A modelling framework for Synthetic Biology

Jakob Jakobsen Boysen
Sune Mølgaard Laursen

Kongens Lyngby 2014

Summary (English)

Synthetic biology is the field of engineering new biological functions through composition and regulation of genes. The current trend of an exponential decrease in the cost of the enabling technologies indicates that sophisticated CAD tools will soon be of significant importance which will require the involvement of computer scientists and software engineers.

The overall goal of this thesis is to establish a foundation for the construction of these kinds of CAD tools in order to enable computer scientists and software engineers to more easily get engaged in the field of synthetic biology.

This thesis examines and explains how to model and simulate these gene compositions and how parallels to electronic design automation can be drawn by treating the simulated behaviour as logical lows and highs. By doing that new compositions of genes fulfilling some behavioural specifications can be proposed automatically.

The modelling framework *DTU-SB* employs many of the classical approaches to simulation as well as modelling and contributes with a novel way of performing genetic technology mapping oriented towards the practical issues that may arise in large gene compositions.

Keywords: *Synthetic biology; Gene regulated networks; Stochastic simulation; Petri net modelling; Genetic design automation; Genetic logic synthesis*

Summary (Danish)

Syntesebiologi er en ingeniørvidenskabelig tilgang til at konstruere nye biologiske funktioner gennem sammensætning og regulering af gener. Den nuværende eksponentielt faldende udvikling af prisen for at benytte støtteteknologierne indikerer, at sofistikerede CAD-værktøjer snart vil blive efterspurgt og derved kræve involvering af dataloger og softwareingeniører.

Det overordnede mål for denne afhandling er at etablere et fundament for konstruktionen af denne type CAD-værktøjer, så dataloger og softwareingeniører nemmere kan engagere sig i syntesebiologi.

Denne afhandling undersøger og redegører for, hvordan disse gensammensætninger kan modelleres og simuleres og, hvordan paralleller til elektronisk design automatisering kan drages ved at opfatte den simulerede opførsel som logisk lav og høj. Derved kan nye gensammensætninger, der opfylder en ønsket adfærd automatisk foreslås.

Modelleringsrammeværktøjet *DTU-SB* indarbejder mange af de klassiske tilgange til simulering samt modellering, og bidrager med en ny måde at udføre genetisk *technology mapping*, som er orienteret mod de praktiske problemer, der kan opstå i større gensammensætninger.

Nøgleord: *Syntesebiologi; Genregulerede netværk; Stokastisk simulering; Petri net modellering; Genetisk design automatisering; Genetisk logisk syntese*

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfilment of the requirements for acquiring a M.Sc. in Computer Science and Engineering.

This thesis serves educational purposes and is aimed at computer scientists and software engineers wishing to emerge into the field of synthetic biology.

We wish to express our sincere gratitude towards our thesis supervisor, professor Jan Madsen, for his dedicated guidance through our lengthy discussions and for providing thorough feedback within very short time-frames up to the thesis deadline.

We would also like to thank professor Chris J. Myers from University of Utah and author of the textbook *Engineering Genetic Circuits* for taking the time to answer some of our questions regarding modelling of genetic engineered devices.

Lyngby, 14-03-2014

[illegible] [illegible]

Jakob Jakobsen Boysen & Sune Mølgaard Laursen

Contents

Chapter 1

Introduction

Synthetic biology is the engineering of, possibly new, biological organisms by altering or defining its DNA. Few such examples already exists such as rice enriched with vitamin A for third world countries or tomatoes with prolonged expiration dates. Once the field of synthetic biology has matured enough to allow us to engineer more complex systems, it has the potential to contribute in multiple ways, e.g. development of personalised medicine for more efficient disease treatment or development of biosensors that can be used to detect biomolecular species in patients and characterise appropriate medical treatments. An entire research field in synthetic biology aims at modularising components of biosensors to be able to rapidly tailor new custom biosensors.

Engineered biological applications can revolutionise the efficiency of conventional devices, e.g. biological signal processing in the inner ear of a human consumes about $14\mu W$ while a computer with similar floating point performance needs $50\,W$, Sarpeshkar (2006). This serves as another motivation for utilising existing biological functions into completely new applications.

Other synthetic biology applications are already in the works or just needs commercialisation:

- Professor Jay Keasling from UC Berkeley works with commercialising solutions to synthesise *artisiminin*. Artisiminin is a critical ingredient for anti-malaria medicine and is currently harvested from the herb *wormwood*. Cultivating this herb is relatively difficult making the availability and prices very unstable which in turn can lead to fatal consequences. In 2006 Keasling's group engineered an entire strain of yeast able to synthesise artisiminin, Keasling et al. (2006), and the company Amyris Biotechnologies established by Keasling and three of his postdoctoral students now produces artisiminin on a large-scale[1].

- Another team of researchers from UC Berkeley have modified the bacteria *E. Coli* to automatically synthesise bio-diesel from sugar found in crops, Howard et al. (2013). Currently this is an infeasible process, requiring more fuel from harvest and transportation of the crops than what is synthesised, but the team have dedicated the next 3-5 years on improving the yield[2].
- *The Arsenic Biosensor Collaboration*[3] is a team composed of three previous *iGEM* winner teams that works on creating bacterias that are triggered by the presence of arsenic to emit pigment visible to the human eye which in turn can be used to create simple and cheap water quality tests for use in third world countries.

Many additional applications are surveyed in Ahmad S. Khalil (2010) and Baldwin et al. (2012).

1.1 Background

The first enabling technologies that allowed sequencing, synthesis and merging of DNA were discovered in the 1970s and were initially quite unreliable and costly which for long have been the determining factor of realisable complexity. Due to continuous technological improvements the cost is now outpacing Moore's law while the reliability is also improving, see Fig. 1.1.

This turns the attention to how these complex systems can be designed without having to manually define the base-pair sequence of DNA strings through trial and error, and instead focus on the desired behaviour of these systems.

Regulatory genes control the synthesis of proteins on the basis of other proteins being present which conceptually is similar to the behaviour of an electric transistor. This similarity has already been used to design the necessary logic gates

[1] http://www.amyris.com/Products/176/Artemisinin
[2] http://www.bbc.com/news/science-environment-22253746
[3] http://arsenicbiosensor.org/

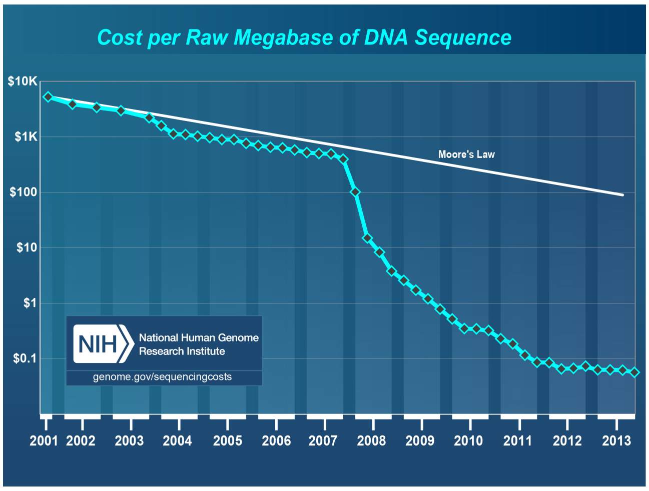


Figure 1.1: Cost of DNA sequencing vs. Moore's law. DNA synthesis follows a similar pattern, (Baldwin et al., 2012, p. 45). *Figure from* KA (2014).

to – in theory – mimic the complexity of any electronic circuit. For digital electronics many layers of abstractions have already been developed, e.g. CAD tools for designing hardware as well as software that runs on top of these electronic circuits. The software is typically compiled to binary code from programming languages specified at high abstraction levels, so instead of manually defining functions from wires and transistors we can simply write a program that do not prerequisite any knowledge about electrical engineering and are many times faster to deliver.

1.2 Problem and contributions

There is still many difficulties involved in treating regulatory genes as simple digital logic which limits what is currently realisable to a very few and very simple systems. These difficulties are very dynamic and arise from environmental changes such as temperature, crosstalk from nearby genes, type of cell and output concentration levels. In order to overcome these challenges we need to be able to predict the outcome of specific designs, or more precisely we need an accurate and efficient model able to account for these factors.

Similar to digital electronics, in synthetic biology CAD tools allowing us to create biological systems from abstract high-level specifications are naturally a necessity for effectively designing new, complex biological systems. In this thesis we will first investigate synthetic biology with focus on being able to explain the necessary theories and processes involved to identify the open challenges in the field, and from that propose a modelling technique for modelling of genetic devices. On the basis of this we will propose a modular and extensible modelling framework that easily can serve as test-bench for biologists to develop new improved models. Further we will propose how these models can be used to perform *genetic design automation* (GDA) where entire designs can be created from simple behavioural descriptions.

This thesis also serves educational purposes and tries to ease the transition for computer scientist and software engineers to the field of synthetic biology.

The code repository and documentation for the modelling framework *DTU-SB* can be found at `https://bitbucket.org/jboysen/dtu-sb` and `http://jboysen.github.io/dtu-sb-docs` respectively.

1.3 Work process

The reason for creating our own framework instead of altering an existing was to gain a broader perspective by delving into all the details to better identify interesting areas to investigate further. Therefore we established an initial plan of carrying out the thesis with two milestones:

1. Establish the required knowledge of biology, possible modelling techniques and simulation algorithms to be able to create a framework for simulation of genetic devices.

2. On the basis of the insight gained by creating the framework, identify an area to investigate further.

The first milestone was planned to take 2/3 of the available time to complete and the second milestone the remaining 1/3. The work carried out in chapters 2-7 corresponds to the first milestone and chapter 8 to the second.

Although we have worked in very close collaboration in researching, implementation and writing this thesis, the main responsibility of the chapters have been divided as follows:

Jakob *Biology* (Ch. 2), *Modelling* (Ch. 6) and *Implementation* (Ch. 7).

Sune *Engineering Biology* (Ch. 3), *Petri Nets* (Ch. 4) and *Quantitative Analysis* (Ch. 5).

The remaining chapters, including *Genetic Logic Synthesis* (Ch. 8), have been produced in complete cooperation.

1.4 Reading guide

In Ch. 2 the basic concepts of gene expression including *DNA*, *RNA*, *transcription*, *translation* and the *genetic parts* will be explained. In Ch. 3 the foundational technologies used for engineering DNA will be presented and convenient abstraction levels will be established. In Ch. 4 the fundamentals of the petri net – with focus on stochastic petri nets – will be described and it will be motivated how petri nets can be used as modelling language for modelling of genetic devices. In Ch. 5 it will be explained how these petri nets can be analysed using stochastic simulation algorithms and in Ch. 6 it will be shown how gene expression and genetic devices can be modelled with stochastic petri nets. In Ch. 7 the implementation details of the *DTU-SB Framework* implementing the theories from the preceding chapters will be explained. Finally in Ch. 8 we will use the same theories and show how new biological devices can be synthesised by specifying behaviour by simple truth-tables.

The chapters 2, 3, 4 and 5 can be read independently or alternatively skipped if the reader already has knowledge about these topics. The remaining chapters prerequisites a foundation as established in these preliminary chapters.

This thesis includes a glossary where the definition of some less common terms can be looked up. The first use of such term in each chapter is written with the typography as in e.g. "*Brownian dynamics*" and links directly to the glossary in the electronic version. Likewise all abbreviated terms are written in their full-form in italics with the abbreviation in parenthesises at their first use in each chapter.

Chapter 2

Biology

This chapter serves as an introduction to concepts not familiar by computer scientists. We will go over the basics of DNA and form the foundation for being able to understand the remaining chapters. In the following sections we assume the reader has basic knowledge about chemistry and biology, i.e. corresponding to high school level.

This chapter will evolve around the *Central Dogma of Molecular Biology* stated by Francis Crick in 1958 and later reformulated in Crick (1970). The dogma is summarised in Fig. 2.1, and shows how sequential information stored in the DNA passed into the protein cannot escape again.

The terms *DNA*, *RNA* and *protein* will be explained and we will go in depth with how information is stored in DNA, what the information is, how this information flows in biological systems and how it is important in the process of protein generation. Furthermore we will go through what a gene is and how genes can produce protein.

The cell, referred to in Fig. 2.1, is considered the smallest piece of life that exists, inside this the information store is DNA and its function is carried out by proteins, e.g. the protein insulin helps metabolise food. There are two types of cells: *prokaryotic*, which can be found in bacteria and *eukaryotic*, which can be found in e.g. animals and plants. In 1997 the complete DNA sequence for

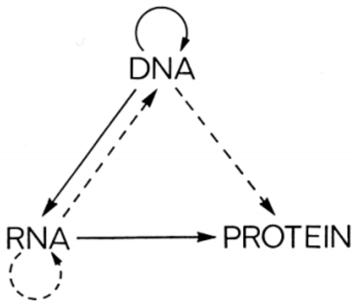


Figure 2.1: The *Central Dogma of Molecular Biology.* The diagram shows how information can flow in the cell. The circular arrow on the DNA shows *DNA replication.* We will focus on the solid arrows, so-called general transfers, which can occur in all cells. The dashed arrows refer to special transfers which do not occur in most cells. *Figure from* Crick (1970).

the bacteria *Escherichia coli* (E. coli), widely used in synthetic biology, was published in Blattner et al. (1997). In general there is a great understanding of the prokaryotic cell and it is used as host of genetic devices in many applications and experiments in synthetic biology, which is why we in this thesis refer to the prokaryotic cell when we refer to *the cell.*

Several sources of literature have been used to compose this chapter: (Baldwin et al., 2012, Ch. 1), Medicine (2013), Institute (2013), Gregory (2013), Bryce and Pacini (1998) and (Karp, 2009, Ch. 10-11).

2.1 DNA

DNA is short for the molecule *deoxyribonucleic acid* that stores information in biological systems. On its own DNA cannot do anything but in the protein generation process DNA plays a vital role as information storage. DNA is one of three important *macromolecules* — the others being protein and RNA, of which RNA also is a nucleic acid just like DNA. Macromolecules are just molecules with a high relative molecular mass composed of many molecules with low relative molecular mass.

DNA is composed of two complementary strands; each being a chain with links of *nucleotides*: each nucleotide contains a phosphate group, a sugar group and a nitrogenous base. The chain of nucleotides is linked together by the phosphate and the sugar from two nucleotides. The important part here is the nitrogenous base, as the sugar and the phosphate are always the same. In DNA there are four different nitrogenous bases: *adenine (A), thymine (T), guanine (G)* and *cytosine (C)*. The sequence, or order, of the nucleotides determines the information stored in the DNA, later we shall see how this information can be used.

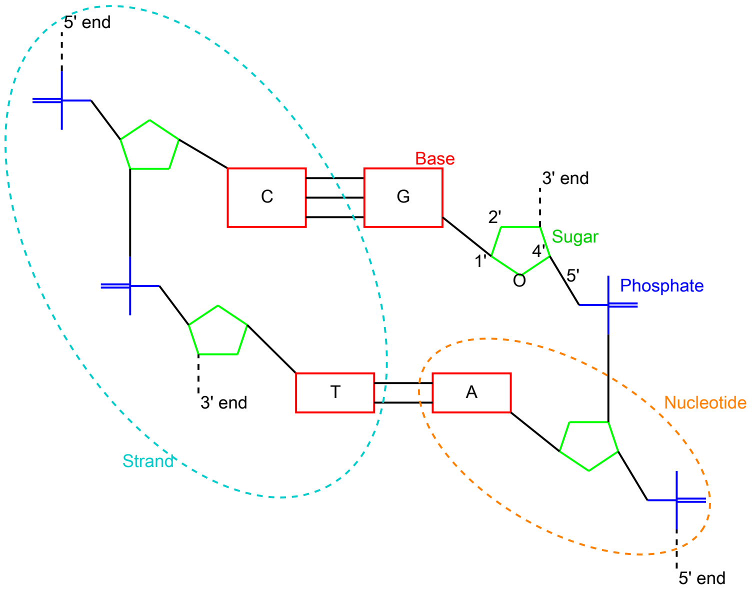


Figure 2.2: Simplified schema of the DNA structure. Black lines are chemical bindings between bases, sugar and phosphate. The big O at the bottom of the sugar molecule is Oxygen, and the numbers refer to the carbon atoms in the sugar molecule.

The two strands are twisted in a double-helix structure and are tied together by *hydrogen bonds* between the bases on each strand. Two bases held together by hydrogen bonds are called *base pairs*. There is only two possible combinations of pairs, which is an important feature of DNA; A can only pair with T, and C can only pair with G and vice versa. This is crucial as it means a cell always has two copies of the information sequence stored in the DNA, which in turn means that if one strand is damaged, that strand can be repaired directly by the other strand just by filling in the missing base in each pair. The reason only these base pairs can exist is because of the hydrogen bonds between the bases; between A and T there are two bonds, and between C and G there are three bonds. Informally speaking the DNA structure is like a twisted ladder with the sides being the sugar and phosphate and the base pairs being the rungs.

As we can see in Fig. 2.2 the two strands run in opposite directions and the ends of the strands are named 3' and 5' referring to the 3rd and 5th carbon atom in the sugar molecule facing towards the ends. The sugar molecule contains 5 carbon atoms in a chain, where the numbering starts at 1', where the nitrogenous bases bind. The numbering of 3', 4' and 5' can be a bit confusing but essentially the carbon in the CH_2OH-group (not shown in Fig. 2.2) attached to the 4' carbon is the 5' carbon. The 3' and 5' ends are used when we talk about which direction to read the strands, more on that later.

2.2 Genes

With a basic understanding of what the DNA molecule is composed of and how it is structured, we now turn to genes. A gene is a stretch of the DNA molecule including *regulatory elements*, which means that DNA contains many genes. In the previous section we saw that DNA is the information store and cannot do anything on its own. The gene produces a functional gene product using the information from the DNA in a process called *gene expression*. The gene product is either RNA or protein. We call genes that generate proteins for protein-coding genes, these are the genes we will be focusing on in this thesis.

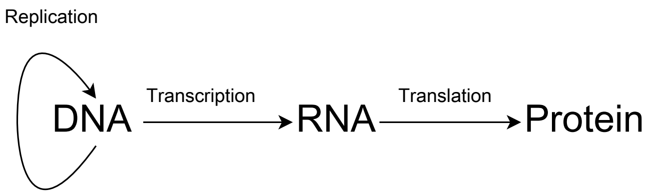


Figure 2.3: DNA replication and the steps in gene expression. Refering to Fig. 2.1 on page 8, this is the same figure without the dashed arrows.

Fig. 2.3 shows the steps *transcription* and *translation* in gene expression which will be explained in the next section. The *replication* step is a process which only DNA can undertake when the cell containing the genes is divided. In this thesis we will not go into detail with DNA replication as the process is somewhat similar to gene expression explained below.

The most simple gene consists of the parts *promoter*, *ribosome binding site (RBS)*, *protein coding sequence (PCS)* and *terminator*. The parts are placed on the stretch of the DNA molecule sequentially in the order just presented, e.g. the RBS is often a six to seven base long nucleotide sequence placed about eight bases upstream[1] from the PCS. These parts will be explained and mentioned in the following sections.

2.3 Gene expression

Protein-coding genes can translate its DNA to protein by first transcribing the DNA sequence into an RNA molecule and after that translating this RNA molecule into amino acids which in turn is what proteins are composed of.

In gene expression the RNA (ribonucleic acid) molecule plays an important role in several parts of the process. There are different kinds of RNA molecules of which we will mention *mRNA (messenger)*, *sRNA (small)* and *tRNA (transfer)*. mRNA contains genetic information just like DNA; in fact it is a copy of one of the DNA strands in the gene and is recognized by a ribosome that translates it into amino acids. sRNA is, as the name indicates, small non-coding RNA molecules produced naturally by sRNA-encoding genes in E. coli, Hershberg et al. (2003). sRNA can be used to regulate gene expression. tRNA transports the amino acids to the ribosomes during the protein synthesis.

The RNA molecule is, just like DNA, a chain of nucleotides. The main differences between RNA and DNA is that the sugar molecule in the nucloetides in RNA is *ribose* whereas in DNA it is *deoxyribose*, in RNA the nucleotide thymine (T) is replaced with uracil (U) which also binds with adenine (A) and at last RNA only has one strand whereas DNA has two.

2.3.1 Transcription

In the transcription process an mRNA molecule is synthesised from the DNA. The DNA strand with the same sequence of nucleotides as the mRNA strand (but with T replaced by U) is called the *coding strand*, the opposite DNA strand is called the *template strand*.

1. The process is initiated by proteins called *sigma factors* binding to the promoter in the DNA. There are several sigma factors, the specific sigma factors used depend on the specific gene and the surrounding environment.

[1]Here upstream just means before.

2. The promoter will identify the strands and the direction to copy in, afterwards the *enzyme RNA polymerase* will bind to the promoter. RNA polymerase always works in the direction from the 5' end to the 3' end, thus this is the direction of synthesis. Polymerase is an enzyme that creates a chain of molecules, e.g. RNA polymerase will create the mRNA molecule which consists of many nucleotides.

3. RNA polymerase is now bound to the template strand and moves towards the 3' end of the coding strand while it adds complementary RNA nucleotides to the template strand. Starting at the promoter site the DNA will unwind by breaking the hydrogen bonds between the base pairs on each strand. This can involve many RNA polymerases at once, meaning that several mRNA molecules can be synthesised at once, where the first molecule is called the primary mRNA.

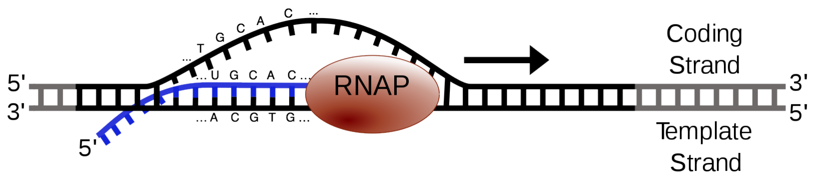


Figure 2.4: Transcription. RNAP is the RNA polymerase that unwinds the DNA strands (black), creates the mRNA (blue) from the template strand and detaches the mRNA again. *Figure from* Forluvoft (2007).

4. The RNA polymerase will break the hydrogen bonds between the new complementary nucleotides to the nucleotides on the template strand and they will form an mRNA strand held together by sugar and phosphate, just like DNA.

5. The transcription will stop when the RNA polymerase reaches a characteristic sequence of nucleotides on the template strand, also known as the termination sequence, and shortly after the mRNA is detached completely from the template strand, the strands of the DNA rewinds to its usual structure again.

The transcription process described above is depicted in Fig. 2.4. The next step in gene expression is translation of the mRNA to protein, but before we explain that, we will in the next section explain how the chain of nucleotides in the mRNA codes for amino acids.

