together with timing parameters for both the reaction and the degeneration template.

B.2 Molecular Species template

The parameters for the molecular species template:

- *int startstate* – The state/activity location the instance should start
- *chan &down* – The channel on which to synchronize on as receiver for the *down* signal
- *chan &up* – The channel on which to synchronize on as receiver for the *up* signal
- *broadcast chan &s_br*[4] – The array of broadcast channels on which to synchronize on as sender for the state change signals

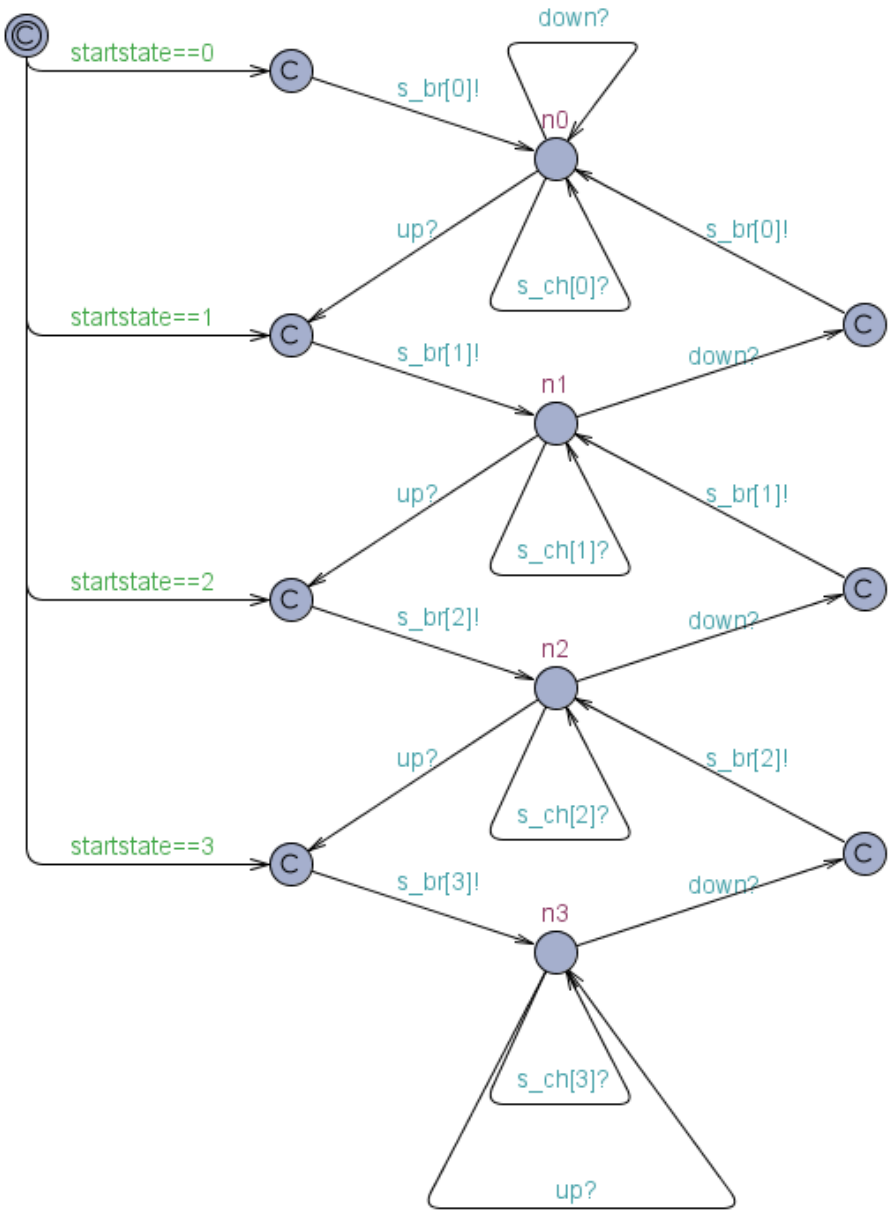


Figure B.1: Molecular Species Template

- *chan &s_ch*[4] – The array of channels on which to synchronize on as receiver when another process (degeneration or reaction) wants to know in which state of activity this process is.