2.3.2 The genetic code

The information in the mRNA molecule is determined by the sequence of the nucleotides, in pairs of three the nucleotides codes for an *amino acid* which is the building blocks for proteins. A sequence of three nucleotides is called a *codon*. Consequently there must be three reading frames on the mRNA strand, i.e. if the starting point on the mRNA has not been identified yet, each nucleotide in the mRNA can be used in three different codons, see Fig. 2.5. Because of this we need a way to determine the correct first codon on the mRNA strand.

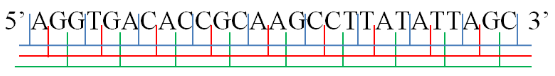


Figure 2.5: Three (blue, red and green) reading frames of the mRNA. E.g. the third nucleotide, G, can be used in all of the three frames. *Figure from* Ákos (2011).

The first amino acid on each protein is Methionine corresponding to the codon AUG (in mRNA, in DNA this corresponds to the codon ATG), this means that the frames to read can be determined by looking for the AUG codon. How this process carries on will be explained in the next section.

As mentioned, proteins are composed of amino acids of which there are 20 different kinds. There are four different nucleotides in mRNA, namely A, C, G and U, this gives $4^3 = 64$ different codons, which in turn means that several codons code for the same amino acid. Fig. 2.6 shows all amino acids decoded from codons including the three stop-codons UGA, UAG and UAA which is used in the translation process.

2.3.3 Translation

Translation is the last step in gene expression. Here the sequence of nucleotides in the mRNA is translated into amino acids using the genetic code described above and tRNA transporting the amino acids. The process takes place in the *ribosome*, which is a so-called molecular machinery that catalyses the creation of the chain of amino acids (this chain is also called a polypeptide chain) that forms a protein. The ribosome binds to the gene at the RBS and consists of two subunits: a small subunit reading the mRNA and a large subunit linking the amino acids together to the polypeptide chain. The large subunit consists three sites: E, P and A, each containing a tRNA. Each tRNA contains an *anticodon* matching a codon on the mRNA and one amino acid associated with

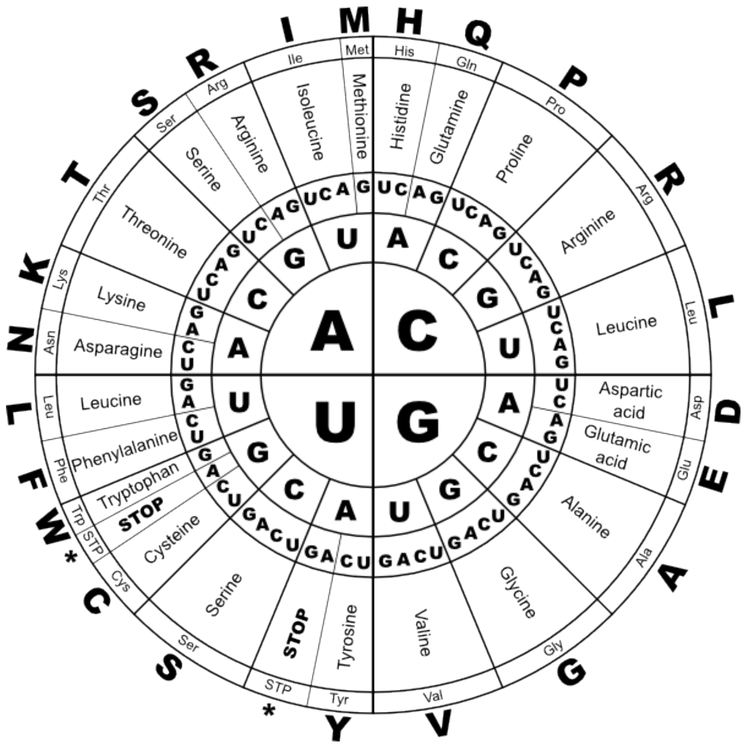


Figure 2.6: The genetic code. The diagram should be read from the center and towards the edge of the circle, where the amino acid coded from a codon can be read. E.g. the codon CUA codes for the amino acid Leucine. *Figure from* Alves (2010).

the anticodon. E.g. the anticodon UAC matches the codon AUG which in turn matches the amino acid Methionine.

1. The process is initiated by the small ribosomal subunit binding to tRNA with the amino acid Methionine and finding the so-called *Shine-Dalgarno sequence* on the 5' end on the mRNA. This sequence, AGGAGG, is usually located 8 nucleotides upstream of the correct start codon AUG. The small ribosomal unit now binds to the mRNA and the large ribosomal subunit binds to the small so that the tRNA is located in the P site of the large subunit.

2. A tRNA matching the codon located at the A site binds to the ribosome, while the amino acids attached to the tRNA at the P and A site will create a link in the polypeptide chain. The binding between the amino acid and

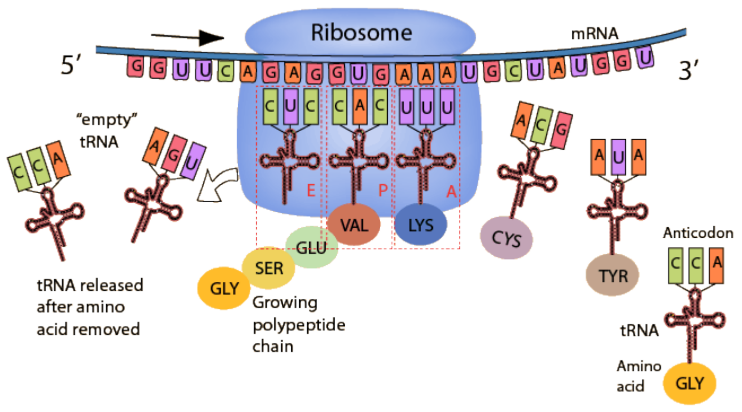


Figure 2.7: Translation. The ribosome part above the mRNA strand is the small subunit and the part below is the large subunit. Minor details are omitted, e.g. here the polypeptide chain does not start with Methionine. *Figure from* Nave (2013).

the tRNA located at the P site now breaks and the ribosome will move one codon towards the 3' end of the mRNA. The tRNA located in the P site will move to the E site and leave the ribosome shortly after and the tRNA in the A site will move to the P site. Now a new matching tRNA will enter the A site and the previous process will be repeated.

3. When the ribosome has encountered one of the three stop codons no matching tRNA can be found and proteins called *release factors* will enter the ribosome causing it to detach from the mRNA and the polypeptide chain to detach from the ribosome.

After this process the chain of amino acids will now fold into a protein. The process described above is carried out by several ribosomes, thus several copies of the same gene are generated. In the prokaryotic cell mRNA has a relatively short life time, which is why the translation process takes place at the same time the mRNA is being transcribed. After the translation has finished the mRNA dissolves and the nucleotides in the mRNA are ready to be used in new gene expressions.

In the processes transcription and translation there are some inherent delay, e.g. when the tRNAs are moving into the correct positions in the cell, these delays can cause random fluctuations of how much protein is generated.

2.3.4 Decay

Both mRNA and protein will decay over time, this means that protein will only be produced as long as there is mRNA available, which only happens when transcription is enabled. In the next section we shall see how transcription can be blocked. Furthermore decay of protein also means protein must be produced continuously if it is required at all times.

The decay rate expresses lifetime or the stability of a product. The lifetime of mRNA in prokaryote is relatively short, varying from a few seconds to about an hour, Rauhut and Klug (1999). We will not go into details with how decay of mRNA happens, but just note that the decay of mRNA plays a very important role in the regulation of gene expression.

2.4 Gene regulation

One very important ability of genes is their ability to regulate, i.e. to turn on and off as needed. Cancer is the result of a erroneous *always on*-regulation, Gregory (2013). The regulation happens by *operators* close to the promoter (the regulatory element briefly mentioned earlier) and occurs on basis of certain protein concentrations. For instance in E. coli enzymes for converting lactose to glucose are only being synthesised if there is not sufficient available glucose while lactose is present. The operator is located downstream of the promoter on the DNA.

When *repressor protein* is bound to the operator, the operator effectively regulates gene expression by physically denying RNA polymerase to bind to the promoter on the gene. There are two types of operators:

Inducible By default the repressor protein is bound to the operator, i.e. transcription is repressed. To unbind the repressor protein from the operator an *inducer* protein must bind to the repressor. See illustration in Fig. 2.8.

Repressible By default the repressor protein is not bound to the operator, i.e. transcription is active. When a *co-repressor* is present it becomes active and binds the operator. See Fig. 2.9

Inducers and co-repressors are practically equivalent; it is the repressor protein that determines whether it is inducible or repressible. It is important to note that these repressor proteins are specific in the sense that they only bind to some specific operators and can only be repressed or induced by some specific inducer or co-repressors. This is illustrated in Fig. 2.8 and 2.9 by their puzzle-shapes.

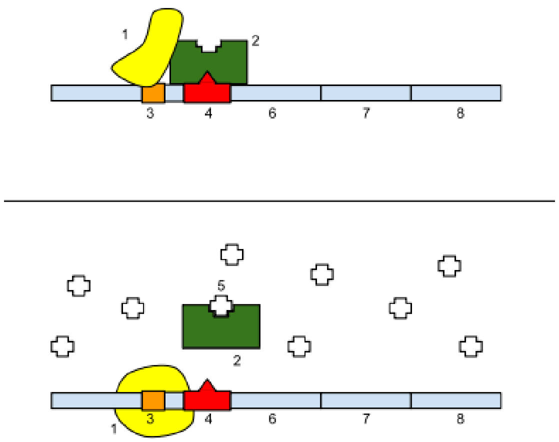


Figure 2.8: **1**: RNA Polymerase, **2**: Inducible repressor, **3**: Promoter, **4**: Operator, **5**: Inducer, **6,7,8**: PCS.
Top) The repressor prevents the transcription process by blocking the promoter.
Bottom) An inducer becomes present, so the repressor unbinds from the operator to bind with the inducer. Transcription can now occur. *Figure altered from RAJU.*

The time from releasing a regulatory protein, e.g. an inducer, to a complete switch can be observed, is typically in the order of minutes.

Some secondary regulation can occur outside the operator sequence due to external changes. For instance the rate of transcription can also be influenced by other proteins and temperature changes but is usually not as determining as regulating the operator directly.

The regulation described above is usually referred to as *transcriptional regulation*, another type of regulation is referred to as *translational regulation*. One example of this is sRNA base-pairing with mRNA thus influencing translation or mRNA stability, effectively repressing gene expression, Shimoni et al. (2007).

As the behaviour of gene regulation is like that of an electric transistor, logic gates can also be created by genes hence, in theory, arbitrarily complex biological systems can be created. This leads us to the next chapter on engineering.

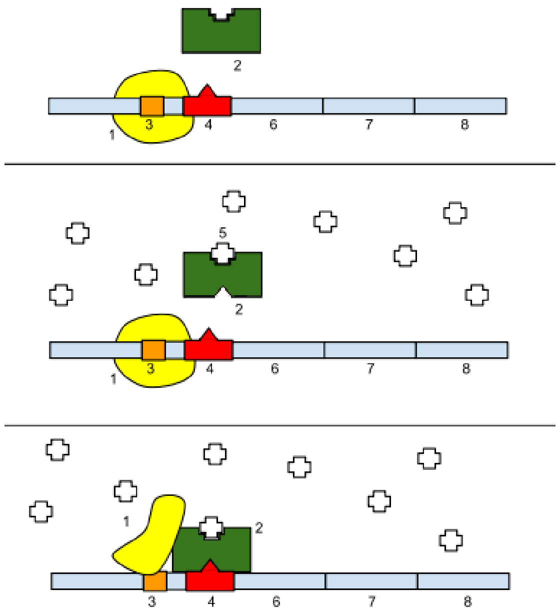


Figure 2.9: **1**: RNA Polymerase, **2**: Repressible repressor, **3**: Promoter, **4**: Operator, **5**: Co-repressor, **6,7,8**: PCS.
Top) The repressor is inactive but present.
Middle) A co-repressor becomes present which binds to the repressor thus enables binding to the operator.
Bottom) The repressor binds to the operator and blocks further transcription. *Figure altered from RAJU.*

Chapter 3

Engineering Biology

This chapter describes the foundational technologies for engineering DNA strands, how computer-tools can aid the design process, through which simplifying abstractions biological systems can be regarded and finally some of the difficulties in engineering biological systems will be discussed.

Several sources of literature have been used to compose this chapter: (Baldwin et al., 2012, Ch. 2-3,5), Beal et al. (2012), Beal et al. (2011), Densmore and Hassoun (2012) and Pedersen and Phillips (2009).

3.1 Enabling technologies

The technologies in this section are the technologies that allow us to read, write and combine DNA fragments. Some of these are quite complex and therefore only briefly described while referring to more comprehensive literature on the matter.

3.1.1 DNA sequencing

Sequencing is the process of obtaining the base pair representation of a given DNA strand. Many DNA sequencing methods exist, each with varying accuracy, cost, speed and read length. Advances in DNA sequencing currently receive a lot of focus as it is believed to be the foundation of (near) future disease diagnosis by sequencing the entire genome and prescribe accordingly. The widely used *Sanger sequencing* method basically works as follows:

1. Split the double stranded DNA into a template strand and a complementary strand by applying heat.
2. Put the template strand mixture in 4 different containers along with some polymerase.
3. Put some nucleotides (A, C, G and T) as well as one unique type of PCR terminating nucleotide (A, C, G or T) in each container.
4. Now the template strand will try to repair itself but will randomly get terminated by the PCR terminating nucleotide unique to that container.
5. By gel-electrolysis, which sorts the partial strands on their weight, it is now possible to identify the positions of the labeled nucleotides hence the location of all complementary nucleotides.
6. Merging the results from each of the containers now yields the complete sequence of DNA.

More details can be found in (Baldwin et al., 2012, Appendix 1) and detailed comparison of novel methods can be found in e.g. Liu et al. (2012).

3.1.2 DNA synthesis

Synthesis is the process of creating artificial DNA strands. Typically *oligonucleotide synthesis* by McBride and Caruthers (1983) is used which is a chemical process to produce short strands of 15-20 base pairs by treating nucleotides as building blocks that can be sequentially coupled in a growing order using four chemical processes for each addition. In general the majority of the synthesis methods are only able to produce small strands to avoid introducing errors why the assembly methods is of great importance to the synthesis of large strands.

3.1.3 DNA assembly

DNA assembly is the process of merging two strands. There are numerous ways to assemble DNA strands. One of the more intuitive, compelling methods is the *standard assembly method* which uses restriction enzymes to merge DNA strands as illustrated in Fig. 3.1. This method is cheap, but prone to errors, Densmore and Hassoun (2012). Some of the available assembly methods are compared in Baldwin et al. (2012).

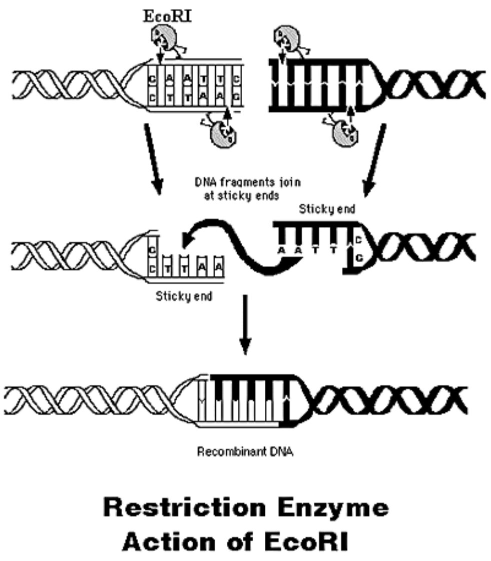

EcoRI
DNA fragments join at sticky ends
Sticky end
Sticky end
Recombinant DNA
Restriction Enzyme Action of EcoRI

Figure 3.1: The *standard assembly method.* EcoRi enzymes seek and remove the *AATTC* from 3' strands and the complementary *TTAAG* sequence from 5' strands. This forces the strands to combine. *Figure from* Excellence.

BioBrick$^{\text{TM}}$ is an interface for painless assembly of biological parts. It works by having some universal defined DNA sequence appended to both ends of the part sequence with the ability of forming stable bonds with other parts that implement the same interface. The design along with the behavioural characteristics of these parts are stored and made available in rapidly increasing databases, e.g. the *Registry of Standard Biological Parts*[1]. A goal from the foundation behind the BioBrick, the *BioBricks Foundation*, is to make it possible to engineer entire organisms just from these parts.

[1]http://parts.igem.org

3.2 Tool-chain

Effectively engineering biological systems requires extensive tool-chains able to model and simulate the complex nature of biology at suitable abstractions. Typical tool-chains have the structure illustrated in Fig. 3.2.

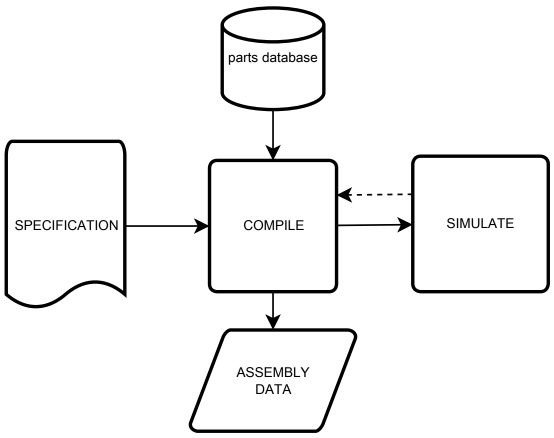


Figure 3.2: A typical tool-chain for synthetic biology. A high-level specification of some biological system is compiled into a more suitable internal representation using an existing parts-database. This internal representation can either be emitted as assembly instructions or simply be simulated in order to determine how well the design works. These results can then be used to refine the compilation by selecting alternative parts until the desired design requirements are met.

The actual synthesis and assembly, whether it being automated or manual, is still quite costly in time and money why the simulation process is of great importance in order to eliminate weak design candidates before *wet-lab* experiments. Naturally a successful simulation is no guarantee for the design actually being realisable in a wet-lab experiment, why if possible several design alternatives are often emitted.

3.3 Abstractions

In order to support the high-level descriptions for a biological compiler, we need to break up and simplify the problem into biological defining functions.

Similar to designing circuits in a hardware description language, to simplify the process and increase the achievable complexity one usually do not build new circuits entirely from scratch, but rather uses existing modules such as *adders*, *io-modules*, etc. From the bottom and up we have *parts*, *devices* and *systems*. BioBricks implement the same level of abstractions.

Parts are the smallest biological building block found in genes and have one of the following four biological functions:

Promoter Initialises the transcription and instructs it to start. Several operators can be chained to the promoter in order to control the rate of transcription from concentrations of nearby DNA binding proteins in complex manners. For the sake of simplicity the chain of operators is considered as a part of the promoter.

Ribosome Binding Site (RBS) Gathers and initiates ribosomes for DNA translation.

Protein Coding Sequence (PCS) The strand from which the ribosomes should translate protein.

Terminator Indicates the end of the gene expression.

Some of the part names above should be recognised from earlier; the promoter is found upstream of the other parts on the DNA, RBS is where the the gene expression occurs, PCS is the codons coding for the protein and the terminator is just the end of the transcription.

In many cases several alternative base pair representations exist of similar parts, but with different resistances and susceptibility to different DNA binding proteins and/or external environments, so one of the engineering difficulties is also to select the correct encoding of the different parts to avoid interference while maintaining an acceptable degree of performance.

The graphical notation using *Pigeoncad* by Bhatia and Densmore (2013) has been adapted and can be seen in Fig. 3.3. This notation is greatly inspired by the *SBOL Visual* standard from Quinn et al. (2013).

Devices, sometimes referred to as genetic devices, are a combination of parts with some biological function. The generic feedback device in Fig. 3.4 ensures a more or less constant concentration of a given protein, i.e. when the concentration of produced protein is high the promoter is repressed hence stops further transcription until the concentration has decayed enough to continue.

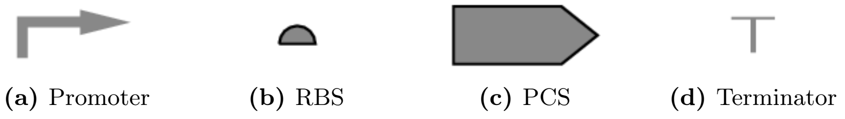


Figure 3.3: Graphical notation of parts

On the device in Fig. 3.4 there is an arc from the PCS to the promoter with a vertical bar at the end. This arc can be seen as the protein produced repressing the promoter, i.e. the arc represents a regulation. An arc with an arrow from a PCS to a promoter means that the protein produced from the PCS activates the promoter thereby allowing gene expression. No incoming arcs to a promoter means it will constantly initiate transcription, these promoters are called constitutive promoters. Several of these arcs can point to a promoter, meaning that several proteins can initiate the gene expression.

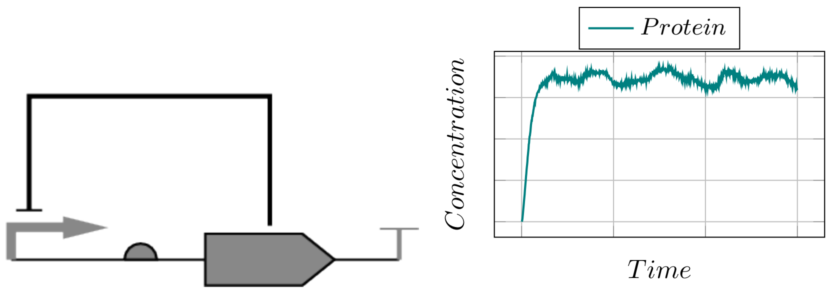


Figure 3.4: Generic feedback device and its expected behaviour.

Systems are combined of several devices in order to achieve a more complex behaviour such as the generic NOR-gate system in Fig. 3.5.

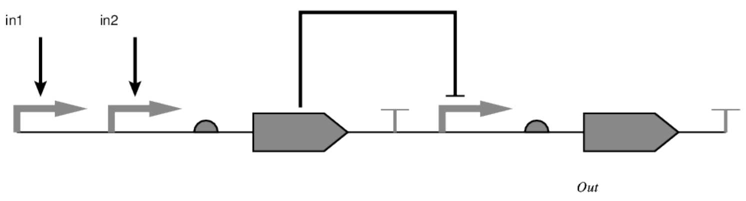


Figure 3.5: Logic NOR-gate system. Any of the two input proteins *in1* or *in2* may initiate transcription which is inverted giving the desired NOR-behaviour of the output protein *Out*.

3.4 Discussion

Promoters are usually very context sensitive in the sense that temperature, nearby proteins and other environmental changes easily can influence the transcription of the coding sequence possibly leading to domino effects that can influence a larger system. Therefore even wet-lab experiments are quite unpredictable and imposes huge future challenges for future *real-world* realisations.