The molecular species template looks like the way the population of RAF is modeled in Figure 3.5, with some committed locations added. The committed locations take care of the communication around the state/activity changes of the modeled population. As the template is used for all the different molecular species in the system, a parameter to declare in which state/activity

the instance of the template starts is used, called the *startstate*. The initial location of the template is a committed location and thus the instance proceeds immediately to the next state. The guards on the outgoing arcs define which location this will be. The next location is also committed, and by proceeding along the outgoing arc, the instance broadcasts the state it will be in.

The template itself has no clock and hence no guards or invariants with respect to time; all the rest of the state/activity changes is done by synchronizing on both reactions and the degeneration instance, which give the *up* and *down* signals on which it will synchronize.

If a reaction or a degeneration process needs to know in which state the molecular species process is, it can try to synchronize on one of the normal channels. Most of the state changes are communicated via a broadcast signal, but it can happen that a reaction process needs this information again, after performing an (de-)activation action on the downstream molecular speciess process. This is actually the only scenario in which this type of synchronization is used in the prototype.

B.3 Reaction template

The parameters for the reaction template:

- *chan &action* – The action channel the reaction template should synchronize on as sender
- *broadcast chan &s_br*[4] – The array of broadcast channels on wich to synchronize on as receiver for the state change signals
- *int tijd*[3] – The array of time constraints taken into account when waiting to pass on the action signal
- *chan &s_ch*[4] – The array of normal channels on which to synchronize on as sender when the reaction process wants to know in which state the sending process is.

All the reactions between two molecular species processes are modeled with reaction templates. The initial location of the template is to wait for a state/activity change of the "sending" process ($sarray[i]$). If a broadcast is sent from this process, the reaction then proceeds to one of the locations in the middle, and, by proceeding to such a state, sets the clock of the template to 0. From here the following four things can happen:

1. The time the reaction has to wait, before it will pass on the signal, has elapsed ($clock\_u >= tijd[i]$). The reaction now proceeds to a committed location, and proceeds from this location to another committed location, hereby sending a signal to the receiving process. This signal is either an *up* or a *down* signal. After sending the signal, the reaction tries to synchronize with the sending process on one of the shared normal channels. With this synchronization, the reaction process gets to either the initial state or to one of the other states.
2. A state change to a "higher" state is received. The reaction now proceeds to a committed location ($sarray[i]$) and based on the internal clock ($clock\_u >= tijd[i]$) the reaction either proceeds to the committed location that sends the signal, or to the internal next state (n_i).

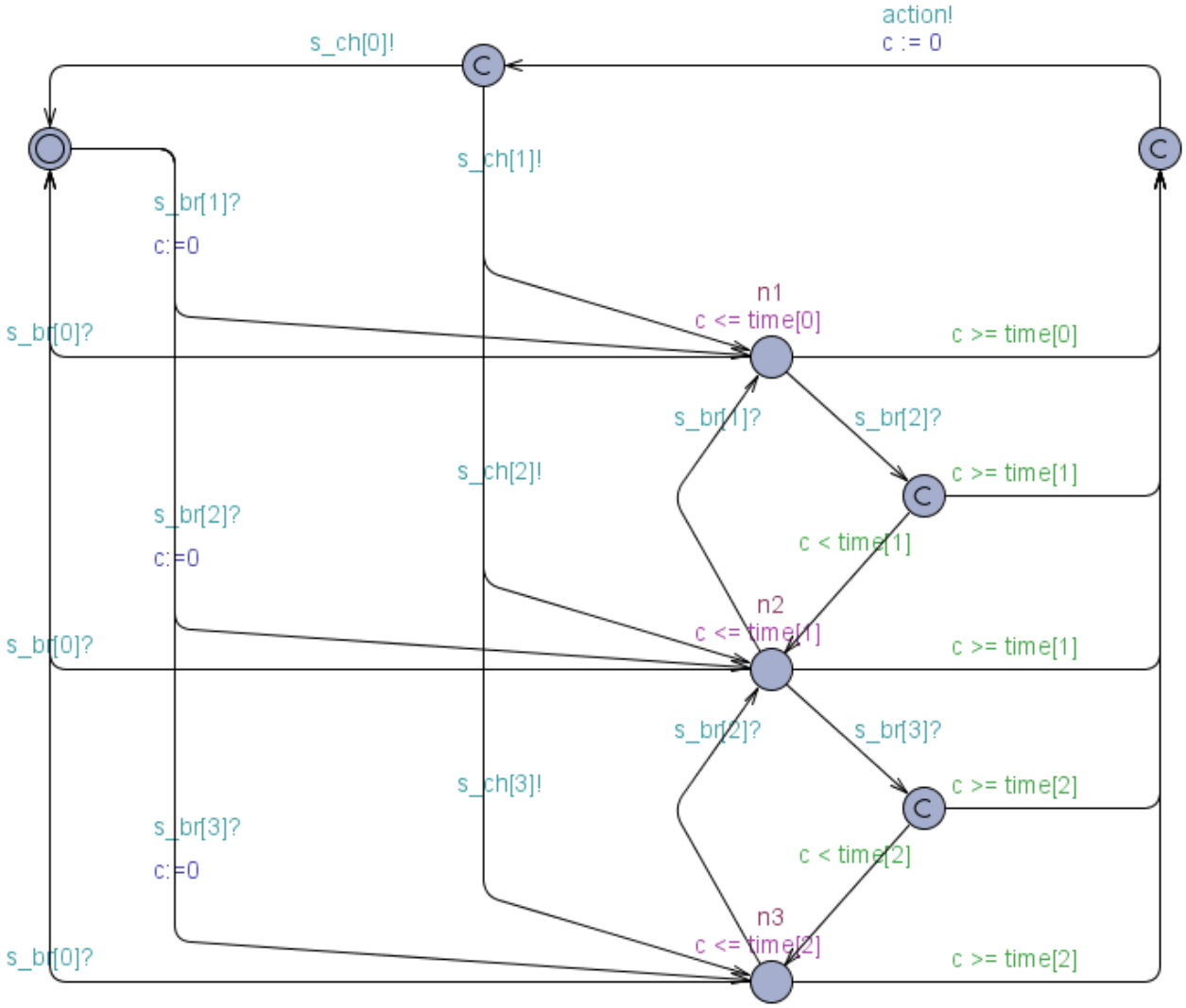


Figure B.2: Reaction Template

3. A state change to a "lower" state is received. The reaction now proceeds to this lower state (n_i).

4. A state change to the inactive state is received. The reaction now proceeds to the initial state.

This template has its own clock, called c.

B.4 Degeneration template

The parameters for the degeneration template:

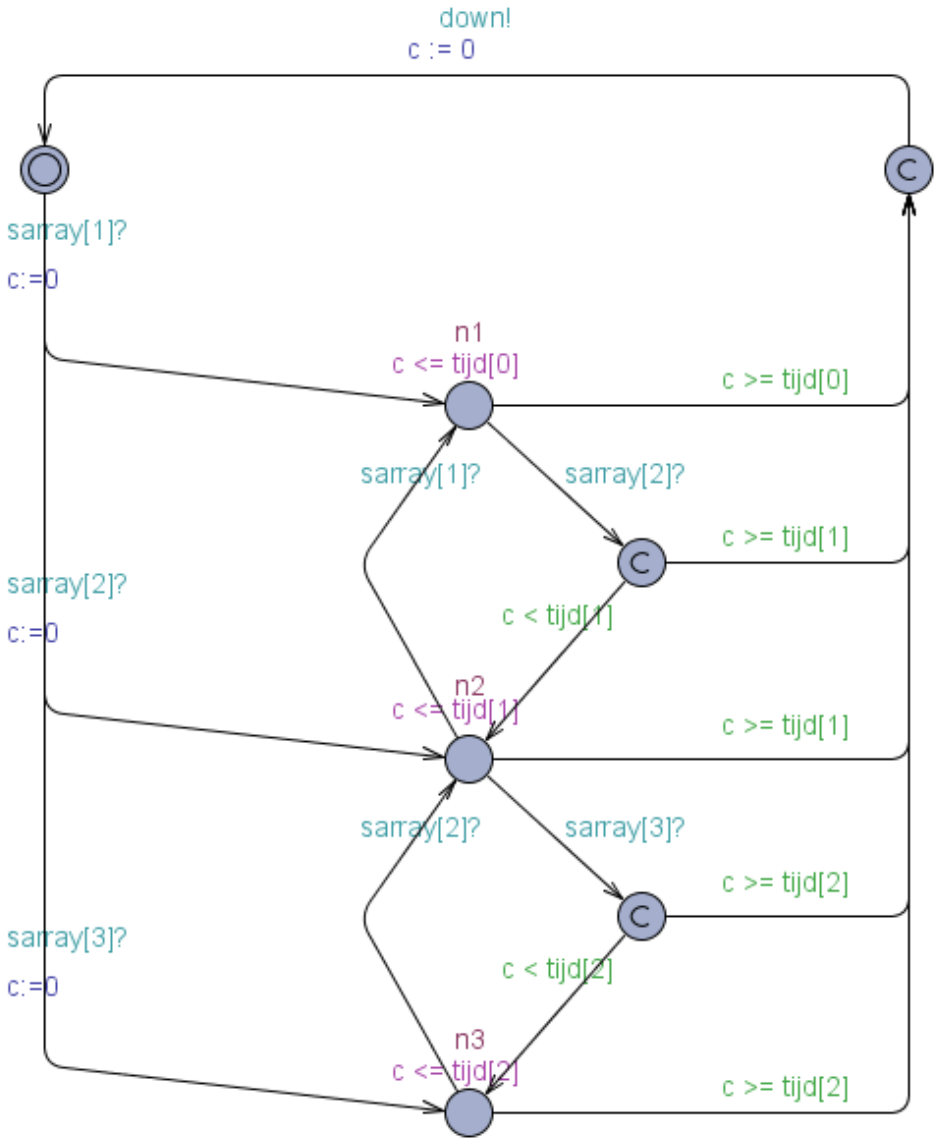


Figure B.3: Degeneration Template

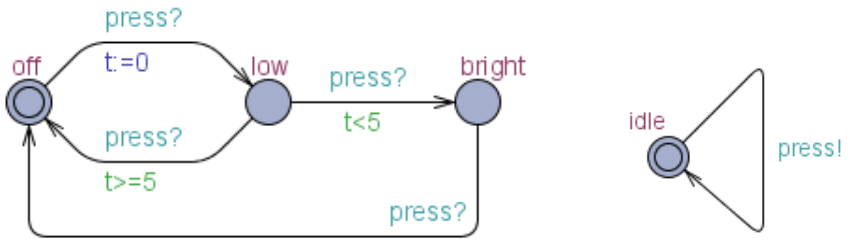
- *chan &down* – The down channel the reaction template should synchronize on as sender
- *broadcast chan &sarray*[4] – The array of broadcast channels on wich to synchronize on as receiver for the state change signals
- *int tijd*[3] – The array of time constraints taken into account when waiting to pass on the *down* signal

The degeneration template is more or less the same as the reaction template, one of the differences being that it does not act on an inactive state signal it receives but continues waiting untill it can send a *down* signal. In the molecular species template the reaction to this signal

is the arc which synchronizes on the *down* signal from the inactive location (n_0) to the same location. The other difference is, that the degeneration process does not need to synchronize againg with the molecular species process it belongs to. After sending the *down* signal, the molecular species process will send a broadcast with its new state, on which the degeneration process can synchronize. This template has its own clock, called *c*.