When designing digital circuits we have the problem of cross-talk when the signal in a wire is disturbed due to electromagnetic interference from another wire. This is generally avoided by increasing the shielding and/or placing the wires more apart – something the CAD tools relatively easy can account for. But in synthetic biology we do not have these possibilities as we cannot shield reactions or guarantee that reactions will occur within a certain distance.

To reduce cross-talk in wet-lab experiments typically these simple techniques are used:

- If the host is a well-studied mechanism, such as E.coli, where all embedded protein coding sites have been identified, it is possible to create an *orthogonal system* by using parts known not to interfere with those of the host.
- Physically isolation by placing smaller quantities in lesser populated areas of the cell.

A few entirely different approaches are mentioned in Sec. 9.3 on page 110.

In designing digital electronics it is easy to reuse parts and pipe-line execution as every executional step happens within a predefined cycle defined by a global clock. A difficulty in synthetic biology is that every reaction happens at different rates depending on the specification of the parts used and this can still vary quite much due to the different external factors just described, making it very hard to pipeline systems and even guarantee a consistent output.

Chapter 4

Petri Nets

The *Petri net* (PN) was developed to illustrate and model chemical processes and was formally presented in Petri (1966). Due to its ability to describe concurrent processes in a concise way, many other scientific fields have also adopted its notation. In this chapter the required fundamentals of the PN will be described in order to motivate its use as a modelling language and intermediate representation as further described in Ch. 6 and 7 respectively. Some necessary extensions to the core PN will be introduced. Ch. 5 will show how PN with associated rate functions can be analysed. This chapter is based on Blätke et al. (2011), Heiner et al. (2008) and Petri and Reisig (2008).

4.1 Reaction equations

Consider the following chemical reactions:

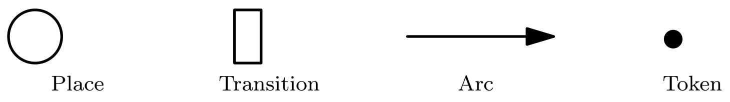
$$
\begin{aligned}
S_1 + 2S_2 &\rightarrow S_3 \\
S_1 &\rightarrow \emptyset
\end{aligned}
$$

They describe a system consisting of two concurrently enabled reactions with the behaviour that the two reactant species S_1 and S_2 in the respective quantities of 1 and 2 *can* react to form the product species S_3 as well as that species S_1 *may* decay. This notation is typically used to describe chemical equations. Below we will see how these kinds of reactions can be illustrated as PNs.

4.2 Petri net definition

A PN is given by the quadruple $PN = (P, T, f, m_0)$ where P is a set of places, T is a set of transitions, f is a function that describes a set of directed arcs by non-negative integer values $f : ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N}_0$ and finally m_0 is an initial marking of the places $m_0 : P \rightarrow \mathbb{N}_0$.

A PN is illustrated using the following elements:



Places A place p can contain any discrete amount of tokens and may have arcs pointing to any number of transitions.

Transitions A transition t may have arcs pointing to any number of places.

Arcs Arcs are used for pointing places to transitions and vice versa. The multiplicity of the edge from place p to transition t is denoted $f(p, t)$, likewise the multiplicity of (t, p) is denoted $f(t, p)$. The multiplicity of non-connected components is by definition 0.

Tokens Tokens indicate the marking of a given place. The marking of place p is denoted $m(p)$.

Let m be the current marking of the PN (initially m_0) and let $\bullet t$ denote the pre-places of transition t: the set of places that have outgoing arcs connected to t. Similarly $t\bullet$ is the post-places of transition t.

The *firing rule* describes the dynamics of a PN: a transition t is *enabled* for the marking m if all of its post-places contains sufficient tokens, i.e. if $\forall p \in \bullet t : m(p) \geq f(p, t)$, otherwise it is disabled. Thus a transition without any pre-places is always enabled.

A transition that is enabled *may* fire. If a transition t fires, the multiplicity of each incoming arc is subtracted from its corresponding pre-place and the multiplicity of each outgoing arc is added to its corresponding post-places to obtain a new current state m', i.e. when t fires then $\forall p \in P : m'(p) = m(p) - f(p,t) + f(t,p)$. This means that the amount of tokens is not necessarily preserved.

The firing of an enabled transition takes no time and happens non-deterministically as there is no timing requirements of transitions firing when enabled. Only one transition is allowed to fire at any given time instant.

A set of chemical reactions is represented as a PN by representing each molecule as a place. The reaction arrow is represented by a transition with ingoing arcs from places representing reactants and outgoing arcs to places representing products. Fig. 4.1 represents the simple reaction system from Sec. 4.1 as a PN.

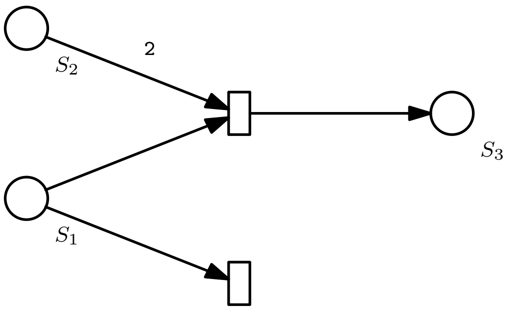


Figure 4.1: PN of two simple chemical reactions. Notice the multiplicity on the ingoing arc from S_2 to the transition.

4.2.1 Example: predator-prey

Now consider a simple predator-prey model described by the following reaction equations:

$$\begin{aligned} Prey &\to 2\ Prey \\ Prey + Predator &\to 2\ Predator \\ Predator &\to \emptyset \end{aligned}$$

That is, *prey can reproduce at will, the predators need to eat a prey in order to reproduce* and *the predators die.* Notice that it is assumed that preys do not die unless eaten.

These equations translate into the PN illustrated in Fig. 4.2 by using reactions as transitions and reactants and products as places. Fig. 4.2 assumes an initial state of 2 Prey and 1 Predator.

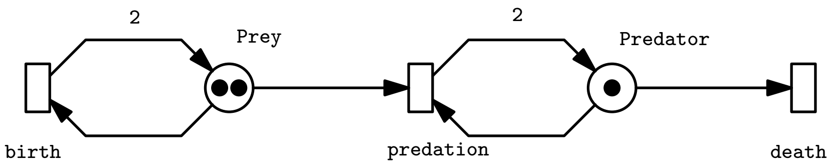


Figure 4.2: The predator-prey model as a PN.

The PN is formally described as:

$$
\begin{aligned}
P &= \{\texttt{Prey}, \texttt{Predator}\} \\
T &= \{\texttt{birth}, \texttt{predation}, \texttt{death}\} \\
f &= \{(\texttt{Prey}, \texttt{birth}) = 1, \\
&\quad (\texttt{Prey}, \texttt{predation}) = 1, \\
&\quad (\texttt{Predator}, \texttt{predation}) = 1, \\
&\quad (\texttt{Predator}, \texttt{death}) = 1, \\
&\quad (\texttt{birth}, \texttt{Prey}) = 2, \\
&\quad (\texttt{predation}, \texttt{Predator}) = 2\} \\
m &= m_0 = \{\texttt{Prey} = 2, \texttt{Predator} = 1\}
\end{aligned}
$$

Initially all of the transitions in T are enabled. Assuming that `predation` fires gives rise to the state $m_1 = \{\texttt{Prey} = 1, \texttt{Predator} = 2\}$ still leaving all transitions enabled. Assuming `predation` fires again gives $m_2 = \{\texttt{Prey} = 0, \texttt{Predator} = 3\}$ leaving only `death` enabled. Now `death` fires 3 times yielding $m_5 = \{\texttt{Prey} = 0, \texttt{Predator} = 0\}$.

4.2.2 Extensions

In order to extend the expressiveness some typical extensions to the core PN language are introduced. With the introduction of the *inhibitor arc* PNs becomes *turing complete*, Zaitsev (2013), why any other extensions introduced can be regarded as macros of this set and will not be formally introduced.

4.2.2.1 Inhibitor arc

The inhibitor arc disables transition t if any of its pre-places are connected with an inhibitor arc to t while having sufficient tokens. With this addition we now have: $PN = (P, T, f, i, m_0)$, where i is a set of directed inhibitor arcs by a non-negative integer value $i : (P \times T) \to \mathbb{N}_0$. Let $\star t$ denote the pre-places to t from i.

Transition t is now only enabled if $\forall p \in \bullet t : m(p) \geq f(p,t)$ and $\forall p \in \star t : m(p) < i(p,t)$ as illustrated in Fig. 4.3.

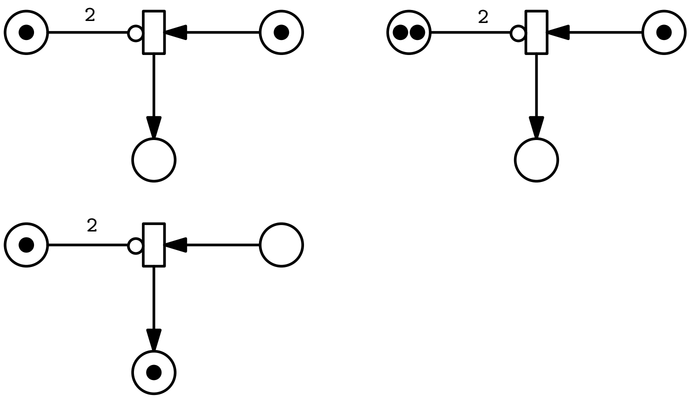


Figure 4.3: Behaviour of the introduced inhibitor arc.
Top left) The transition is enabled.
Bottom left) The transition taken.
Right) Transition is disabled.

4.2.2.2 Read arc

The read arc can fire transition t (if enabled) without removing any of its pre-place tokens. Fig. 4.4 illustrates its graphical representation and how it can be regarded as just a macro of the already introduced components.

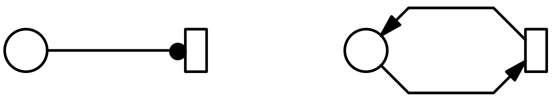


Figure 4.4: *Left)* The read arc. *Right)* Semantically equivalent.

4.3 Rates of firing

For the PNs to be of any use in modelling we need to specify firing rates for the transitions. A firing rate λ_t for transition t can either be a constant value or some expression that depends on the individual markings of $\bullet t$. Further, the type of firing rate instructs how to analyse the PN, where the type can be either continuous or stochastic giving rise to *continues petri nets* (CPN) and *stochastic petri nets* (SPN) respectively. Both types are formally introduced and discussed in Ch. 5. The visual representation of such PNs remains unchanged but for reaction equations the firing rate is written above the reaction arrow as follows:

$$S_1 + S_2 \xrightarrow{\lambda} S_3$$

With knowledge of such rates the *modifier arc* is introduced. It has no influence on transitions and places, thus transitions with only ingoing modifier arcs are always enabled and do not consume any tokens upon firing. The modifier arc is used in models to alter the reaction rates and is often used to model inhibition using inverse rate functions as explained in Sec. 6.2 on page 50.

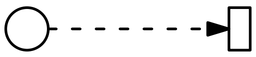


Figure 4.5: The modifier arc

4.4 Discussion

The expressiveness of the inhibitor arc becomes greatly limited with the introduction of rates and is most likely to be replaced by network constructs or *modifier arcs* with inverse rate functions that are in better cohesion with the kinetics. We have included the inhibitor arc to inform about its behaviour as it is very often encountered in literature. Some PN editors such as *Snoopy*, Heiner et al. (2012b) do not support the inhibitor arcs during exportation to SBML and only uses them internally for simulation using a custom behavioural definition of the arc.

The reasons for employing PNs instead of just using reaction equations are two-fold:

- It makes the creation of models more intuitive as one easily can see how the species are connected as well as allowing animating token movements of simulation traces for debugging of designs.

- It is highly analysable where for example *liveness*, *reachability* and *boundedness* easily can be assessed using available third-party tools. These types of analysis can be used as foundation for genetic compiler optimisations.

Chapter 5

Quantitative Analysis

We turn the attention on how to analyse *petri nets* (PNs) to give predictions on how the amount of involved species will evolve over time. Manual or automated inspection of such time-series can then asses the quality of the given model compared to some desired specifications such as behaviour, latency and resulting strength of protein concentrations. Much like voltages in electric circuit design.

In Sec. 4.3 we mentioned two types of PN with associated rate functions, namely *continuous petri net* (CPN) and *stochastic petri net* (SPN). These types are nothing more than instructions on how they should be analysed; CPNs should be analysed using continuous and deterministic approaches and SPNs should be analysed using stochastic approaches. Examples of both approaches are presented in this chapter.

Such analysis methods are not exclusive to the PN structure and are therefore presented using the following notation: a system consists of N chemical species $S_1, ..., S_N$ interacting through M reaction channels $R_1, ..., R_M$ with associated kinetic functions $\lambda_1, ..., \lambda_M$ as well as the state-change vectors $v_1, ..., v_M$ where $v_j = v_{1j}, ..., v_{Nj}$ and v_{ij} denotes the change in species S_i when firing reaction R_j. $X(t)$ is the state at time t and denotes the individual markings of the N species $X(t) = \{X_1(t), ..., X_N(t)\}$.

Biochemical systems are often composed of relatively few molecules from a few reactant species, Gillespie (1977), so in order to understand the need for stochastic approaches it is motivated why the deterministic and continuous solution alone often are insufficient for models having these characteristics. Solutions obtained using these two approaches are fundamentally different in the sense that the deterministic approach gives a single solution whereas a stochastic simulation gives rise to single random trajectory. The later is usually repeated in order to obtain a representative average.

Both methods assumes that the systems are well stirred, or spatially homogeneous, as it is otherwise needed to track the position and speed of each molecule which quickly can become intractable. Some approaches removing this spatial constraint have been developed and are discussed in the end of this chapter along with other more novel approaches.

Due to the desire to solve increasingly large and detailed biochemical systems we also present run-time analysis of the presented algorithms.

5.1 Law of mass action

Relating to PNs and letting the transition μ relate to reaction R_μ, often the rate function of reaction R_μ is somehow dependent on the markings of the reactant species $\bullet\mu$. In the predator-prey example from Sec. 4.2.1, such functions can be used to describe the following logical deduced population characteristics:

1. The rate of any prey to reproduce increases with the amount of prey present.
2. The rate of any predator to reproduce, by consuming a prey in the process, increases with the number of prey and predators present.
3. The rate of any predator to die increases with the number of predators present.

The kinetics of R_μ denoted λ_μ *may* be determined using *the law of mass action* by Waage and Gulberg (1864) which, although developed for chemistry, generalises these types of observations and is applicable to many types of population dynamics. The law basically states that for R_μ, the rate function λ_μ can be determined as:

$$\lambda_\mu = k_\mu \prod_{S_i \in \bullet\mu} X_i \tag{5.1}$$

That is, λ_μ is the product of a reaction specific constant k_μ and the product of all the reactant species.

5.2 Deterministic methods

CPNs can be analysed using ordinary differential equations (ODE). These equations are relatively straightforward to solve numerically but will not be able to capture the typical non-negligible fluctuations occurring in biochemical systems of smaller quantities, Levine and Hwa (2007); McAdams and Arkin (1997).

To motivate these problems further, consider the predator-prey example. The model from this example can be analysed by applying Eq. (5.1) *the law of mass action* to **1)-3)** as follows :

$$\begin{aligned}\frac{d[Prey]}{dt} &= k_1[Prey] - k_2[Prey][Predator]\\ \frac{d[Predator]}{dt} &= k_2[Prey][Predator] - k_3[Predator]\end{aligned}$$

Here k_1 describes the rate of **1)** and represents the rate of Prey birth, k_2 the rate of Predation of **2)** and k_3 the rate of Predator death of **3)**. Given the initial condition that $Prey = Predator = 100$ as well as setting $k_1 = 10, k_2 = 0.01$ and $k_3 = 10$ gives rise to the numerically obtained solution depicted in Fig. 5.1.

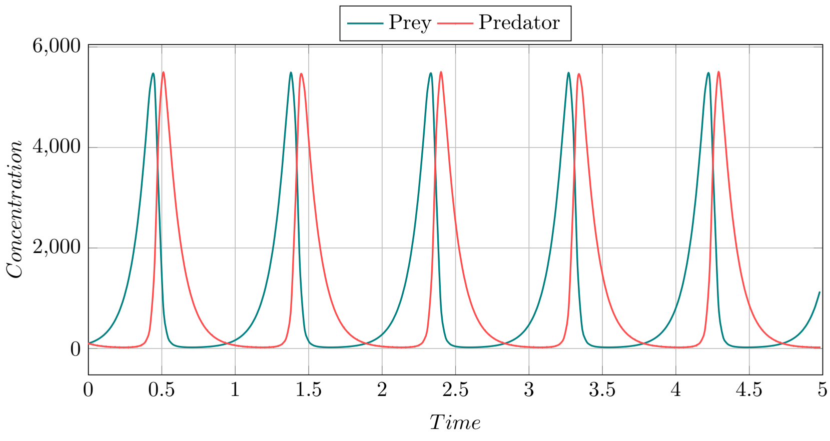


Figure 5.1: Numerical solution of the ODE based Predator-Prey model. The highly cyclic behaviour continues indefinitely.

Inspection of Fig. 5.1 reveals that both populations gets quite close to extinction in each cycle. Population dynamics at this level of coarseness is in reality considered a somewhat random process so it can be expected that in some trajectories extinction could actually occur, i.e. the following three types of trajectories should be possible:

- All prey get eaten so no one will be left to reproduce and all the predators will eventually starve to death.
- All predators die so no one will prevent the population of prey from increasing indefinitely.
- While both species are present in the system, oscillating behaviour similar to that of Fig. 5.1 will occur.

As Fig. 5.1 reveals the deterministic solution is unable to describe the (propagating) effects from the random fluctuations, hence the first two types of solution trajectories will never be observed.

Another problem with the deterministic approach is that it gives rise to continuous values that can cause imprecision in small molecular quantities that only make sense to measure in discrete quantities as for example the binding of promoters. Also notice how the ODE based model cannot satisfy the discrete valued edge multiplicities as e.g. any non-zero real concentration of prey influences the birth rate of further preys as in the case where 0.2 Prey leads to a Prey birth even though the PN dictates that 1 Prey is necessary. To overcome these shortcomings we turn the attention to stochastic methods.

5.3 Stochastic methods

SPNs are analysed using either the *chemical master equation* (CME) or by a *stochastic simulation algorithm* (SSA) that simulates a random trajectory. Two equivalent and exact[1] SSAs were presented in Gillespie (1977): the *direct method* SSA and the *first reaction method* SSA. These algorithms are today considered the gold standard in stochastic simulation and are used as foundation for numerous refined approaches. In this section we present and analyse the CME and the *direct method* SSA.

The stochastic models calculate the likelihood of reacting by the elapsed time it has been enabled, so that the longer the reaction has been enabled, the greater

[1]Generalises the CME.

the probability is for it being fired. Such behaviour is expressed by a poisson process using an exponential distributed random variable $T_\mu \in [0, \infty)$ that denotes the waiting time that has to elapse before firing reaction R_μ. So T_μ has the probability distribution:

$$f_{T_\mu}(\tau) = \lambda_\mu \cdot e^{-\lambda_\mu \cdot \tau} \tag{5.2}$$

5.3.1 Propensity

In the context of stochastic analysis the rate function λ is often referred to as a *propensity function* as it now expresses a likelihood or inclination towards firing. In the remainder of this thesis a will denote the stochastic rate function. Again the *law of mass action* is a very common scheme that gives rise to the propensity a_μ of reaction R_μ often being defined as:

$$a_\mu = c_\mu h(\mu) \tag{5.3}$$

That is, the product of a stochastic reaction rate constant c_μ as well as a combination function h typically defined as the *law of mass action*:

$$h(\mu) = \prod_{S_i \in \bullet\mu} X_i \tag{5.4}$$

The calculation of c_μ for a *bimolecular reaction* on the form $S_1 + S_2 \xrightarrow{c_\mu} S_3 + ...$ was derived in Gillespie (1977) as:

$$c_\mu = \Omega^{-1}\pi(r_1 + r_2)^2 \overline{v}_{12} p_\mu \tag{5.5}$$

Using the simplifying assumptions of *Brownian dynamics* that relates reaction rates to spherical collisions in a constant volume Ω with radii r_i of the molecules and the average relative speed S_2 sees S_1 moving at $\overline{v}_{12}$ as well as the probability p_μ of firing if colliding. Naturally Eq. (5.4) cannot directly account for bimolecular reactions when $S_1 = S_2$ so here $a_\mu = c_\mu(h(\mu) - 1)/2$.

For a *monomolecular reaction* on the form $S_1 \xrightarrow{c_\mu} S_2 + ...$ where reactions occur spontaneously, the constant c_μ relates to the quantum mechanical workings of S_1.

Trimolecular reactions and higher can then just be considered as a sequence of bimolecular reactions. This formulation requires only a constant c to be uniquely supplied for each reaction where Gillespie (1977) also showed how the stochastic- and deterministic-constant c and k respectively relate: for monomolecular reactions $c_\mu = k_\mu$ and for bimolecular reactions $c_\mu = k_\mu/\Omega$. Any n-molecular reaction should be considered as a chain of bimolecular reactions so $c_\mu \propto \Omega^{-n-1}$.

An important thing to note is that a deterministically obtained rate constant k is not necessarily directly applicable as the constant c in a stochastic analysis. A very common mistake or assumption made is to treat these as equal due to the difficulty of obtaining rates as will be further elaborated in Sec. 6.3. As the ratio between constants is the most determining factor, the effect of this mistake or assumption is typically negligible but depends on the particular case as explained in Wu et al. (2011) which also presents a method to convert these rates between a stochastic- and deterministic model.

5.3.2 Chemical master equation

As $X(t)$ is a discrete jump *Markov process*, the probability of each transition is only determined on the current state and do not take history into account – although the input network can be modelled to accommodate for this with the risk of state-space explosion.

The CME describes the probability $P(x, t|x_0, t_0)$ of the system to be in the state $X(t) = x$ given $X(t_0) = x_0$ as

$$\frac{dP(x,t|x_0,t_0)}{dt} = \sum_{j=1}^{m} a_j(x - v_j)P(x - v_j, t|x_0, t_0) - a_j(x)P(x, t|x_0, t_0) \tag{5.6}$$

That is, the probability of arriving at x at time t is the probability of arriving at x from the associated state-change vector v_j to reaction R_j minus the probability of leaving x by v_j.

The CME implicitly calculates every possible trajectory of the systems which is intractable for all but the smallest examples.

5.3.3 Gillespie's direct method

This SSA simulates a single trajectory true to the CME so repeating and averaging approximates the solution obtained from the CME.

Instead of solving with a sufficiently low dt to capture the dynamics, often recalculating the same state multiple times, this method only needs to recalculate the current state after a reaction has occurred, see Algorithm 1. Gillespie's algorithm basically progresses by repeatedly asking the two questions: 1) *which reaction is the next to occur?* and 2) *when does it occur?* These questions

Algorithm 1 Gillespie's direct method

```
Input propensity functions to $a_j \quad (j = 1, ..., M)$
Input initial markings $X(0)$
Set $t = 0$
Initialize URN
repeat
    Calculate $a_j \quad (j = 1, ..., M)$
    Set $a_0 = \sum_{j=1}^{M} a_j$
    Generate $r_1$ and $r_2$ from URN
    Take $\tau = (1/a_0)\ln(1/r_1)$
    Find $\mu$, the smallest integer satisfying$\sum_{j=1}^{\mu} a_j > r_2 a_0$
    Generate the state $X(t+\tau) = X(t) + v_\mu$.
    Put $t = t + \tau$
until Satisfactory results
```

are answered probabilistically by the joint *probability density function* (PDF) $P(\mu, \tau)$ where μ indicates that reaction R_μ occurs at time τ.

a_0 is here defined as the sum of all propensities:

$$a_0 = \sum_{j=1}^{M} a_j \tag{5.7}$$

r_1 and r_2 are two random numbers in the range $[0; 1]$ drawn using a *uniform random number* (URN) generator. τ, the time for the next reaction to occur, is then determined as:

$$\tau = (1/a_0)\ln(1/r_1) \tag{5.8}$$

Fig. 5.2 illustrates how a_0 affects τ. Finally reaction R_μ is identified as being the reaction with the smallest μ fulfilling the following:

$$\sum_{j=1}^{\mu} a_j > r_2 a_0 \tag{5.9}$$

Which describes a PDF of the form a_μ/a_0. So if e.g. $a_0 = 225$ and $a_1 = 50$, $a_2 = 75$ and $a_3 = 100$. Then the probability of selecting reaction R_3 becomes $100/225 \approx 0.45$.

5.3.3.1 Run-time analysis

As each of the operations have the following complexities the run-time complexity for performing <u>one step</u> is linear to the number of reactions $O(M)$:

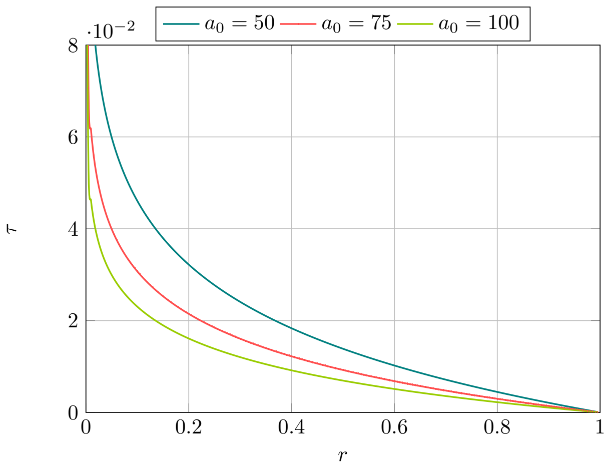


Figure 5.2: Illustrating Eq. (5.8) on how a_0 affects the choice of τ. Here depicted for $a_0 = 50$, 75 and 100. r is the URN r_1 so that if $r = 0.40$ and $a_0 = 100$ then $\tau \approx 0.01$

- Calculating the M propensity values requires $O(M)$ time. As at most two molecules are allowed to collide in a single step the propensity function will take constant time to calculate[2].
- Calculating a_0 requires summing each of the M propensity values $O(M)$.
- Calculating τ takes constant time.
- Retrieving μ requires $O(M)$ as each of the M reactions may need to be evaluated.
- Updating the current markings takes $O(N)$ time as a single reaction is allowed to have up to N species as products. As degradations always should be specified for each species we have that $N \leq M$ hence in total $O(M)$.

Due to the stochastic nature it is difficult to statically make bounds of the number of steps required to simulate a given interval, and as formulated in Gillespie (2007) *"Any procedure that simulates every reaction event one at a time, no matter how efficiently it does that, is simply too slow for many practical applications"* so we have to take a look at improving this simulation algorithm.

[2]Assuming it only consists of constant time arithmetic.

5.3.3.2 Refinements and extensions

Below are some typically encountered refinements and extensions to the Gillespie algorithm:

Next Reaction Method Gibson and Bruck (2000) By only updating the reactions whose propensity has changed, while using a binary search tree to retrieve μ, brings down the asymptotic running-time to $O(\log(M))$ per step for weakly coupled networks (reactants and products in any reaction are bound by a constant) and $O(M)$ otherwise.

Composite-Rejection Slepoy et al. (2008) By maintaining a grouping of reactions by their current propensity and employing techniques similar to the *Next Reaction Method*, it is possible to bring down the running-time to an expected $O(1)$ per step for weakly coupled networks and $O(M)$ otherwise.

Tau-leaping Gillespie (2001) This method trades exactness for simulation speed by setting τ to a predetermined interval instead of going to the next reaction firing, allowing multiple reactions to fire within a single leap. This has been shown to speed up the simulation by up to 4 orders of magnitudes without experiencing a significant loss in approximation. As reactants can be consumed more times within a leap than what is available, tau-leaping introduces the risk of observing negative populations. Several fixes to this shortcoming have been proposed in Tian and Burrage (2004); Chatterjee et al. (2005). Much ongoing SSA enhancement research refines or extends this idea of tau-leaping.

Delayed Stochastic Simulation Bratsun et al. (2005) Extends the Gillespie algorithm to account for delayed kinetics. This addition allows users to specify a minimum time from a given reaction is enabled to it is allowed to fire.

These techniques can be combined into very expressive and efficient ways of simulating. Due to the low implementation overhead a combination of *Tau-leaping* and the *Next Reaction Method* is widely used in many stochastic simulation applications.

5.4 Discussion

Hybrid approaches that use both deterministic and stochastic methods have been proposed in e.g. Alfonsi et al. (2004); Menz et al. (2012) often referred to as *slow-scale SSA*. Solving a system deterministically is orders of magnitudes faster than using any explicit stochastic simulation approach. The idea is to switch methods when sufficiently large quantities of species becomes present as the stochastic effects here becomes negligible anyway. Gillespie (2007) also proved this to work while employing tau-leaping.

Recent developments on both reducing the dimensionality of the CME as well as developing numerical solvers and heuristics tailored for it, shows promising results. The *Finite State Projection Method* by Munsky and Khammash (2006) is one such example that in some selected cases outperforms the SSAs based on tau-leaping.

Modelling cell divisions and growth rates are deemed important when the reactions rates in comparison are slower as can be the case for protein production from infrequently expressed genes. A modification to Gillespie based SSAs taking this into account is presented in Lu et al. (2004).

Simulation methods taking full spatial information into account are in general computationally limited to handle only very small quantities of species. In order to overcome this, models which only partially considers the spatial attributes are being considered, Burrage et al. (2011); Wylie et al. (2006), but are still very computational demanding. The main driving-force of these methods is to group the spatial inhomogeneous compartments into less inhomogeneous sub-compartments where the coarseness of these groupings are determined by the available computational resources. Each of these sub-compartments can then be solved in conventional manor where crosstalk is modelled as additional reactants with reaction rates corresponding to their spatial attributes. Having to support spatial heterogeneous systems also introduces the difficulty of having to capture the spatial attributes.

Chapter 6

Modelling

Because of the inherent stochastic behaviour in biochemical systems such as gene expression, the *stochastic petri net* (SPN) described in the previous chapters is an excellent fit to be used to model and describe gene expression in general and genetic devices composed of parts in particular.

In this chapter we will present how the different parts and processes involved in gene expression can be modelled as networks, how parameters for the models can be found and what these parameters represent. Furthermore we will go into details with a case study of a genetic device, namely a negative feedback device for which a SPN model, the parameters for the model and simulations of it will be shown.

In the prokaryotic cell the processes transcription and translation of gene expression have been demonstrated noisy, i.e. stochastic, in several experiments, e.g. McAdams and Arkin (1997); Elowitz and Leibler (2000). Furthermore fluctuations of the concentration levels are very common in gene expression, Chalancon et al. (2012), these fluctuations can be caused by e.g. the low concentration of molecules causing the reactions to be less predictable. The fluctuations can also be a result of the cellular environment in general, which is never completely homogeneous across different cell populations. The stochastic behaviour of biochemical systems in general and gene expression in particular makes SPNs an excellent choice as model.

A model based on SPN is highly extensible, thus a SPN model of gene expression can always be extended with more details, e.g. the entire process of translation happening in the ribosome, how the RNA polymerase binds to operators and initiates or represses transcription or how the protein folds after translation has finished. The SPN model is intuitively understandable and is a great way to explain rather complex processes happening in biochemical systems. Using SPNs to model these biochemical systems also enables easy model checking and verification of certain mathematical properties of the models.

6.1 Abstraction level

Gene expression can be modelled on many levels of abstraction, e.g. Roussel and Zhu (2006) models all stages of the transcription process described in Sec. 2.3.1 on page 11: RNA polymerase binding to the promoter, the RNA polymerase moving down the DNA and finally termination of the transcription. Another paper, Arkin et al. (1998), models how the ribosome binds to the mRNA and translates the codons into amino acids.

Common for these different models is that they can be described as *biochemical reactions*, which in turn can be expressed as SPNs. The following reactions describing gene expression are adapted from Ribeiro et al. (2006):

$$RNAP(t) + Pro_i(t) \xrightarrow{k_i} Pro_i(t+\tau_1) + RNAP(t+\tau_2) + n_i r_i(t+\tau_3) \tag{6.1}$$

$$r_i \xrightarrow{k_{i,2}} \emptyset \tag{6.2}$$

In Reaction (6.1) gene expression of gene i is described, where $RNAP$ is RNA polymerase, Pro_i is the promoter site, r_i is the protein created during the translation of the $mRNA$ created during transcription. t is the time, τ_1, τ_2 and τ_3 are the times for each of the products in the reaction to become available in the system. n_i is a constant associated with the rate of translation and transcription describing the protein production rate. k_i is the rate constant for $RNAP$ to bind to the promoter site on the DNA. Note that in one step this reaction expresses how $RNAP$ binds and how transcription and translation produces protein. A lot of details are omitted here, e.g. the inherent delay in gene expression when ribosomes start translation. Reaction (6.2) represents the decay of the protein happening with the rate constant $k_{i,2}$.

Note that we are not providing any information about which rate laws are used with the rate constants k_i and $k_{i,2}$. The law of mass action explained in Sec. 5.1 is often used as rate law, thus in the remainder of this chapter we will use

c_i above the reaction arrow when the symbol should be used as rate constant in the law of mass action and if a custom rate function is used we use a_i.

Referring to Reaction (6.1) we want to abstract away the *RNAP* binding to the promoter site, but we want to be more specific on *mRNA* production and decay. The following altered, simplified biochemical reactions take this into account:

$$DNA \xrightarrow{a_1} DNA + mRNA \tag{6.3}$$

$$mRNA \xrightarrow{c_1} mRNA + r \tag{6.4}$$

$$mRNA \xrightarrow{c_2} \emptyset \tag{6.5}$$

$$r \xrightarrow{c_3} \emptyset \tag{6.6}$$

These reactions describe *the central dogma*, refer to Ch. 2 on page 7, with decay of the *mRNA* and the gene product r. The rate function a_1 represents the constant transcription rate of the specific *DNA* when the promoter is available, c_1 represents the translation rate of the *mRNA* and the rate constants c_2 and c_3 represent the decay rates of *mRNA* and the gene product r respectively.

Here a_1 is a rate function because the availability of the promoter decides if transcription is enabled, thus the rate function a_1 should evaluate to 0 when the promoter is unavailable and greater than 0 otherwise. The other rate constants c_i are used with the law of mass action.

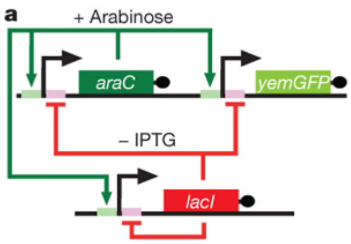
We will focus on simple models with simple pathways, which is why we in the following sections build models with a few assumptions:

- Constant temperatures in the cell environment.
- Only one cell is considered, i.e. no cell-to-cell interactions are modelled.
- No protein-protein interactions, e.g. reactions between different proteins, do not happen unless specified.

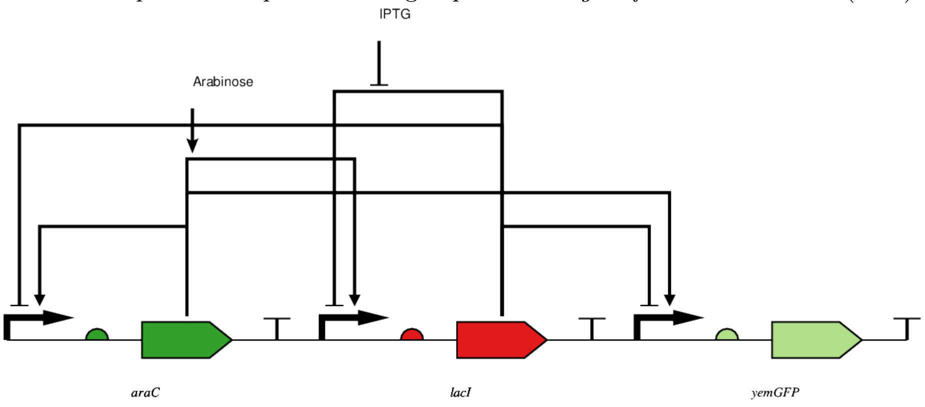
6.1.1 Case study: Negative feedback loop

The reactions above omit details about regulation. In this section we show an example of a gene with a negative feedback loop where the promoter is negatively regulated by the protein produced by the gene itself. This gene is part of a genetic oscillator device (device A), Stricker et al. (2008), though we have slightly simplified the model to be able to incorporate modelling parameters from another genetic oscillator device (device B), Elowitz and Leibler (2000).

Device A, see Fig. 6.1, consists of three identical hybrid promoters repressed by LacI protein in the absence of isopropyl β-D-1-thiogalactopyranoside (IPTG) and activated by AraC protein in the presence of arabinose. The protein coding sequences araC[1], lacI and yemGFP[2] were placed under control of these promoters, thus co-regulating each other. The system exhibit positive feedback when AraC is produced and negative feedback when LacI is produced.



(a) Device A. The small, light colored boxes represent the promoters and the other boxes represent the protein coding sequences. *Figure from* Stricker et al. (2008).

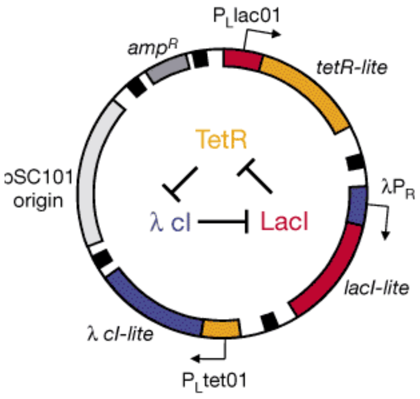


(b) Device A in the format used in this thesis.

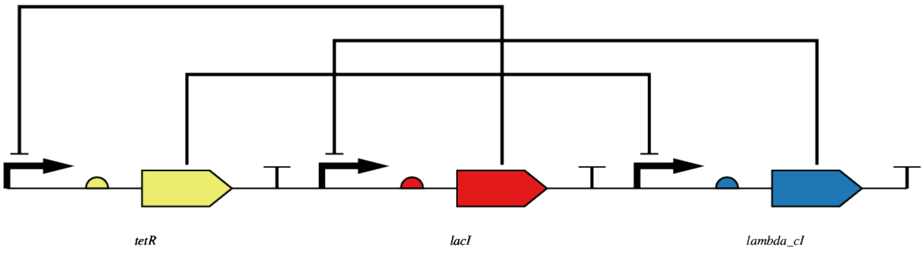
Figure 6.1

Device B, see Fig. 6.2, contains a cyclic negative feedback loop with the protein LacI from E. coli repressing transcription of TetR from the tetracycline-resistance transposon Tn10 gene, which in turn represses expression of λcI from the lambda phage gene which finally represses LacI expression in E. coli. The genes are manipulated to align the overall behaviour enabling oscillations, i.e. the behaviour is described by the rates of transcription, translation and decay.

[1]In general when referring to proteins the first letter is capitalised, e.g. AraC, and when referring to genes (or protein coding sequences) the first letter is small, e.g. araC.
[2]Producing green fluorescent protein, used as a reporter gene.



(a) Device B, here depicted in plasmids. The diagram in the middle shows how the different genes repress each other. *Figure from* Elowitz and Leibler (2000).



(b) Device B in the format used in this thesis.

Figure 6.2

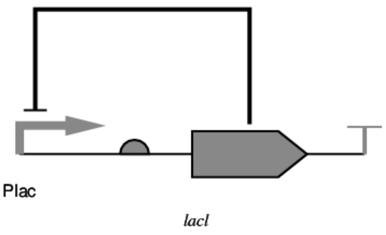
Because we only want to show the negative feedback loop we make assumptions, abstract details away from device A and incorporate some details from device B:

- In *wet-lab* experiments the yemGFP gene is very important, as this is used to determine the protein levels at certain time intervals, in computer simulations we can read the levels directly, thus we will not model this part.
- Presence and absence of IPTG and arabinose is not modelled, we assume the rate constants chosen will reflect this.
- In device B the three genes behave alike, we use this fact to replace the hybrid promoter from device A with the lacI promoter from device B. This promoter also exhibits repression from the LacI protein.
- The promoter from device B is assumed to have two operator sites, to

simplify the model we assume only one operator site is present.

The resulting gene described above and its reactions describing promoter regulation is shown in Fig. 6.3. Here the species $P_{lac} \cdot LacI$ indicates that $LacI$ has been bound to P_{lac}. The rate constant c_4 typically will be close to 0 because $LacI$ represses the promoter. The rate constants c_2 and c_3 represent the rate of binding and unbinding. c_1 is the transcription rate constant and c_5 is the translation rate constant. c_6 and c_7 are decay rate constants. If the promoter is strongly regulated by the protein the rate constant c_2 will be considerably larger than c_3, meaning the protein binds to the promoter almost instantly when it is present and unbinds at much lower frequency.

The rates of these reactions do not follow the Reactions (6.3)-(6.6), but the modelling parameters given in Elowitz and Leibler (2000) require the reactions to have rates on this form.



$$
\begin{aligned}
P_{lac} &\xrightarrow{c_1} P_{lac} + mRNA \\
P_{lac} + LacI &\xrightarrow{c_2} P_{lac} \cdot LacI \\
P_{lac} \cdot LacI &\xrightarrow{c_3} P_{lac} + LacI \\
P_{lac} \cdot LacI &\xrightarrow{c_4} P_{lac} \cdot LacI + mRNA \\
mRNA &\xrightarrow{c_5} mRNA + LacI \\
mRNA &\xrightarrow{c_6} \emptyset \\
LacI &\xrightarrow{c_7} \emptyset
\end{aligned}
$$

Figure 6.3: Negative feedback device and its reactions.

Note that we are fully aware that these alterations of the genes might not be possible in wet-lab experiments, but for now this case study is used as proof-of-concept of how we can model genetic devices with SPNs. Later we will argue why this suffices.

6.2 SPN representation

Now we will show how we can make a one-to-one mapping from biochemical reactions to SPNs. Biochemical reactions map to SPN similar to how chemical reactions map to PN, see Sec. 4.2. In the following we will name transitions as well, to indicate which reaction is occurring.

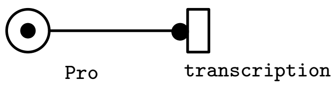
6.2.1 Parts and processes

SPN models of different genetic devices will share some similarities, e.g. a model of a repressed promoter in one genetic device will have a certain design that might only be slightly different than the model of a repressed promoter in another genetic device. Each of the transcription, translation and decay processes will also often be modelled identically in different SPN models.

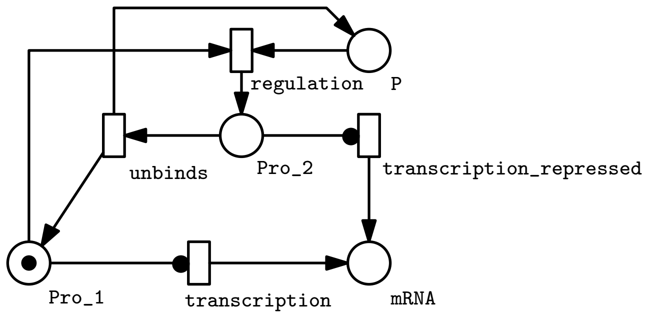
To ease the construction of new models of genetic devices we have modelled these general parts and processes here. These models can serve as templates for new models.

The promoter is modelled by one place `Pro` or several places `Pro_i`, `i` $\in \mathbb{N}$, indicating different states of the promoter, i.e. if RNA polymerase can begin transcription or not. Thus the promoter places will not represent a molecule, but will function as a kind of Boolean operator turning something on and off, which is exactly the way a promoter can be explained: turning a gene on and off.

Constitutive promoter This can be modelled with the marking $m(\texttt{Pro}) = 1$ and a read arc from the `Pro` place towards the `transcription` transition. The read arc ensures that mRNA can be transcribed at all times because the token is not consumed.

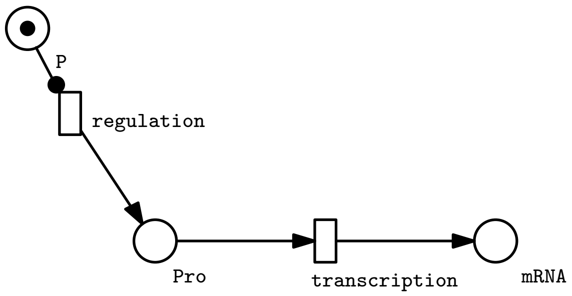


Negatively regulated promoter The protein negatively regulating the promoter is represented by the place `P` and the place `Pro_2` represents the repressed promoter. The `regulation` transition is only enabled when there is protein present, if this is the case the token from `Pro_1` will be consumed, a token from `P` will be consumed and a token will be produced at `Pro_2`, and transcription will not be possible. The marking $m(\texttt{Pro\_1}) = 1$ indicates that the promoter initially is available for transcription.



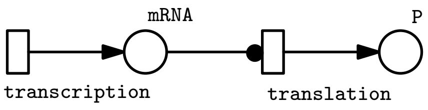
Sometimes transcription also takes place when the promoter is bound (but at a low rate), this can easily be modelled by adding a low rate transition with Pro_2 as reactant and mRNA as product. Negative regulation can also be modelled without places representing promoters by adding modifier arcs directly from the repressor protein to the transcription transition, an example of this including an appropriate rate function is outlined in Sec. 6.4 on page 59.