B.5 Composing a simple UPPAAL system

In Section 3.2, an UPPAAL model of a lamp and a user is used to explain how UPPAAL and Timed Automata work. How to compose an UPPAAL system with the templates in Figure B.5 as building bricks, is explained in this section. The resuling system consists of one lamp and one user in which the user can send a *press?* signal to the lamp. Understanding the composition of this simple UPPAAL system is necessary to understand the composition of the systems done by the prototype, which is explained in the next section.



(a) Model of the Lamp (b) Model of the User

Figure B.4:

A variable that is available for every instantiation of a template is declared in the declarations section of this template. Every instantiation of this template will get its own variable, in this case its own clock, to work with.

```
// local declaration, for the lamp template
clock y;
```

A variable that is used systemwide between processes, for example a channel to synchronize on, is declared in the Systems declarations section.

```
// global declaration
chan press;
```

To actually build a system, first the processes of which the system will be composed have to be instantiated from the available templates.

```
// template instantiations
yellowLamp = lamp();
ourUser = user();
```

And this is the actual composition of the system, where the processes *yellowLamp* and *ourUser* are composed into a system. The *yellowLamp* has its own clock, and the two processes synchronize on the *press* channel.

```
// list the processes to be composed into a system
system yellowLamp, ourUser;
```

B.6 Composing the UPPAAL systems of signaling networks

In this section an example of the composition of an UPPAAL system based on the templates earlier discussed in this appendix will be given. It consists of the composition of the small part of a kinase network where RAF activates MEK (see also Figures 3.1(a) and 4.2).

The declaration of the channels used for the molecular species templates for changing the state/activity.

```
1  chan Raf_up ;
2  chan Raf_down ;
3  chan Mek_up ;
4  chan Mek_down ;
```

This is the code generated from the molecular species node (RAF) in the graphical model; both the degeneration template and the molecular species (process) template are instantiated. In line 5 an array with the timing values for the degeneration is constructed (first is for state 1 and the last for state 3 which is the highest state). In line 6 an array of broadcast signals, to be able to communicate the state/activity changes, is constructed which is used for both the templates as well as for the upcoming reaction template instantiation.

```
5  const int Raf_t[3] := {20,13,6};
6  broadcast chan Raf_c[4];
7  Raf = ProcessTemplate(3,Raf_down, Raf_up, Raf_c );
8  Raf_deg = DegenerationTemplate(Raf_down,Raf_c, Raf_t );
```

This is the code generated for the MEK node, which is similar to the code generated for RAF.

```
9  const int Mek_t[3] := {20,13,6};
10 broadcast chan Mek_c[4];
11 Mek = ProcessTemplate(0,Mek_down, Mek_up, Mek_c );
12 Mek_deg = DegenerationTemplate(Mek_down,Mek_c, Mek_t );
```

Here is the code for the reaction; in line 13 an array with the timing values for the reaction (first is for state 1 and the last for state 3 which is the highest state). The reaction template uses the channel of line 3 to make the MEK process go up. Together with the array of broadcast channels of line 6, to get the state information from the RAF process, and the timing array of line 13 they can be used to instantiate the reaction template.

```
13 const int Raf_r_t[3] := {4,2,1};
14 Mek_r_up = ReactionTemplate(Mek_up,Raf_c,Raf_r_t);
```

The system is now composed from the instantiated templates.

```
15 system Raf, Raf_deg, Mek, Mek_deg, Mek_r_up;
```

B.7 UPPAAL Queries

In the prototype two types of reachability queries are used. The first query is one with respect to the time, and has as purpose to perform a simulation of at least x time units. The second query concerns the reachability of a state, where the state is defined by the locations of one or more processes, possibly accompanied by a time constraint. An example of the first query might be:

```
E<> globaltime > 100
```

This query can be translated to: *There exists a state of the system reachable from the given state (begin state) for which the globaltime exceeds 100 time units.* The answer, or proof, to such a query is a possible trace to a state that satisfies this constraint.