Positively regulated promoter The protein positively regulating the promoter is represented by the place P, here with the marking $m(\mathtt{P}) = 1$ indicating that the promoter is available. In this model the promoter will only be available when the protein is present as the regulation transition produces tokens in the Pro place and the arc from the promoter to the transcription transition is a regular arc, i.e. the token from Pro is consumed every time an mRNA molecule has been produced.



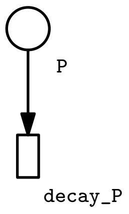
We are aware this has low coherence with reality, as the promoter is not "consumed" but available for transcription, but for modelling purposes this is fine. If finer control is needed a new place Pro_2 representing the non-active promoter could easily be introduced making it possible to model unbinding of the protein from the promoter. This illustrates how the flexibility of SPNs makes them a great choice for modelling of genetic devices.

Transcription and translation The `transcription` transition produces tokens in the `mRNA` place which will enable the `translation` transition, which will produce tokens in the protein place `P`.



The read arc from `mRNA` to the `translation` transition makes sure that no `mRNA` is consumed when the transition is fired. Here the firing rate can be a constant not taking `mRNA` into account. Another design choice could be to move control into the firing rate by replacing the read arc with a modifier arc. In this way the transition will always be enabled but only fire when the firing rate is greater than 0, hence the firing rate needs to take the `mRNA` place into account, e.g. `mRNA`$\cdot k$ where k is some translation rate. The rate of translation depends on the amount of `mRNA`, meaning that the firing rate often would be similar to the latter case.

Decay In the model shown above `mRNA` will keep growing as there is only an outgoing read arc from the place. `mRNA` (and protein) degrades over time, hence we need to model this behaviour. With an arc from the respective place to a transition with no outgoing arcs, we ensure that the molecules will go away over time. In the model below decay of a protein `P` is modelled.



6.2.2 Case study continued

The next step in the creation of the model of the negative feedback device presented earlier is creating the SPN. Fig. 6.4 shows how the device can be modelled as a SPN. The model is created by considering the models of parts proposed above and the reactions from Sec. 6.1.1 one at a time.

The reactions in Fig. 6.3 on page 50 correspond to each of the transitions in Fig. 6.4. The negatively regulated promoter is modelled as shown above: the `regulation` transition will be enabled whenever the marking of the places

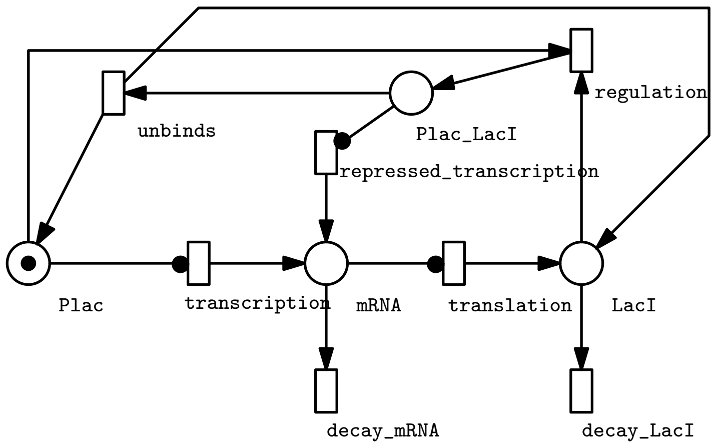


Figure 6.4: Negative feedback device as SPN. This is the SPN of the device and its reactions from Fig. 6.3 on page 50.

`Plac` and `LacI` is greater than 0. The place `Plac_LacI` represents the repressed promoter, which can exhibit transcription through the transition `repressed_-transcription`. When the `unbinds` transition is fired the token from `Plac_-LacI` will be consumed and a token at `Plac` and `LacI` will be produced.

6.3 Parameters

The SPN models of processes in gene expression described in the previous section expresses the flow of the different species in genetic devices. To enable simulations of the SPN models the firing rate for the transitions need to be determined by different parameters describing the speed at which the different processes take place.

We have previously discussed how the deterministic and stochastic rate-constants relates, here we will explain how the deterministic rates for the biochemical reactions transcription, translation and decay presented in the previous section can be interpreted. Often the change of amounts of species involved in the processes mentioned can be explained using the law of mass action introduced in Sec. 5.1.

Transcription The reaction rate for transcription is constant, i.e. it is not dependant on the concentration of the reactants involved in the process, only on other factors, e.g. temperatures as mentioned before:

$$DNA \xrightarrow{a} DNA + mRNA$$

Here it is not clear that the *mRNA* is only transcribed when the promoter on the *DNA* is activated, this means the transcription is constant but only when promoter is available. The amount of DNA will not change over time but the amount of *mRNA* will, thus we define the reaction rate as:

$$\frac{d[mRNA]}{dt} = a, \frac{d[DNA]}{dt} = 0$$

Where a is the transcription rate which is a constant and the change of *DNA* is 0 as this amount will remain the same at all times. The equations above only express the rates when certain processes can take place, a model using these rates must accommodate for the lack of expressiveness of when these processes can take place.

Translation As several ribosomes can translate one *mRNA* molecule at a time, the translation rate must be dependant on the amount of present *mRNA* molecules:

$$mRNA \xrightarrow{k} mRNA + r$$
$$\frac{d[r]}{dt} = k[mRNA], \frac{d[mRNA]}{dt} = 0$$

Here k is some translation rate constant used in conjunction with the law of mass action and r is the protein produced.

mRNA and protein decay The reaction above lacks information on change of *mRNA* per time unit. Decay can be expressed as:

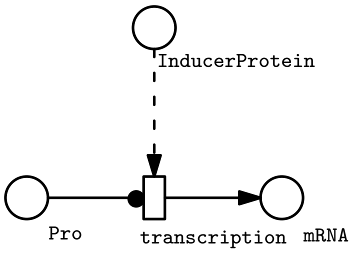
$$mRNA \xrightarrow{k} \emptyset$$
$$\frac{d[mRNA]}{dt} = -k[mRNA]$$

Where k is some decay constant. This gives us the total change of *mRNA*:

$$\frac{d[mRNA]}{dt} = a - k[mRNA]$$

The same holds for protein r mentioned in translation above.

Some of the rates above are described by the law of mass action but other rate functions are allowed as well. In the law of mass action only the reactants in the chemical reaction in question are involved, but the general rate function for biochemical reactions allows other variables than the reactants. Referring to Sec. 4.3 on Rates of firing in SPNs, these variables are usually called *modifiers* and are not part of the biochemical reaction itself but might be products or reactants in other biochemical reactions in the system and influence the biochemical reaction because of its presence in the system. Fig. 6.5 is an example of how the rate function can include a modifier variable.



$$DNA \xrightarrow{a} DNA + mRNA$$
$$\frac{d[\texttt{mRNA}]}{dt} = k \cdot \texttt{InducerProtein}$$

Figure 6.5: Positively regulated transcription. Only presence of `InducerProtein` (and the promoter `Pro`) will result in transcription. No `InducerProtein` or `Pro` is consumed by this transition. Here the transition will only be enabled when `Pro` is present, and the rate is following the amount of `InducerProtein` present.

6.3.1 Identifying rates

The simulation of the models should give an idea of how the synthetic engineered genetic devices will behave when wet-lab experiments are carried out. Preferably the parameters used to derive rate constants should be identified in other wet-lab experiments to give the most accurate models. In a system of biochemical reactions the ratio between rates should reflect the ratio between one reaction happening more often than another. This means that often different parameters with different units for different reactions have to be related to one another to make them usable in simulations. The process of finding parameters for these models to be used in computer simulations is not trivial and several challenges have appeared:

- One huge challenge is the current lack of experimental data that can be used to set the parameters of these models. Resources such as BioNumbers[3], Milo et al. (2010), contains a lot of raw parameters from different

[3]`http://bionumbers.hms.harvard.edu/`

experiments, e.g. rates of transcription in different media expressed as *nucleotides per second* can be found.

- The problem with data from sources like BioNumbers is that experiments show that these parameters rely heavily on cell environment, temperatures, protein concentrations and other factors.

- Different data have to be related to each other. E.g. the unit *nucleotides per second* mentioned above needs to be related to e.g. translation expressed as *amino acids per second*.

- When relevant experiments have been found, the process of fitting the data from the experiments and estimate parameters is quite complex as well, Weiss et al. (2005); Lillacci and Khammash (2010).

The task of finding the accurate parameters for the rates boils down to reviewing existing literature and experiments and fitting the data and results to get the desired behaviour of the model. But if relevant data is lacking an experiment has to be carried out, thus the task of identifying the correct parameters is a huge area by itself and often detailed understanding of biology and performing wet-lab experiments is needed.

Another difficulty, as mentioned in Ch. 5, is that empirically obtained deterministic rates cannot always be used directly as stochastic rates although they often make a good approximation.

6.3.2 Case study continued

We will now return to the previous case study of the negative feedback device. Table 6.1 shows the rates for the biochemical reactions involved in the negative feedback loop. The rates are based on the modelling parameters from Elowitz and Leibler (2000) which in turn is based on the components used to compose the oscillator.

The unit of the different rates above indicate that the law of mass action has been used as rate function. It should be clear that the SPN combined with the rates here model a behaviour where `LacI` is increasing or decreasing constantly, thus experiencing negative feedback. Of course the rate constants presented here play an important part in simulation of the model, e.g. if the half-time of mRNA is too low translation might never be exhibited.

Reaction	Rate	Comment
$P_{lac} \xrightarrow{c1} P_{lac} + mRNA$	$0.5s^{-1}$	Transcription from unoccupied promoter.
$P_{lac} + LacI \xrightarrow{c2} P_{lac} \cdot LacI$	$1nM^{-1}s^{-1}$	Binding.
$P_{lac} \cdot LacI \xrightarrow{c3} P_{lac} + LacI$	$9s^{-1}$	Unbinding.
$P_{lac} \cdot LacI \xrightarrow{c4} P_{lac} \cdot LacI + mRNA$	$5 \cdot 10^{-4}s^{-1}$	Transcription from repressed promoter.
$mRNA \xrightarrow{c5} mRNA + LacI$	$0.167s^{-1}$	Translation.
$mRNA \xrightarrow{c6} \emptyset$	$\frac{log(2)}{2 \cdot 60} = 0.0058s^{-1}$	mRNA decay: 2 minutes half-life.
$LacI \xrightarrow{c7} \emptyset$	$\frac{log(2)}{10 \cdot 60} = 0.0012s^{-1}$	Protein decay: 10 minutes half-life.

Table 6.1: Rates for the negative feedback device.

6.3.2.1 Simulation

Finally we have a complete model with the SPN model in Fig. 6.4 on page 54 and the parameters for the transition in Table 6.2. The model has been simulated using the framework presented in Ch. 7. Fig. 6.6 shows one simulation and Fig. 6.7 shows the average of 10 simulations.

Transition	Firing rate function
`transcription`	`0.5*Plac`
`regulation`	`1*Plac*LacI`
`unbinds`	`9*Plac_LacI`
`repressed_transcription`	`5*10^(-4)*Plac_LacI`
`translation`	`0.167*mRNA`
`decay_mRNA`	`0.0058*mRNA`
`decay_LacI`	`0.0012*LacI`

Table 6.2: The firing rate functions for each of the transitions.

Fig. 6.6 with one simulation clearly shows how there is some correlation between the $mRNA$ and $LacI$ concentration levels: $LacI$ is produced when there is small spikes of $mRNA$, and decay of $LacI$ happens when there is no spikes of $mRNA$, just as expected. Fig. 6.7 with 10 simulations shows the average behaviour of the negative feedback device: a fairly stable production of $LacI$ in the concentration level interval $[320; 370]$. Again this is just as expected. Parameters of the model can be tuned to exhibit more extreme behaviour, e.g. if the binding of $LacI$ to the promoter is very strong, there will be a much greater span between the minimum and maximum concentration levels of $LacI$, see Fig. 6.8 where the firing rate function of the `unbinds` transition is multiplied by 10^{-1} and the rate function of the `repressed_transcription` is multiplied by 10^2.

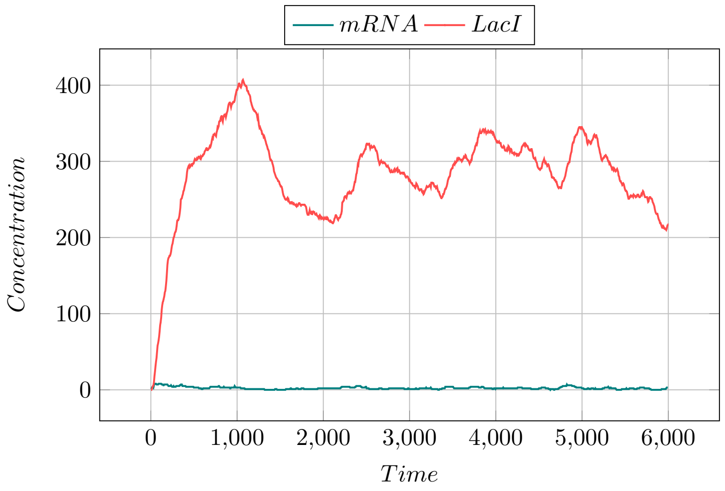


Figure 6.6: 1 simulation of the model of the negative feedback device.

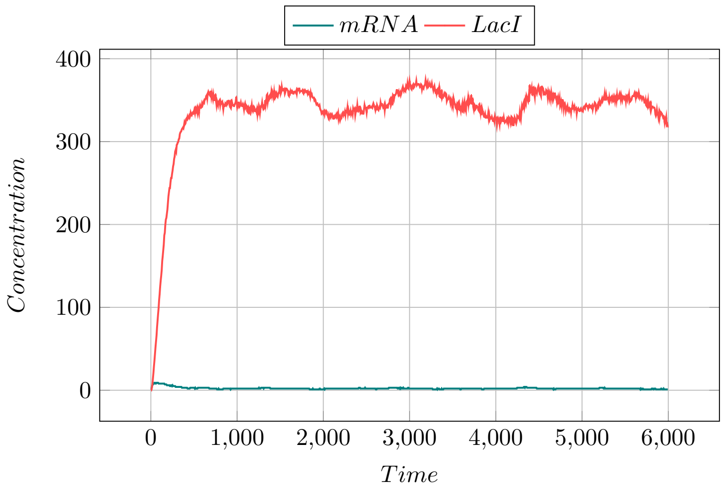


Figure 6.7: 10 simulations of the model of the negative feedback device.

6.4 Example: Oscillator

Here we show how a genetic device can be modelled differently than shown up until now. This example goes through the oscillator device used to construct the negative feedback device studied in this chapter. The oscillator has already been modelled by Elowitz and Leibler (2000), a description of the model has

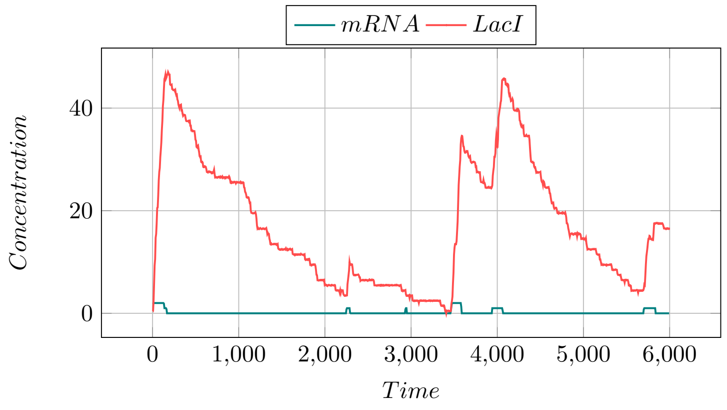


Figure 6.8: 1 simulation of the model of the negative feedback device with stronger promoter binding.

been submitted to the BioModels Database[4] which we will be studying here.

Here it is interesting how relatively simple the model is: promoters and binding of proteins to promoters are not modelled by transitions and places but simply by rate functions on the transitions `ts_LacI`, `ts_cI`, etc. E.g. the rate function (transition `ts_LacI`) for transcription of *LacI* is described by:

$$a0_{tr} + \frac{a_{tr} \cdot KM^n}{KM^n + cI^n} \tag{6.7}$$

This rate function is explained in detail in Appendix A, but it should be obvious that transcription is decreasing when the concentration of *cI* increases. The more simpler modifier arcs are used instead of e.g. read arcs, because the behaviour of the model is primarily described by the rate functions (and not the design of the SPN).

6.5 Discussion

In this chapter we have seen how we can go from a set of biochemical reactions with rates to a complete SPN describing a desired behaviour. The mapping from biochemical reactions to SPNs is unproblematic as SPNs are an extension of PNs which originally were invented for chemical reactions. The challenge is to define a satisfiable set of biochemical reactions defining the behaviour we want.

[4] `http://www.ebi.ac.uk/biomodels-main/BIOMD0000000012`, the SBML format will be further described in the next chapter about the implementation.

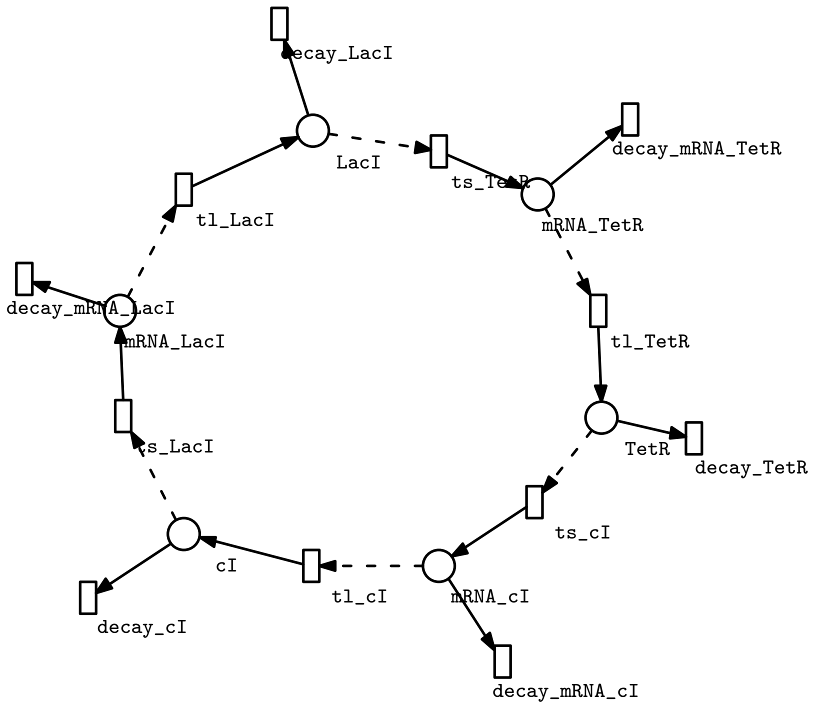


Figure 6.9: The SPN model of the oscillator.

Initially the goal was to standardise how the different parts can be modelled, but we soon realised that to do this we always need to be certain that specific parameters can be obtained for specific reactions. There is a great challenge in finding these parameters: often it is not possible to find the parameters that fit in the specific reaction and the model needs to be modified to reflect the available parameters.

Another issue arose in our review of different literature: there is no de facto standard of describing the internal mechanics of parts in synthetic engineered devices. Hence different literature describe these details of the parts and the processes of gene expression on very different levels.

The purpose of modelling synthetic devices is to be able to predict the behaviour of a device before the device is constructed in wet-labs and by that hopefully save both money and time. This scenario is greatly desirable but the reality is that this is still one of the main challenges in synthetic biology: to get the level of understanding of biological systems to be able to construct a model with all important interactions and parameters, see the email from professor Chris

J. Myers, author of *Engineering Genetic Circuits*, in Appendix B. Currently it is still very hard to find literature with experiments and models with complete justification of all parameters that simulate the behaviour of the experiment (Lodhi and Muggleton, 2010, Ch. 2).

The consequence of this is that detailed, e.g. time-series with protein levels, comparisons of simulations of these models with wet-lab experiments is not always meaningful, on the other hand the overall behaviour of these models can still be compared with the desired behaviour of the wet-lab devices.

The case study of the negative feedback device shows that a SPN model with appropriate parameters exhibit the desired behaviour, which is why we in the remainder of this thesis assume the modelling techniques proposed here are usable for modeling of genetic devices.

Chapter 7

A framework: DTU-SB

In this chapter we present a framework: DTU-SB[1]. The purpose of this framework is to model synthetic engineered genetic devices and make simulations of these models. In this chapter we will list the requirements for the framework, sketch the overall architecture of it, go into details with some interesting parts of the implementation and finally we will list how the framework can be used in different use cases.

7.1 Requirements

The framework should be able to simulate arbitrary complex models of real genetic devices. This means the theory from the previous chapters on SPNs, modelling and simulation should be used to construct the framework.

The simplest form of the framework should be able to take some specification of a model as input, internally this specification should be transformed to a SPN, which in turn should be simulated using Gillespie's direct method SSA explained earlier. At last the result of the simulation should be presented in some pre-defined format.

[1]Short for DTU Synthetic Biology.

To allow comparisons with other tools, a very important requirement is that input parsers and output formatters should be modularised. Specifically the framework should allow the following modules:

- It should be possible to define custom parsers for arbitrary input specification languages. This allows support for an endless amount of model specification languages.
- It should be possible to define a custom algorithm or easily modify or extend existing algorithms for the simulator. This allows optimisations of existing algorithms and experimenting with new algorithms.
- It should be possible to define custom output formatters.
- It should be possible to add intermediate compilers of the SPN models produced by the input parsers. These compilers could be used to e.g. hook up a database with additional information or by making optimisations on SPN models.

One might notice that the framework sketched here differ from the proposed tool-chain in Fig. 3.2 on page 22 in that the output here focuses on simulation results rather than complete DNA sequences describing some synthetic engineered biological function.

In addition to the requirements of the behaviour and features of the framework described above, there is also some requirements to how the framework should be implemented to achieve these features. As the framework is not only intended as proof-of-concept but should form the basis for a long living modelling framework for synthetic biology, the following requirements capture how this can be achieved:

Maintainable It should be easy to modify, update and fix the current behaviour.

Flexible It should be possible to add functionality purely by adding new modules, not by modifying existing modules.

Testable It should be possible to verify the internal logic of the framework components through test cases.

Wide OS Support The user's operating system should not be an obstacle to the availability of this framework.

Maintainability, flexibility and testability often go hand in hand and can be obtained simply by making the framework modular by implementing the *dependency injection* design pattern. To support as many operating systems as possible we choose *Java* as programming language, which run on the Java Virtual Machine (JVM) on several operating systems. This choice also allows future usage of the framework in the functional programming language *Scala*, which compiles into Java bytecode (which can be run on the JVM), thus Java code can be used directly in Scala code.

7.2 Architecture and data flow

Based on the requirements above we sketch the design of the foundation of the DTU-SB Framework. Fig. 7.1 shows the steps involved in a simulation of a model in the framework. The parts `PARSER`, `OUTPUT FORMATTER` and `ALGORITHM` are all modules which allows custom made versions of these parts. To add further extensibility the `COMPILER` consists of a chain of compiler modules as well.

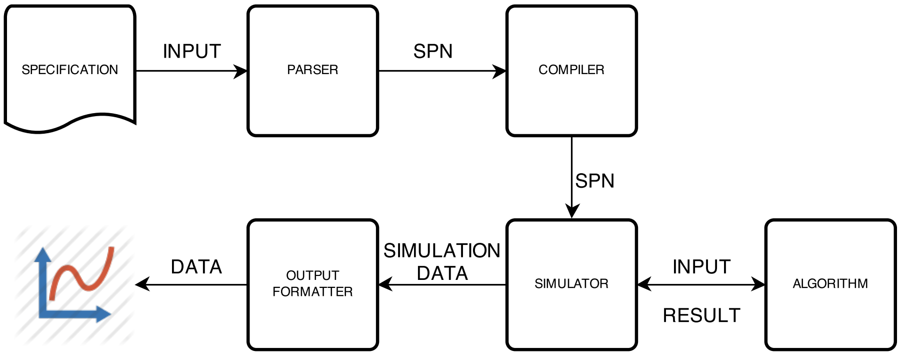


Figure 7.1: Data flow from the specification of the model to the final result of the simulation.

The `SPECIFICATION` is a model specification in an arbitrary specification language. The `PARSER` will translate this specification into a stochastic petri net (`SPN`), which will be verified by the `COMPILER`. One could imagine that the `COMPILER` could have several other uses as well, e.g. adding more information about specific modelling parts. The `COMPILER` will output a modified `SPN` which the `SIMULATOR` can use as input. The `SIMULATOR` simulates the `SPN` using the `ALGORITHM` and outputs data that can be formatted using the `OUTPUT FORMATTER`. The `SPN` used here will support regular arcs, modifier arcs, read arcs, places and transitions with rate constants and functions.

7.3 Review of formats and third-party libraries

We will not introduce any new formats or standards but leverage the formats already used by the biology community. To avoid spending time on writing parsers from the bottom-up, we will make use of third-party libraries for these formats as well.

7.3.1 Input formats

By easily exposing the simulator to as many frontends and pre-defined setups as possible, verification and comparison becomes much more manageable. Below there is a short review of some popular description formats supported by various applications.

SBML *Systems Biology Markup Language*, Finney et al. (2001), is an XML based language for describing biological processes. It implements MathML for describing mathematical relations such as reaction rates.

CellML The goal of CellML is similar to that of SBML but is more general and have a more flexible syntax. SBML is practically a subset of CellML. The teams behind SBML and CellML work in close collaboration and are discussing the possibilities of merging the two languages.

PNML *Petri Net Markup Language* is an XML based language focused on describing petri nets. PNML is similar to SBML in many ways but lacks details about rates, thus the compiler must accommodate for this.

For now the framework will support SBML out of the box as it seems to be the *de facto* standard with approximately 300 software tools making use of it. Several libraries for reading, manipulating and writing SBML are available, specifically we can make use of JSBML for Java. Furthermore many tools for converting CellML to SBML are freely available on the Internet.

7.3.2 Intermediate format

The compiler compiles the specifications to an intermediate representation (IR) which the simulator uses as input format. As the diagram in Fig. 7.1 indicates, we use a custom defined SPN on the form of a simple Java data-holder class as our IR.

Some of the advantages of using a custom defined SPN is that we can incorporate special operations in the SPN and it allows seamless translation to PNML for quantitative analysis and model checking using third-party tools as well as easy visualisation, e.g. by using the DOT format (a graphics description language).

7.3.3 Simulation output formats

The raw format of the output from the simulations should be time series of concentration levels of different interesting species involved in the reactions. The comma-separated values (CSV) format is a common plain-text file-format used to store tabular data such as time-series, which we will provide as one output format.

To enable easy, immediate visualisation of the simulations we also provide a GUI with a graph based on the JFreeChart library.

7.4 Implementation details

The class diagram in Fig. 7.2 shows the overall design of how the implementation support the data flow shown in Fig. 7.1, and how the different parts of the framework are completely decoupled, making the framework maintainable, flexible and testable as desired.

The diagrams in Fig. 7.2 and 7.3 capture the most important parts of the entire framework. Each of the abstract classes and the `Compiler`- and `Simulator`-class also have a method `setParams(Parameters params)` that can be used to pass in an object with custom parameters – this avoids endlessly recompilations and endless command line arguments when several simulations are required to find the optimal simulation parameters.

In the diagrams the data-holder classes `StochasticPetriNet`, `SimulationResult` and `PlotPoint` are shown (there are a few others not shown here). These classes

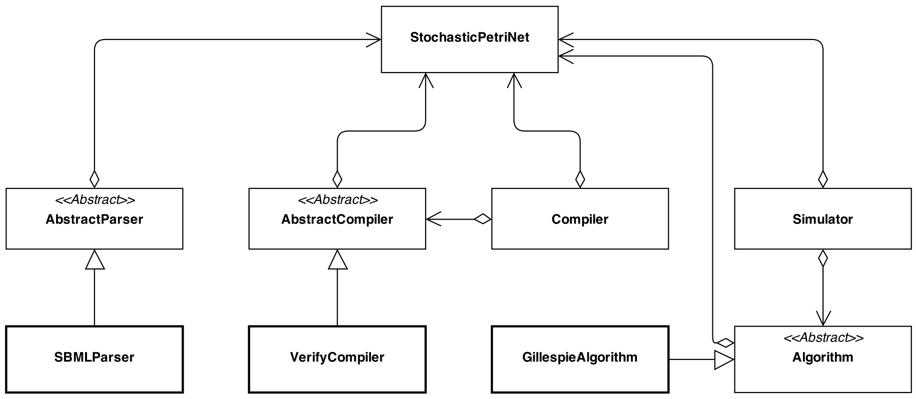


Figure 7.2: Class diagram showing the main parts of the implementation. The classes in the boxes with bold borders are concrete implementations of the abstract classes.

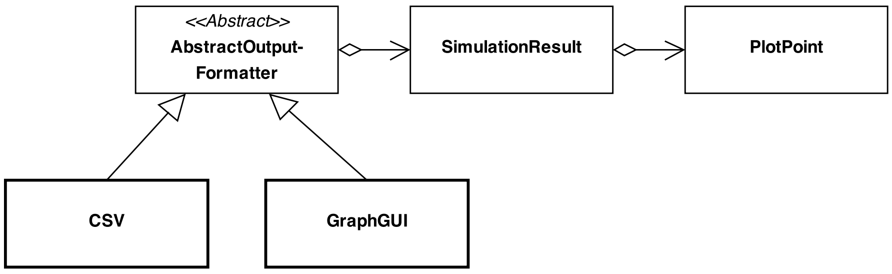


Figure 7.3: Class diagram showing how output formatters are implemented. Here `CSV` and `GraphGUI` are concrete implementations of the `AbstractOutputFormatter`.

primarily holds data and implement methods making the data more accessible, e.g. a method `getSpecies()` for extracting only the names of the species involved in a simulation can be found in `SimulationResult`.

In the following we give a brief overview of how the different modules can be implemented.

7.4.1 Input parser

Custom parsers can be implemented by extending the abstract class `AbstractParser` and implementing the method `parse()`. We provide a parser for SBML, namely the `SBMLParser`-class which takes input on the SBML form and translate it into a SPN defined by the class `StochasticPetriNet`. The JSBML library, Dräger et al. (2013), has been used to parse the SBML into concrete Java classes, which are used to form the SPN. We will not go into detail with how the SBML-files are parsed, but just give a short review of some of the elements considered in the SBML-format below:

- The `compartment` element (Hucka et al., 2010, p. 40) is ignored as we only consider systems designed in one bacteria cell, thus we will ignore the position of the different species in the cell.
- The `initialAmount` attribute on the `species` element is used as the initial marking of the places in the SPN.
- We assume the SBML input is valid, thus we will not validate that a species identifier used in a reaction is actually present.
- Rate functions on reactions consist of several elements holding different parameters. These are recursively looked up until no more parameters can be found – the remaining parameters are interpreted as unknown variables, typically these will be species concentrations that change over time.

7.4.2 Compiler

Custom compilers can be implemented by extending the abstract class `AbstractCompiler` and implementing the method `compile(StochasticPetriNet spn)`. If errors happen in the compilation a `CompilerException` should be thrown.

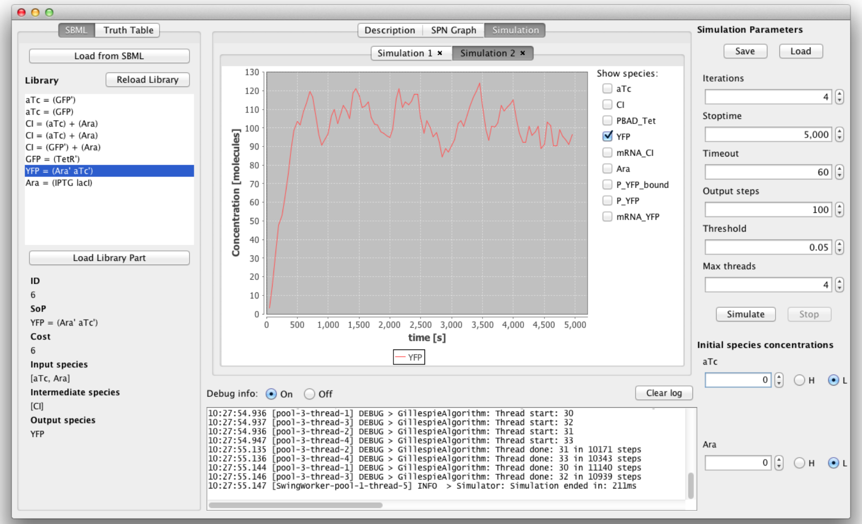
7.4.3 Simulation and algorithms

The `Simulator`-class uses an algorithm specified in the `Parameters`-object. The simulator can run several simulations of the same model at a time by spawning several threads, usually the number of cores on the host will limit the number of concurrent threads. Algorithms should extend the abstract class `Algorithm` which in turn implements the `Runnable`-interface in order to be able to run several instances of the algorithm in parallel. Specifically the `run()`-method needs to implement the details of the algorithm. The input to the algorithm is shared among all instances of the algorithm, hence the input must not be manipulated in the `run()` method.

7.5 Usage

The modularised design of the framework allows several ways of use. The `Main`-class implements the default usage shown as data flow in Fig. 7.1. The `Main`-class can be invoked from the command-line with several arguments: e.g. show debugging information with the argument `-debug` and show a graph with the simulation result with the argument `-graph`.

The framework can also be used as a Java library, the different components of the framework do not depend on each other, meaning that e.g. the `Compiler` can be completely omitted.



Library

aTc = (GFP')
aTc = (GFP)
CI = (aTc) + (Ara)
CI = (aTc) + (Ara)
CI = (GFP') + (Ara)
GFP = (TetR')
YFP = (Ara' aTc')
Ara = (IPTG lacI)

ID
6
SoP
YFP = (Ara' aTc')
Cost
6
Input species
[aTc, Ara]
Intermediate species
[CI]
Output species
YFP

Show species: ☐ aTc, ☐ CI, ☐ PBAD_Tet, ☑ YFP, ☐ mRNA_CI, ☐ Ara, ☐ P_YFP_bound, ☐ P_YFP, ☐ mRNA_YFP

Iterations: 4
Stoptime: 5,000
Timeout: 60
Output steps: 100
Threshold: 0.05
Max threads: 4

aTc: 0 ○ H [✓ L]
Ara: 0 ○ H [✓ L]

Debug info: [✓ On] ○ Off

```
10:27:54.936 [pool-3-thread-1] DEBUG > GillespieAlgorithm: Thread start: 30
10:27:54.937 [pool-3-thread-3] DEBUG > GillespieAlgorithm: Thread start: 32
10:27:54.936 [pool-3-thread-2] DEBUG > GillespieAlgorithm: Thread start: 31
10:27:54.947 [pool-3-thread-4] DEBUG > GillespieAlgorithm: Thread start: 33
10:27:55.135 [pool-3-thread-2] DEBUG > GillespieAlgorithm: Thread done: 31 in 10171 steps
10:27:55.136 [pool-3-thread-4] DEBUG > GillespieAlgorithm: Thread done: 33 in 10343 steps
10:27:55.144 [pool-3-thread-1] DEBUG > GillespieAlgorithm: Thread done: 30 in 11140 steps
10:27:55.146 [pool-3-thread-3] DEBUG > GillespieAlgorithm: Thread done: 32 in 10939 steps
10:27:55.147 [SwingWorker-pool-1-thread-5] INFO  > Simulator: Simulation ended in: 211ms
```

Figure 7.4: This graphical user interface is just a frontend using some of the components of the framework. The *Load from SBML*-button uses the `SBMLParser` and the *Simulate*-button uses the `Simulator` and `GillespieAlgorithm`.

To illustrate how the framework easily can be leveraged as engine in a third-

Input A	Input B	Output
1	1	0
1	0	0
0	1	0
0	0	1

Table 7.1: Truth table for the NOR-gate.

party tool a GUI has been built using the framework as back-end engine, see Fig. 7.4.

We refer to the documentation[2] and code repository[3] for further details on usage.

7.6 Evaluation: Simulating a NOR-gate

In this example we will show the construction of a model of a genetic device with NOR behaviour and show simulations of it using the DTU-SB Framework. Table 7.1 shows the behaviour of a NOR-gate.

Fig. 7.5 is a diagram showing the genetic NOR-gate described in Tamsir et al. (2011). The two inputs arabinose (Ara) and anhydrotetracycline (aTc) positively regulate (induce) the two promoters P_{BAD} and P_{Tet}. The output of the device is the *yellow fluorescent protein* (YFP). The input promoters are placed in tandem, expressed as $P_{BAD} - P_{Tet}$, meaning that the promoters are located right next to each other and that the repressor protein CI is only produced if either or both of these promoters are positively regulated, i.e. they do *not* interfere with each other. The promoter P_{YFP} is repressed by CI, hence the yellow color will only be present if none of the inputs Ara and aTc are present. It should be clear that the NOR-device is composed of two genes, one producing CI and one producing YFP.

The promoters of this device can be interpreted as input, Tamsir et al. (2011) states that the device can be engineered to use other promoters as input, making it possible to use this device in context with other devices. The output as well can be changed by replacing the YFP producing gene with another gene, e.g. one that produces a protein that regulates another promoter. This modularity is greatly desirable, but obviously also imposes some challenges that is out of

[2]`http://jboysen.github.io/dtu-sb-docs`
[3]`https://bitbucket.org/jboysen/dtu-sb`

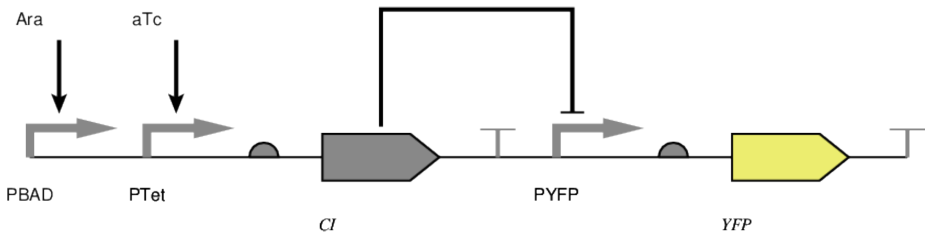


Figure 7.5: Genetic NOR-device. The protein Ara and the antibiotic aTc positively regulate (induce) the promoters PBAD and PTet respectively. If the concentration of Ara and aTc is > 0, the repressor protein is produced and YFP is not produced, otherwise it is.

scope of this thesis – in the next chapter we will discuss this subject further.

Based on the modelling section explaining how to construct a SPN of a gene, we present the model of the genetic NOR-gate in Fig. 7.6. We will not go into detail with the model as it should be self-explanatory and the focus of this section is on simulation – this also means that the rates chosen for the model are not shown.

7.6.1 Simulations

The purpose of this example is to show the output of simulations made with the DTU-SB Framework, and show how different simulation parameters can be changed to receive different output. The graph in Fig. 7.7a shows how several iterations per simulation often give a clearer picture of the behaviour of the simulated device.

7.6.2 Verification of simulations

To verify that the implementation of Gillespie's direct method *stochastic simulation algorithm* (SSA) is correct, we have compared simulations of the NOR-gate made with the DTU-SB Framework to simulations made with Snoopy, Heiner et al. (2012b), an editor/simulator of petri nets. Snoopy implements two SSAs: *fast adaptive uniformization* (FAU) and Gillespie's, of which we will only evaluate our implementation against their implementation of Gillespie's SSA.

In Fig. 7.8 we show the exact same simulations as in Fig. 7.7. These graphs

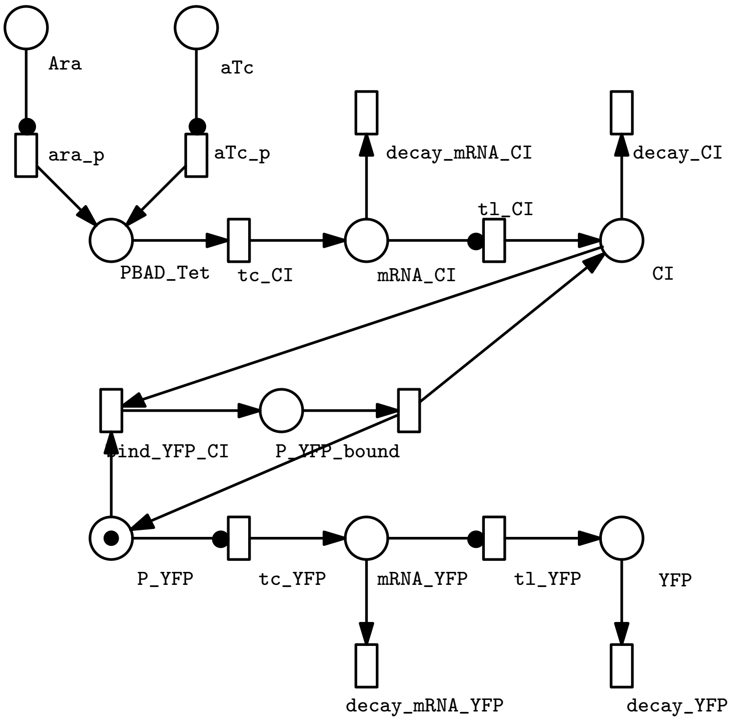


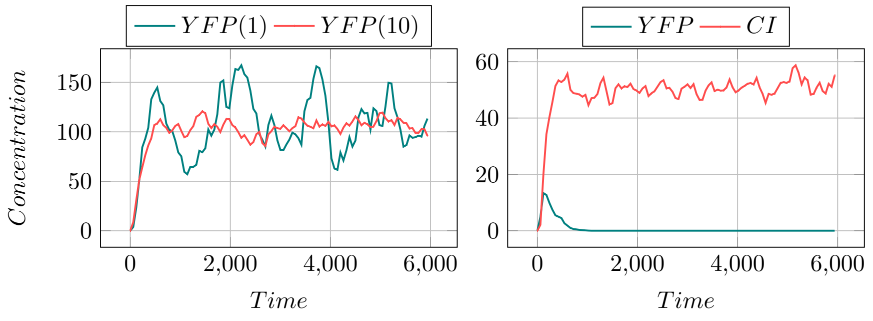
Figure 7.6: SPN of genetic NOR-gate.

clearly shows that there is a coherence between our and Snoopy's implementation. Stochastic simulations can of course not be completely identical, but the overall behaviour is clearly identical, with Snoopy's simulations consequently having concentration levels somewhat under our simulations.

Note that this verification is by no means thorough, as we have presented the formalities of Gillespie's algorithm in previous chapters, and our implementation of it is not efficient but usable.

7.7 Concluding remarks

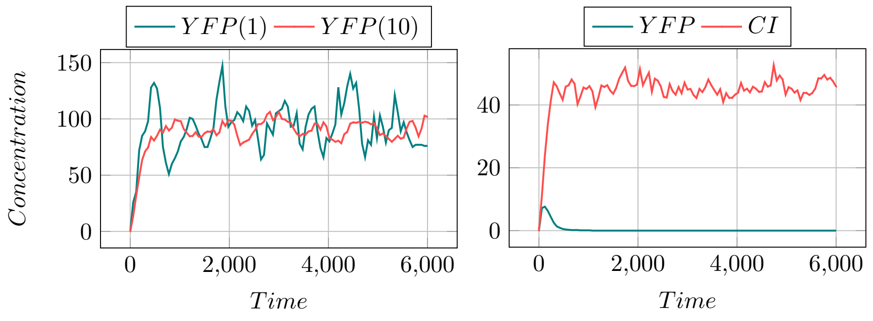
In this chapter we have seen how the *DTU-SB Framework* has been implemented with modularity and extensibility in mind. The framework can be used as library for third-party applications or it can be used as standalone application either by using it from the command-line or by using the graphical



(a) The numbers in the parenthesises in the legends indicate the number of iterations per simulation.

(b) One simulation with 10 iterations.

Figure 7.7: The graph in **(a)** contains two simulations of the NOR-gate with no Ara or aTc present, thus YFP is high.
The graph in **(b)** contains one simulation of the NOR-gate with aTc present, this YFP is low.



(a) This graph is directly comparable to the graph in Fig. 7.7a.

(b) This graph is directly comparable to the graph in Fig. 7.7b.

Figure 7.8: Simulations with Snoopy.

user interface provided. As mentioned the code repository is located at `https://bitbucket.org/jboysen/dtu-sb` and the documentation for the framework can be found at `http://jboysen.github.io/dtu-sb-docs`. To get a head start by avoiding to set up development environments and compilation of java-files the file `DTU-SB-GUI.zip` containing a pre-compiled JAR-file with belonging library of small genetic devices can be downloaded at `https://bitbucket.org/jboysen/dtu-sb/downloads`.

The implementation of Gillespie's direct SSA has been evaluated by comparing simulations of devices with simulations of the same devices using Snoopy. The evaluation is by no means thorough but purely serve the purpose of establishing confidence in our implementation.

In the next chapter Genetic Logic Synthesis will be explained thoroughly, and the results of it has been incorporated into the framework as well, see Appendix D for a tutorial on this matter.