The query used to see if a system can reach a certain state, given a start condition, might be as follows (note: not all the processes have to be used in this type of query):

```
E<> ERK.n5 && MEK.n0
```

This query can be read as: *There exists a state of the system, that is reachable, in which ERK is in location n5 and MEK in location n0.* The n-states stand for the level of activity of the processes, in this case MEK should be in the inactive state and ERK in state 5. This type of query can be extended to include a timestamp as a constraint. This can be either an upper or a lower bound, or the exact time this state will occur, found in the following example queries:

```
E<> ERK.n5 && MEK.n0 && globalTime > 100
E<> ERK.n5 && MEK.n0 && globalTime < 100
E<> ERK.n5 && MEK.n0 && globalTime == 100
```

Appendix C

Formats

C.1 UPPAAL trace file

The trace files that can be generated using an UPPAAL trace file and the tracer tool, as described in Appendix A.2, can be desribed by the following syntax (NL stands for a newline, INT for an integer value, either positive or negative):

file	::=	*state* (*transitions state*)∗
state	::=	"State: " *location* + *variable* ∗ *clocks* NL
transitions	::=	"Transition:" *transition*+ NL
location	::=	*name* DOT *place*
clocks	::=	(*clock* − *clock*(">=" | ">" | "<" | "<=") INT)+ NL
transition	::=	*location* − > *location* "" *select* "," *sync* "," *update* "" NL
variable	::=	(*name* DOT)? *variable* = INT
clock	::=	"t(0)" | "globalTime" | *name* DOT *clockname*

The clock values always present an (open) interval and are based on integer values and other clocks, of which one is always zero; t(0). There is one global clock in the UPPAAL systems generated by the prototype, which is called globalTime.

Example of clock values:

```
t(0)-globalTime<=-163
globalTime-t(0)<=175
globalTime-SIGNAL_deg.c<=101
```

The variables present the variables used for instantiating the templates. In the final prototype, this is limited to the startstate. The rest of the variables to instantiate the templates, the channels and the constant times, do not appear in the trace.

Example of a startvalue:

```
SIGNAL.startstate=2
```

C.2 IKNAT system

```
<iknat>
  <concentrations>integer</concentrations>
  <model>
    <jgraphnode>
      <name>string</name>
      <bounds>x=float y=float h=float w=float</bounds>
      <border>boolean</border>
      <iscell>boolean</iscell>
      <opaque>boolean</opaque>
      <identifier>integer</identifier>
    </jgraphnode>
    ...
    <jgraphedge>
      <isedge>boolean</isedge>
      <lineEnd>integer</lineEnd>
      <endFill>boolean</endFill>
      <identifier>integer</identifier>
      <fromSource>integer</fromSource>
      <toTarget>integer</toTarget>
    </jgraphedge>
       ...
   </model>
  <data>
    <datanode>
      <identifier>integer</identifier>
      <degeneration state="integer">integer</degeneration>
      <degeneration state="integer">integer</degeneration>
    </datanode>
    ...
      <dataedge>
      <identifier>integer</identifier>
      <positive>boolean</positive>
      <reaction state="integer">integer</reaction>
      <reaction state="integer">integer</reaction>
      </dataedge>
      ...
      <timeslice>
      <timestamp>integer</timestamp>
      <statenode>
        <identifier>integer</identifier>
        <state>integer</state>
      </statenode>
          ...
      </timeslice>
       ...
   </data>
</iknat>
```

C.3 UPPAAL system file

Table C.1: Information available via `http://www.it.uu.se/research/group/darts/uppaal/flat-1_1.dtd`.