Chapter 8

Genetic Logic Synthesis

The theory established in the preceding chapters will here be utilised to perform *genetic design automation* (GDA) so that systems of genetic devices can be synthesised automatically using a logic function as target and a library of pre-defined genetic devices. The main idea is to perceive each genetic device as a logical gate that can be specified by a truth-table where low and high respectively represent absence or presence of certain proteins . These devices are then combined to express the target-function. Such abstractions are not unrealistic, e.g. Bonnet et al. (2013) have shown how several genetic devices quite precisely followed their predicted logical behaviour.

Perceiving the genetic behaviour logically allows us to reuse the widely developed theory for *electronic design automation* (EDA). Further the desired applications for synthetic biology are something that is often easily expressible as logical problems, for example the biosensor that detects presence of certain combinations of material and takes action accordingly (whether that is emitting fluorescent material signalling the need for external intervention or synthesising material that can react in-place).

Different genes with different promoters can have widely different activation levels why it is not immediately clear what the concentration levels of absence or presence specifically represent for a set of genetic devices. The examples throughout this chapter all assume compatible concentration levels where the general case will be discussed further in Sec. 8.6.

This chapter will first establish a suitable level of abstraction while explaining an overall approach to the problem, then propose a simple prototype method while explaining the similarities to EDA tools and finally discuss how this could be improved.

8.1 Logic synthesis

In electronics, logic synthesis is the process where a high-level description of a design is converted into an optimised composition of gates found in a given library, these gates are typically referred to as *library parts* or just *parts*. The composition is found using a *technology mapper,* and the high-level description can be specified as e.g. a truth-table. Typically some optimisation can be done on the high-level description by minimising the expression.

Both of these steps might output several design candidates, so objective functions are used to evaluate the order from best to worst according to some given criteria. The theory behind minimisation and technology mapping have roots in digital electronics where commercial VHDL and Verilog IDE's such as Altera and Xilinx employ these techniques to either minimise or promote factors such as *cost, delay, die-area, speed* and *fault-tolerancy* of a given chip design. Often the actual synthesis engines used in these tools are both more efficient and more sophisticated than those presented here.

8.2 Genetic design automation

With an overall idea of how logic synthesis proceeds in EDA we outline the key differences that are crucial to take into consideration when designing a GDA synthesis tool:

Compatibility: In electronic circuits the output from one arbitrary gate can always be used as input to another gate. In GDA compatibility between input- and output-proteins of the library parts must be ensured.

Orthogonality: Cross-talk can have devastating effects on circuits of genetic devices, so parts with the same intermediate proteins are not allowed in the circuit and neither should library parts be allowed to be reused except for amplification of protein concentrations.

Size: The designs derived should fit in a single cell. Currently only 20 orthogonal promoters have been identified, Rhodius et al. (2013), which acts as an upper limit of what is possible within a single cell. As our knowledge and models get better this value will most likely increase. If translational regulation using sRNA is considered as well, this limit might be increased as the promoters are not influenced by sRNA.

For the sake of simplicity it is assumed that these three requirements need to be strictly enforced. In reality, the **size** requirement can be relaxed as some models allow inter-communication of cells using chemical signals enabling much richer complexity of circuits, Tamsir et al. (2011). With models taking the timings into consideration it will also be possible to relax the **orthogonality** requirement if guarantees of the behaviour can be established using e.g. a model checker.

8.2.1 A work-flow

The requirements above must be taken into consideration when designing a system for logic synthesis. The diagram in Fig. 8.1 shows the overall work-flow for a GDA synthesis tool.

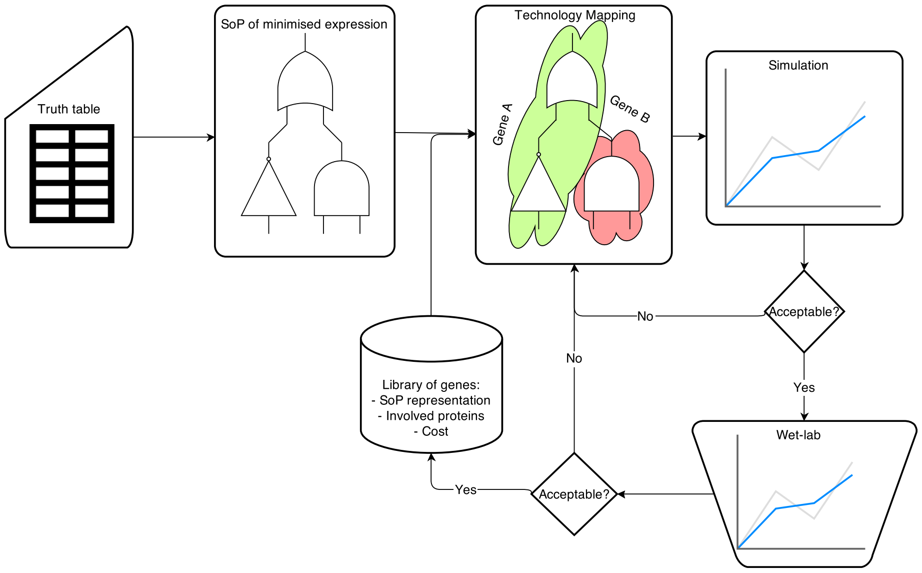


Figure 8.1: Overall design of a system for a GDA synthesis tool.

The input to the system in Fig. 8.1 is a truth-table describing the behaviour of the desired logical function. To ensure a general low complexity of the synthesised devices the input is initially minimised to a minimal *sum-of-products* (SoP) Boolean expression, this expression is given to a technology mapper that will output one or more devices that can be simulated using the DTU-SB Framework. The technology mapper will use the parts defined in the library.

Due to the coarse assumption of considering simulated behaviour as just logical highs and lows as well as the inherit highly stochastic effects of these genetic circuits, the outputted devices cannot be guaranteed to perform as the target function hence validation through simulation is necessary. If the simulation is acceptable a *wet-lab* experiment is carried out and if the result of this is also acceptable the design is saved as a part in the library for later use.

8.2.2 Representation of library parts

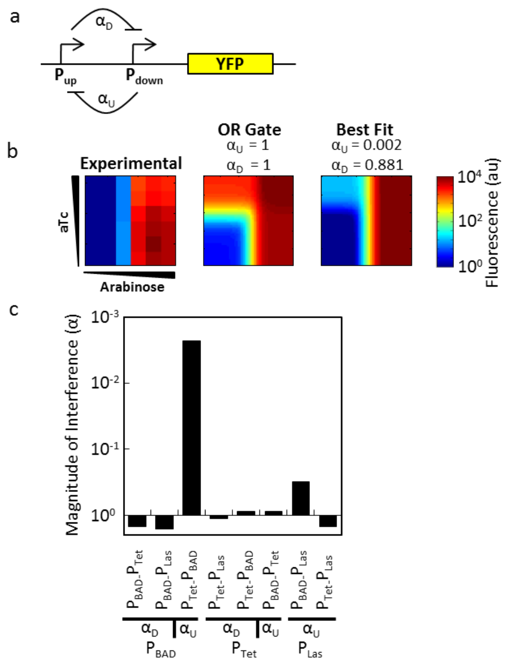
In Sec. 3.3 we saw how a gene can be represented on parts level with promoters, RBSs, PCSs and terminators. There is a close connection between the complexity of the technology mapping and how the parts in the library are represented. In general a library part is just a gene that expresses a logical function with in- and output proteins. Below we outline three different options to how the parts can be represented:

8.2.2.1 Option 1

Abstract away all input and output proteins and populate the database with a few generic genes (i.e. AND and OR) under the assumption that protein interactions always can be established using e.g. the BioBricks interface. This solution enables usage of the existing tools and algorithms used for technology mapping in electronics. The actual gene interactions can then be resolved in a post-processing step. Thus all genes will in theory be compatible.

The drawback is that too many practical details are abstracted away and that this assumption conflicts with some findings indicating that promoters and *protein coding sequences* (PCSs) not always can be replaced without introducing (currently) unpredictable or unwanted behaviour, e.g. Tamsir et al. (2011) shows that a certain combination of promoters placed in tandem do not function as predicted, see Fig. 8.2 for an explanation.

Option 1 is used by Weiss et al. (1999) and partly by Marchisio and Stelling



a) The diagram shows how two promoters placed in tandem with one promoter upstream, P_{up}, and one promoter downstream, P_{down}, can repress (or interfere with) each other, and thereby not exhibit additive transcription. The α_D and α_U are factors in a linear transfer function on the form $X = \alpha_U X_U^{max} P_U + \alpha_D X_D^{max} P_D$ describing the amount of protein generated, where X_i^{max} is the maximum protein generation from the ith promoter at steady-state and P_i is the probability for the ith promoter being ready to transcribe.
b) The *Experimental* graph shows the behaviour of a gene with the tandem promoter $P_{Tet} - P_{BAD}$. The desired behaviour is shown in the *OR Gate* graph, where the promoters do not affect each other.
c) The diagram shows how the interference is almost non-existent for all tandem promoters except for the $P_{Tet} - P_{BAD}$ tandem promoter.

Figure 8.2: *Figure from* (Tamsir et al., 2011, Supplementary Figure 3).

(2011), meaning they have an unlimited number of available logical gates enabling easy realisation of new devices.

8.2.2.2 Option 2

Create models on parts level, i.e. promoter, RBS, PCS and terminator which can be put in arbitrary order. This solution will have a very high expressiveness but will share many of the drawbacks of option 1 and will make the problem of finding correct parameters even more noticeable.

Option 2 is also used by Marchisio and Stelling (2011) and gives the same advantage as of **option 1**. **Option 1** and **2** are theoretically optimal, but to work in practice it will require much more precise models than what is available today. Once more precise models are developed these options will clearly be advantageous.

8.2.2.3 Option 3

Use only genetic logic gates that have been successfully realised in wet-lab experiments. Due to the fewer protein compositions that need to be established this will incur only a minimum of unpredictability to the realised design candidates. The drawback is that we only have a very limited library of parts and need to tailor the technology mapping phase.

Option 3 takes the current lack of a precise models into account by disallowing unspecified/generic devices and genes that have not been proven to work in isolation. To the best of our knowledge this approach have not yet been proposed. We have chosen **option 3** with the belief that the need for doing logic synthesis could arise before a model accurate enough to support **option 1** or **2** can be established.

8.2.3 Technology mapping strategy

Technology mapping is crucial for GDA synthesis. There are several options on how to do this which are outlined and discussed below:

1. **Implement a simple technology mapping algorithm:** Possibly by modifying existing algorithms used in technology mapping tools for EDA to enforce compatibility, orthogonality and size. This approach requires much effort, but will potentially be very efficient and precise depending on the actual implementation. It will also ease future changes or relaxations of the three requirements. It should be noted that altering existing algorithms may invalidate their optimality properties and should be formally verified after each alteration.
2. **Use existing technology mapping tool for electronics:** Use an existing tool to output all possible solution candidates and remove those candidates violating the compatibility, orthogonality and size constraints in a post-processing step. The state space that needs to be explored can here be much larger than for the other two alternatives. A more practical

problem with this approach is that the seemingly only free and mature tool, ABC, only outputs the best candidate and have no options to emit all possible designs thus making this approach impractical.

3. **Formulate the problem to be interpreted by a solver:** These types of problems can typically be specified as *constraint satisfaction problems* (CSPs) or *Boolean satisfiability problems* (SATs). Though most solvers are very efficient, it is very difficult to predict the performance of solving a particular problem potentially making the problems intractable. Some of the properties, such as how to capture the structure of a Boolean expression can be very hard to formulate as a CSP or SAT.

In order to get a very efficient solution with complete transparency for further extensions we have chosen **option 1**; to implement a simple technology mapping algorithm by tailoring existing EDA algorithms.

8.3 Library based technology mapping

Given a logic function f, technology mapping is the process of composing a layout of logic gates from a library $\mathbb{L}$ to express f. Often several solutions exists to this kind of problem why this process is instructed by an objective function to find the best solution.

This is a four-step procedure (heuristic):

1. **Minimisation:** Represent f as a minimal SoP.
2. **Decomposition:** Represent the minimised SoP as an *and-inverter graph* (AIG).
3. **Partitioning:** Translate the AIG into a forest of single output AIGs.
4. **Covering:** Cover each of these AIGs with parts from $\mathbb{L}$.

Several minimal expressions may exists where we – due to the increased difficulty of finding valid designs in GDA mapping compared to EDA mapping – need to examine all minimal forms of a given function.

8.3.1 Minimisation

Several methods for minimising Boolean equations have been developed. Some of the best know are the exact but inefficient *Quine-McCluskey algorithm*[1] from Mccluskey (1956) and the inexact though efficient *Espresso heuristic* from Brayton et al. (1984). Here exactness refers to the ability of finding the minimal expression in terms of literals and offers a trade-off where larger chip designs often have to rely on "near optimum algorithms" to be tractable to layout. Muroga (1979) surveys many of the classical minimisation methods. As will be shown in Sec. 8.3.2 the complexity of the synthesised system is very much dependent on the format of the used SoP. In GDA where it is crucial to keep the general complexity at a minimum selecting a good minimisation procedure is of great importance.

As these genetic systems often will be quite simple, at least compared to the complexity of micro-chips, we will rely on the exact *Quine-McCluskey* algorithm which is a three-step procedure:

1. Produce a minterm expansion (SoP form) of the function f.

2. Eliminate as many literals as possible by systematically applying $XY + XY' = X$ to obtain the prime implicants.

3. Use a *prime implicant chart* to select a minimum set of prime implicant that when ORed together produce f, and that contains a minimum number of literals.

The following explanation of these steps is greatly inspired by Frenzel using the truth-table below as example:

[1]Originally named *The method of prime implicants* but also referred to as *The Tabular Method of Minimisation.*

	A	B	C	D	f
m_0	0	0	0	0	0
m_1	0	0	0	1	0
m_2	0	0	1	0	0
m_3	0	0	1	1	0
m_4	0	1	0	0	1
m_5	0	1	0	1	0
m_6	0	1	1	0	0
m_7	0	1	1	1	0
m_8	1	0	0	0	1
m_9	1	0	0	1	-
m_{10}	1	0	1	0	1
m_{11}	1	0	1	1	1
m_{12}	1	1	0	0	1
m_{13}	1	1	0	1	0
m_{14}	1	1	1	0	-
m_{15}	1	1	1	1	1

Where each row is a called a minterm $m_0 - m_{15}$ and where "-" means *don't care* and is interpreted as f is allowed to be either `true` or `false`. The term *implicant* is used to describe contractions of one or more minterms. A truth-table is often specified in the equivalent but more compact representation:

$$f(A, B, C, D) = \sum m(4, 8, 10, 11, 12, 15) + \sum d(9, 14)$$

1. Minterm expansion

The minterm expansion SoP of this expression is obtained by simply selecting each `true` minterm:

$$f_{A,B,C,D} = A'BC'D' + AB'C'D' + AB'CD' + AB'CD + ABC'D' + ABCD'$$

2. Identifying the Prime Implicants

In order to minimise this expression all `true` and `don't care` minterms are put in a *minterm table* which is a table grouped on the number of 1s in the minterm:

Group	Minterm	Representation
1	m_4	0 1 0 0
	m_8	1 0 0 0
2	m_9	1 0 0 1
	m_{10}	1 0 1 0
	m_{12}	1 1 0 0
3	m_{11}	1 0 1 1
	m_{14}	1 1 1 0
4	m_{15}	1 1 1 1

The rule $XY + XY' = X$ is applied repeatedly to create as large contractions as possible by comparing the implicants that only differs in a single bit position. For example the two minterms $m_4 = 0100$ and $m_{12} = 1100$ only differs in the most significant bit, thus it can be represented by the single implicant $m_4, m_{12} = -100$. The minterm table grouping makes applying this rule efficient as only the neighbour group to each group needs to be checked.

The first stage of contraction yields:

Implicant	Representation
m_4, m_{12}	- 1 0 0
m_8, m_9	1 0 0 -
m_8, m_{10}	1 0 - 0
m_8, m_{12}	1 - 0 0
m_9, m_{11}	1 0 - 1
m_{10}, m_{11}	1 0 1 -
m_{10}, m_{14}	1 - 1 0
m_{12}, m_{14}	1 1 - 0
m_{11}, m_{15}	1 - 1 1
m_{14}, m_{15}	1 1 1 -

And the second stage, where each of the first stage contractions are tried reduced further yields:

Implicant	Representation
m_8, m_9, m_{10}, m_{11}	1 0 - -
$m_8, m_{10}, m_{12}, m_{14}$	1 - - 0
$m_{10}, m_{11}, m_{14}, m_{15}$	1 - 1 -

These stages continue until no new reductions can be made. In this case no new reductions can be made after the second stage.

Now the *prime implicants*, which is all the implicants that cannot be reduced further, simply can be identified by selecting implicants from the highest possible stage until all minterms have been covered. For example none of the second stage implicants cover m_4 hence the m_4, m_{12} implicant from the first stage is selected:

$$f_{A,B,C,D} = AB' + AD' + AC + BC'D'$$

3. Selecting the minimum set of Prime Implicants

The current expression is already greatly reduced compared to the minterm expansion SoP, but it is possible to reduce it further using a *prime implicant chart* that is used to analyse which prime implicants cover the necessary minterms, ignoring `don't care` minterms, i.e. m_9 and m_{14}.

	m_4	m_8	m_{10}	m_{11}	m_{12}	m_{15}
AB'		X	X	X		
AD'		X	X		X	
AC			X	X		X
$BC'D'$	X				X	

If a minterm is covered by only one prime implicant, that prime implicant is called an *essential prime implicant* and have to be a part of the minimal expression, thus these are identified first. In this case the $BC'D'$ and AC are identified because only m_4 and m_{15} cover the minterms respectively. These two implicants cover in total the minterms $m_4, m_{10}, m_{11}, m_{12}, m_{15}$ where the remaining minterm m_8 should be covered using the remaining prime implicants. Here both AD' and AB' cover m_8 and are therefore both valid candidates. Thus the two minimal SoPs become:

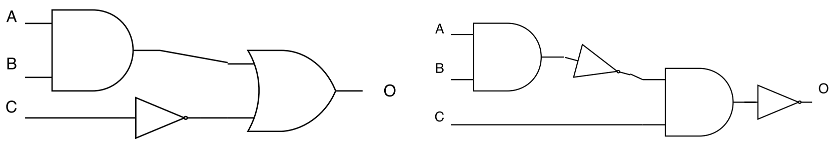
$$\begin{aligned} f_{1(A,B,C,D)} &= AB' + AC + BC'D' \\ f_{2(A,B,C,D)} &= AD' + AC + BC'D' \end{aligned}$$

In more advanced examples, selecting the minimal set of prime implicants covering all remaining minterms is not necessarily trivial. These situations are typically resolved using either an exhaustive approach or using *Petrick's method* introduced in Mccluskey (1956).

8.3.2 Decomposition

The minimal SoP is translated into an AIG as this is the *de facto* standard data-structure for EDA based technology mapping. The popularity of this representation stems from its very efficient mapping algorithms as well as ability to easily perform optimisations by rewriting.

An AIG is a directed, acyclic graph that by its structure implements logical expressions. An AIG only have two types of nodes; namely the two-input *And*-node as well as the *Inverter*-node. Creating an AIG-representation of f is relatively straightforward using *DeMorgan's theorem* and basic propositional algebra, informally summarised as *every AND-gate can be replaced by an OR-gate with inverters on all in- and outputs.* See an example in Fig. 8.3.



(a) Regular layout **(b)** In AIG

Figure 8.3: Equivalent representations of $O = (AB) + C'$

Unfortunately these kinds of translations are non-canonical when the sum has more than two products or these products have more than two literals. Consequently the AIG is not as general as the SoP since an ordering is imposed upon how to convert N-input gates to 2-input gates. Fig. 8.4 illustrates this. This can cause the pattern-matching to not necessarily find viable designs. Within the world of digital electronics this fact is typically only of minor concern as these situations typically are solved by just using the layout leading to the shortest delay (or another objective). Such criteria can here be stated as there typically is an enormous amount of reusable gates available which can be arranged in any order.

Synthesis of genetic systems is somewhat different as we usually have a very limited number of parts that only can be arranged in specific arrangements due to the constraining protein-interactions and orthogonality requirements presented in Sec. 8.2. Henceforth each possible permutation of the AIG should be obtained and tried matched, where each gate with a fan-in of more than 2 causes exponential increase in the amount different AIGs. For larger systems this can lead to an intractable amount of matching where the canonical *AIG-With Choice* representation by Chatterjee (2007) could prove beneficial as this is specifically developed to overcome this problem by enabling multiple AIGs to be encoded in a singe AIG-With Choice by detecting and storing functional equivalences for each node. Further discussion of the benefits of using different data-structure can be found in Sec. 8.6.

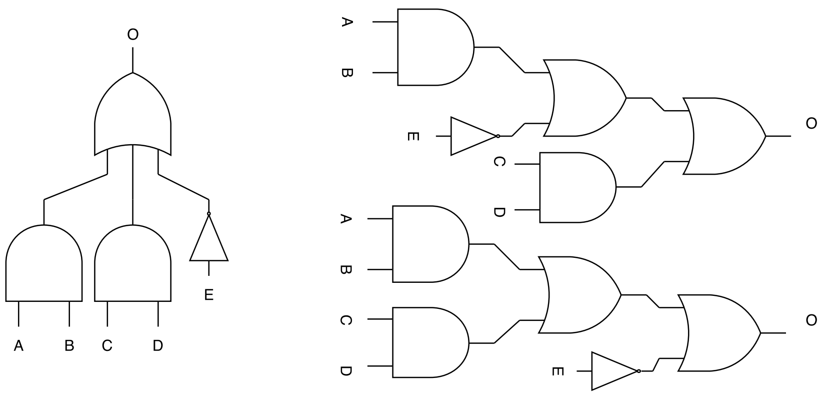


Figure 8.4: *Left)* Canonical representation of a 3-input OR-gate
Right) Two examples of possible conversions using 2-input OR-gates (non-canonical).

8.3.3 Partitioning

As the output of f is not limited to a single output protein[2], the corresponding AIG does not necessarily need to be a tree which makes the problem of matching $\mathbb{NP}$-hard. As a result the AIG is partitioned into a forest of trees allowing seamless top-down matching that can be solved within polynomial time, Keutzer (1987).