file	::=	*info* <nta> *imports*? *declaration*? *template*+ *instantiation*? *system* </nta>
info	::=	<?xml version="1.0" encoding="utf-8"?> <!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN' 'http://www.it.uu.se/research/group/darts/uppaal/flat-1_1.dtd'>
imports	::=	*(not used in this research, see www.uppaal.com for more information)*
declaration	::=	<declaration> STRING </declaration> *(Declaration of variables for either system-wide or template)*
template	::=	*name parameter*? *declaration*? *location* $*$ *init*? *transition*$*$
name	::=	<name x="INT" y="INT"> STRING </name> *(name label with name and x/y coordinates)*
parameter	::=	<parameter> STRING </parameter> *(parameters to instantiate template)*
location	::=	<location id="id INT" x="INT" y="INT"> *name*? *label* $*$ *urgent*? *committed*? </location> *(ID is mandatory and unique throughout system)*
init	::=	<init ref="id INT"/> *(Start or initial state of the template)*
urgent	::=	<urgent/>
committed	::=	<committed/>
transition	::=	<transition> *source target label* $*$ *nail*$*$ </declaration>
source	::=	<source ref="id INT" />
target	::=	<target ref="id INT" />
label	::=	<label kind="STRING" x="INT" y="INT"> STRING </label> (Kind is any of "guard", "assignment", "synchronization" or "invariant")
nail	::=	<nail x="INT" y="INT" />
instantiation	::=	*(not used in this research, see www.uppaal.com for more information)*
system	::=	<system> STRING </system> (Here come all the template instantiations and system composition)

Appendix D

Parameters used for the oscillator

The parameters used in the oscillator example in Chapter 6 can be found in the two tables below. The idea was to model a signal that activated the Protein for (at least) 10,000 time units. The implicit inactivation of the Protein is set to take very long, to make sure this did not interfere with the rest of the reactions between the components. The numbers between parantheses have to be entered in the prototype, but are not actually used in generating the trace. There are two sets of parameters used for modeling the oscillator, which only differ in the implicit inactivation of the Inhibitor components. Therefore an extra column is added to display both these sets of parameters in Table D.1. The timing parameters used to model the reactions are exactly the same in both compositions, and therefore Table D.2 shows the reaction timings for both.

Table D.1: Degeneration parameters, for the inhibitor the degeneration is changed for the second run to create the frequence modulation.

State	Signal	Protein	Activator	Inhibitor (1)	Inhibitor (2)
1	(1)	10,000	40	40	100
2	(1)	10,000	37	38	95
3	(1)	10,000	34	37	91
4	(1)	10,000	31	35	87
5	(1)	10,000	29	34	82
6	(1)	10,000	26	32	78
7	(1)	10,000	23	31	74
8	(1)	10,000	21	30	70
9	(1)	10,000	18	28	65
10	(1)	10,000	15	27	61
11	(1)	10,000	12	25	57
12	(1)	10,000	10	24	52
13	(1)	10,000	7	22	48
14	(1)	10,000	4	21	44
15	10,000	10,000	2	20	40
Startstate	15	0	0	0	0

Table D.2: Reaction parameters (P stands for Protein)

State	Signal-P	P-Inhibitor	P-Activator	Inhbitior-P	Activator-P
1	(10,000)	35	10	5	25
2	(10,000)	33	9	4	23
3	(10,000)	32	8	4	22
4	(10,000)	30	8	4	20
5	(10,000)	29	7	4	19
6	(10,000)	27	7	3	17
7	(10,000)	26	6	3	16
8	(10,000)	25	6	3	15
9	(10,000)	23	5	3	13
10	(10,000)	22	4	3	12
11	(10,000)	20	4	2	10
12	(10,000)	19	3	2	9
13	(10,000)	17	3	2	7
14	(10,000)	16	2	2	6
15	5	15	2	2	5

Appendix E

CSV output

In this appendix, an example of the comma separated value file is presented. The prototype generates this file so the user can express the activity levels of the molecular species components in the model. Columns in the csv file format are separated using a comma:

```
Time, Activator, Inhibitor, Protein, Signal
5, 0, 0, 1, 15
10, 0, 0, 2, 15
14, 1, 0, 2, 15
15, 1, 0, 3, 15
20, 1, 0, 4, 15
...
```

The first column represents the value of the time, or the specific point in time, for which the state holds. All the columns thereafter contain the activity levels of the molecular species components in the model. After opening the file in for example Microsoft's Excel, it will look like the data in Table E.1.

Table E.1: Comma Separated Value file example, as generated by the prototype

Time	Activator	Inhibitor	Protein	Signal
5	0	0	1	15
10	0	0	2	15
14	1	0	2	15
15	1	0	3	15
20	1	0	4	15
...	...	...	...	...