A trivial partition strategy is to break the graph at all multiple fanout points and introduce intermediate inputs as needed, see example in Fig. 8.5. This approach is quite natural for GDA as the intermediate input introduced corresponds to intermediate proteins that needs to be mapped anyway. Unfortunately this can result in a lot of small trees, which in general cannot be covered as well (with respect to an objective function) as a few larger trees can, Alpert and Kahng (1995). More sophisticated strategies, such as the *single-cone partitioning*, duplicate some of the logic in-order to obtain as few trees as possible – but it is not directly clear whether this approach is applicable in GDA partitioning as the library parts, as opposed to EDA, are large indivisible entities. An extensive survey of partitioning strategies has been conducted by Alpert and Kahng (1995) and could serve as inspiration for further research.

[2]In which case the minimisation procedure gets somewhat more complex than otherwise explained in Sec. 8.3.1. Techniques to solve this can be seen in e.g. Jiang et al. (2002). For any practical purposes one can think of the minimised result as a set of SoP expressions

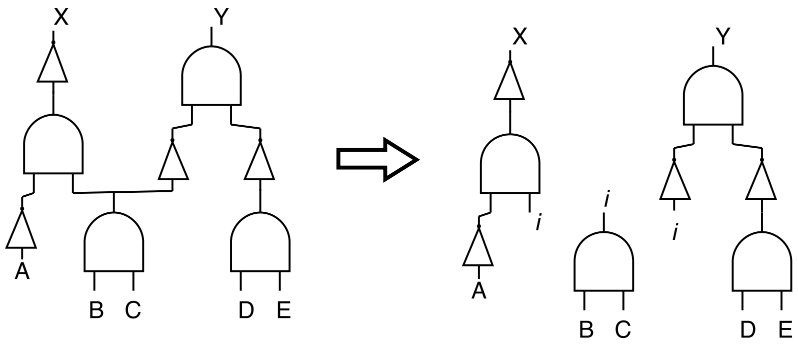


Figure 8.5: Example of naive partitioning strategy. The AIG is partitioned into three AIG-trees at the multiple fan-out node by introducing the intermediate input i.

It should be clear that a general limitation to partitioning is that a subset of exact solutions to a partitioned problem do not guarantee an exact solution to the problem itself.

8.3.4 Covering

The last step is to cover each of the trees using the parts from $\mathbb{L}$. This happens recursively using Algorithm 2 by initiating it on the output-node. As mentioned previously this covering step is different from that of digital electronics, as orthogonality and protein-compatibility also have to be ensured. This is reflected in the library part selection method. An *unmatched branch* "b" refers to the sub-tree spawning from node b. The algorithm relies on a procedure to determine whether a given part can be placed at a branch b. Given the AIG structure this is basically just a matter of doing tree pattern-matching.

An example of this algorithm can be seen in Fig. 8.6. This example is somewhat simplified by just showing how to find a single structural solution.

In EDA it is very common to optimise the AIG by e.g. applying *structural hashing* that identifies repeated patterns and compact accordingly. Due to the quite strict orthogonality enforcement these repeated patterns will never occur but one could imagine these requirements being relaxed in the future thus making *structural hashing* applicable to GDA technology mapping as well.

Algorithm 2 Technology Mapping

```
procedure MATCH($B$)                              ▷ $B$ is a set of unmatched branches
    for Each unmatched branch $b \in B$ do
        for Each compatible and orthogonal part $l \in \mathbb{L}$ do
            if $l$ can be placed at $b$ then
                $B' \leftarrow$ unmatched branches from placing $l$ at $b$.
                MATCH($B'$)                       ▷ Determines whether $b$ can be matched
            end if
        end for
    end for
    if all unmatched branches in $B$ could be matched then
        Record the set of used library parts.
        return $B$ could be matched.
    else
        return $B$ could not be matched.
    end if
end procedure
```

8.4 Characterisation and evaluation

Characterising library parts is an important discipline as these characterisations are used by the technology mapper. The devices found by technology mapping should be evaluated primarily with respect to how well they behave like the input truth-table. The evaluation could be done either manually by inspecting time-series or automatically by making calculations on the time-series, the result hereof should obviously be used in the characterisation of library parts as well.

8.4.1 Characteristics

A library part is typically one or more genes composed to express some Boolean function. Library parts have some characteristics:

- Number of promoters.
- Number of involved proteins.
- Well defined input-, intermediate- and output-proteins.
- A behaviour expressed by a Boolean function or a truth-table.
- The protein-levels considered as logical high and low.

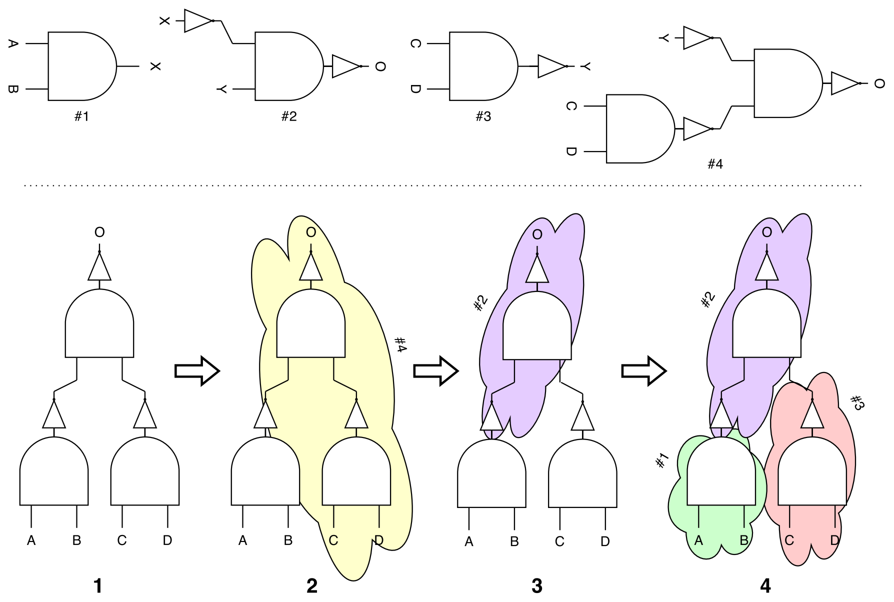


Figure 8.6: *Top)* Library with #1: $X = (AB)$, #2: $O = (X) + (Y')$, #3: $Y = (C') + (D')$ and #4: $O = (Y') + (CD)$.
Bottom) Simplified mapping process of **1:** $O = (AB) + (CD)$.
2: #2 and #4 can be placed at the top, try #4. Unfortunately there are no further parts that can complete the partial match as the output x of #1 is not compatible with the input y of #4.
3: Roll back and try #2 instead, due to the greedy nature there are two possible placements of the children of #2 so both placements are recursively tried matched.
4: With #2 at the top and the "correct placement of its children", #1 and #3 completes the mapping by propagating the inverters to the input-proteins and makes up a valid solution.

- At what time-point the library part can be considered steady, i.e. where the output-protein level is steady enough to decide whether the input proteins yield a high or low output.

The NOR-gate in Sec. 7.6 from the Implementation Chapter is an example of a device that could function as a library part as all of the characteristics above can be identified by inspecting the model and the simulations of it. On the other hand the oscillator device (Fig. 6.2 and Sec. 6.4) used to construct the negative feedback device in Ch. 6 on Modelling cannot function as a library part due to no steady-state can be found, see Fig. 8.7.

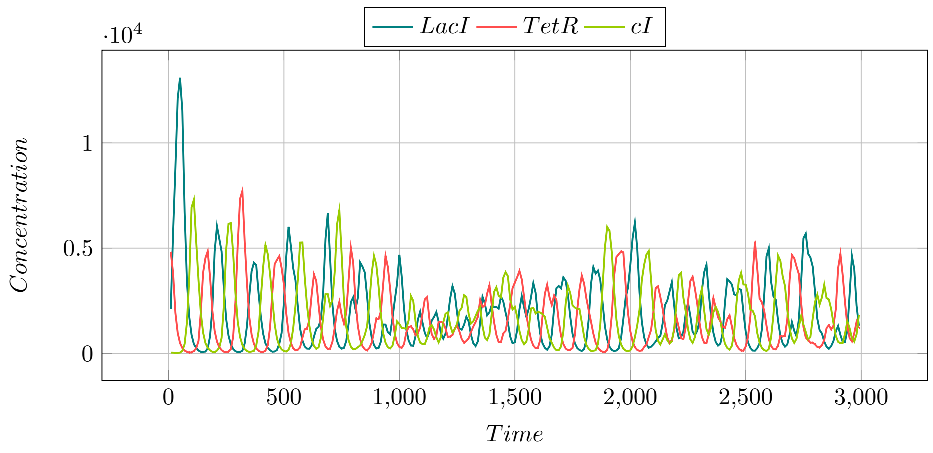


Figure 8.7: Output of the oscillator device. There is no steady-state, thus the output cannot be described by Boolean logic as opposed to the device giving rise to the behaviour in Fig. 8.8.

The characteristics above indicate that the protein levels are very important when parts are evaluated, mainly the possibility to distinguish steady high and low levels and the time-span where levels are still not steady. Marchisio and Stelling (2011) is used to define some of these characteristics: *Signal separation* (σ) is the absolute difference between output concentration levels of the minimal logical 1 steady-state ($min1$) and maximal logical 0 steady-state ($max0$), see Fig. 8.8. Signal separation has to reach a certain level to be practically detectable in wet-lab experiments.

Note that the time-series in the figure are just examples of five different simulations of a genetic device: in three of the simulations the inputs cause high output and in the remaining two they cause low output. The behaviour of the low outputs can be caused by the design of the genetic device: output is expressed but after some time the inputs to the device cause expression of an intermediate protein which in turn represses expression of the output.

One last characteristic of a model is the settling time illustrated in Fig. 8.8 as well. Before this point the output is still not steady, consequently this point can serve as guideline to the minimum length of simulations.

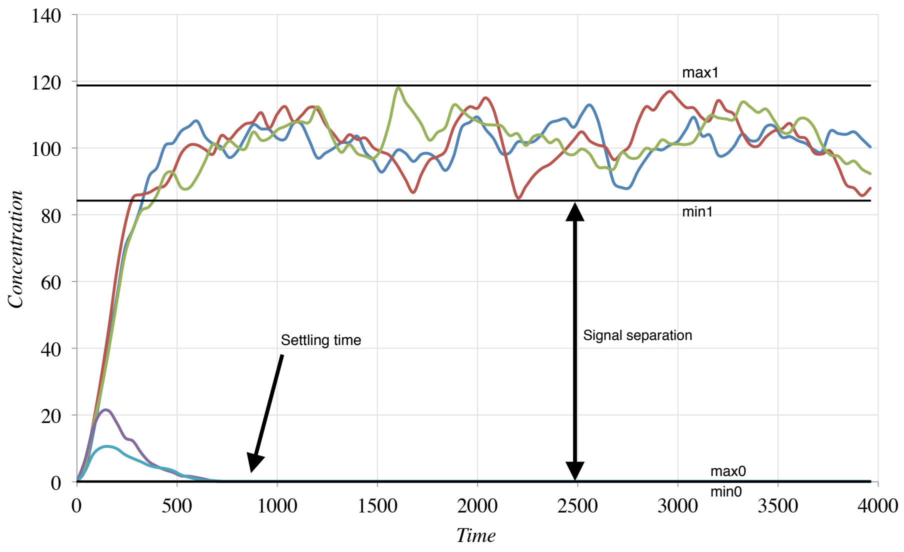


Figure 8.8: All logical 1 outputs are between the lines `max1` and `min1`, and all logical 0 outputs are just 0, i.e. `min0` = `max0` = 0. The settling time is where the concentration levels will not change significantly anymore, regardless of whether high or low output is observed. *Figure altered from Marchisio and Stelling (2011).*

8.4.2 Objective function

The technology mapper needs to differentiate between several parts with the same Boolean function, thus each part in the library need to be evaluated based on some parameters. Marchisio and Stelling (2011) discusses how to achieve this, below we summarise and adapt some of these techniques. Furthermore the objective function is also used to rank designs proposed by the technology mapper, thus in the following the objective function can be used to both rank existing library parts and new designs.

The score is used to find the best design if several designs are proposed by the technology mapping:

$$S = 2^{N_R^{-1}} + 2^{N_A^{-1}} + N_l + N_k \tag{8.1}$$

Here N_R, N_A, N_l and N_k is the total number of repressor promoters, activator promoters, locks and keys respectively. The locks and keys are sRNA base-pairing with mRNA, i.e. translational regulation of gene expression, refer to Sec. 2.4. The score S represents the complexity or the realisability of the part design, not the quality, thus a high score will mean a lower complexity of part design realisability.

It should be clear that the score S from Eq. (8.1) reflects that regulation through locks and keys is better than through protein interactions. We redefine the score S above and call it the cost C of realising a design or part:

$$C = 2^{N_R} + 2^{N_A} + N_{IMP} \tag{8.2}$$

Here we have introduced the new variable N_{IMP} representing the number of intermediate proteins. Further it should be obvious that, opposed to the score S, the lower the cost the lower realisabilty complexity of the part. As we only employ regulation via protein interaction, the number of locks and keys from the score S in Eq. (8.1) are removed from the cost C in Eq. (8.2). Further it should be noted that with better understanding of the complexity of realising the designs in the laboratory, the objective function can be updated accordingly.

When the cost is dependant on the number of intermediate proteins designs composed of few library parts are automatically favored, as there will be fewer internal protein interactions, thus less internal complexity.

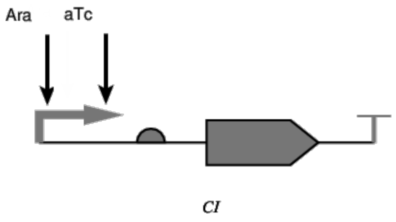
8.4.3 Evaluation

The purpose of evaluating designs is two-fold: *1)* decide whether the design is actually exhibiting the behaviour described by the truth-table and *2)* find the the characteristics of the new design. Many of the characteristics can be found by just looking at the new design, e.g. by counting the intermediate proteins, etc.

If a library part should be evaluated we already know some of its characteristics and can assume the behaviour of the part follows the truth-table attached to it. What is interesting is at what input concentration levels the part will function as described. Many of the parts used in the prototype shown in Sec. 8.5 have *logical high* concentration levels at approximately 100 and *logical low* concentration levels at 0, but as we shall see in the next section these levels can vary due to the different designs of the parts and the genes composing these parts.

8.4.3.1 Case study: Evaluating OR-gates

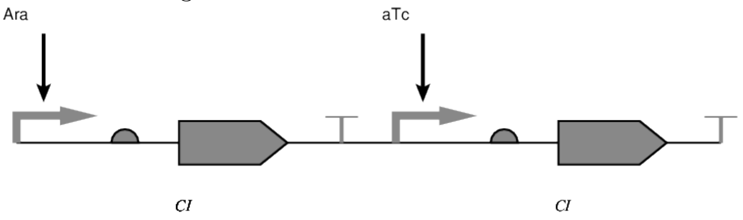
The prototype outlined in Sec. 8.5 contains a library with different parts, their characteristics according to the previous sections and SBML models. In this section we will investigate three OR-gates and see how different modelling techniques often cause different behaviour. Further we will explain how characteristics can be identified and behaviour verified. The OR-gates will be referred to as OR-gate #1, #2 and #11[3]. The genetic devices and their behaviour can be seen in Fig. 8.9.



(a) A single gene expressing CI, the promoter is induced by either aTc or Ara. OR-gate #11 conforms to this design.

aTc	Ara	CI
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth-table describing the overall behaviour of the devices in **(a)** and **(c)**.



(c) Two genes expressing CI, induced by aTc or Ara respectively. OR-gate #1 and #2 conform to this design.

Figure 8.9

We evaluate the OR-gates by looking at the output graphs. We know their behaviour is correct, but we need to define the concentration levels. In Fig. 8.10 the behaviour of the gates when Ara and aTc are present is outlined. We can clearly see that gates #2 and #11 reach a steady-state approximately at time 1000, whereas gate #1 is a bit slower and reaches a steady-state approximately at time 4000.

In Fig. 8.10 we can see the concentration levels when Ara and aTc are high. We need to be more thorough and identify all possible concentration levels, thus we need to simulate all rows of the truth-table. In Fig. 8.11 we have simulated all rows of the truth-table for OR-gate #1, thus we can identify $min1 \approx 37$ and $max0 = 0$, this means all CI concentration levels > 37 can be considered high. The different concentration levels is a result of the gate design shown in Fig. 8.9c, where two independent genes express CI when their promoter is activated.

OR-gate #2 is designed equivalently to OR-gate #1. In Fig. 8.12 we investigate what happens if the inputs are between low and high. The graph is composed

[3]The numbers represent the IDs of the parts in the small library created for the GDA tool-chain prototype.

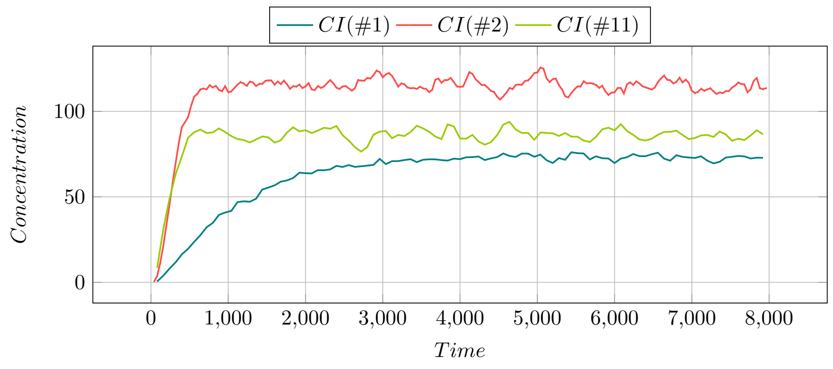
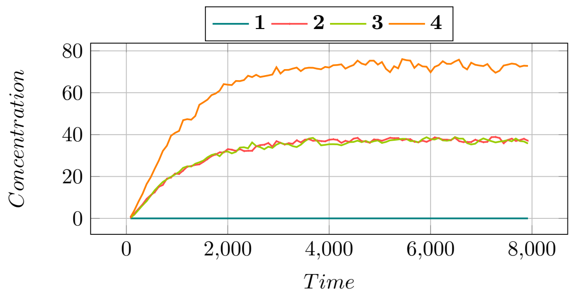


Figure 8.10: OR-gates simulated using the DTU-SB framework. *Ara* and *aTc* inputs are high, each simulation is repated 12 times. The numbers in the parentheses represent the ID of the gate associated with the time-series.

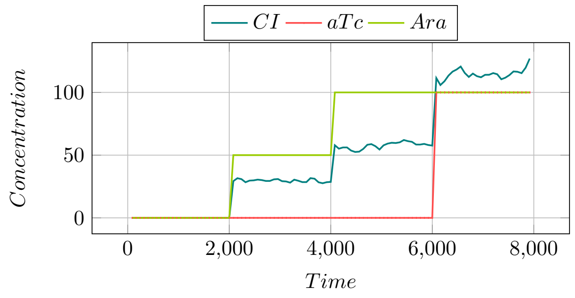


	aTc	*Ara*	*CI*
1	0	0	0
2	0	100	~37
3	100	0	~37
4	100	100	~75

Figure 8.11: OR-gate #1. Here we can see how the concentration of *CI* is stronger when both inputs *aTc* and *Ara* are present.

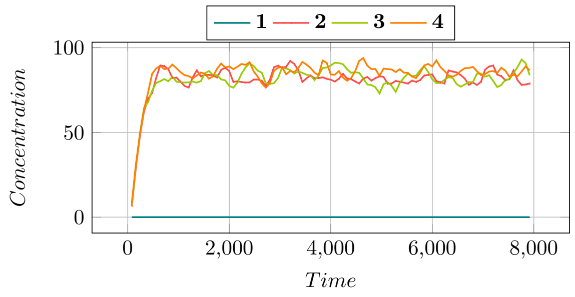
of 4 simulations. If *aTc* and *Ara* were injected during the simulations the concentration levels would change more smoothly. The table on the right summarises the behaviour where each row corresponds to the time-intervals [0; 2000],]2000; 4000],]4000; 6000] and]6000; 8000] respectively. From this graph it should be clear that correct concentration levels need to be identified to correctly decide whether a gate design corresponds to the expected behavior.

The last OR-gate #11 evaluated has more unambiguous outputs with $min1 = 80$ and $max0 = 0$. OR-gate #11 is only composed of one gene, see design in Fig. 8.9a, this gives the same output concentration levels regardless of the input. Furthermore this OR-gate only exhibits *CI* expression when the input levels are > 100, i.e. the behaviour of OR-gate #2 cannot be exhibited.



aTc	Ara	CI
0	0	0
0	50	∼30
0	100	∼60
100	100	∼120

Figure 8.12: OR-gate #2. Here we can see that CI is present even at smaller amounts of Ara.



	aTc	Ara	CI
1	0	0	0
2	0	100	∼80
3	100	0	∼80
4	100	100	∼80

Figure 8.13: OR-gate #11. Here we can see that the concentration level of CI is either steady at 0 or approximately 80 when the inputs are absent or present respectively.

In Table 8.1 the characteristics and behaviour of the three gates are summarised.

	OR-gate #1	OR-gate #2	OR-gate #11
#Promoters	2	2	1
Intermediate	∅		
Input	$\{aTc, Ara\}$		
Output	CI		
$max0$	0		
$min1$	∼40	∼60	∼80
Settling time	∼4000	∼1000	∼1000
Cost	6	6	4

Table 8.1: Characteristics of the three OR-gates evaluated.

8.4.3.2 Automatic evaluation

Automising the evaluation process is an important step in the GDA tool-chain. Evaluation – both automatic and manual – of new designs should follow a certain approach described below.

1. Simulate all rows of the truth-table.
2. For each simulation smooth the output graph to remove fluctuations and highlight trends. This process will have less impact the more iterations per simulation.
3. Identify change-points, i.e. where concentration levels change drastically within short time, refer to *settling time* in Fig. 8.8.
4. Decide from the number of change-points identified whether the design has a steady-state or not:
   Has steady-state: Concentration levels should be characterised and compared to the expected output, i.e. does the design follow the behaviour specified by the truth-table.
   No steady-state: The design should be discarded.

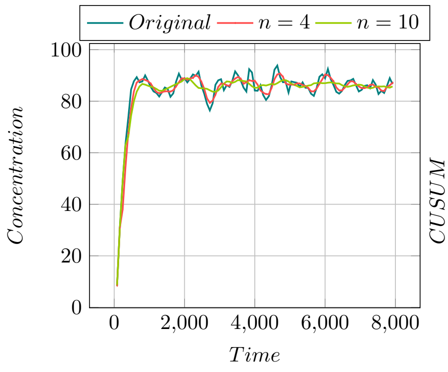
The first step is straight-forward, but the subsequent steps should be completed by employing sophisticated methods. In the following we will show a naïve approach to implementing automatic evaluation, this approach does not guarantee that identified characteristics are correct. In the discussion in Sec. 8.6 we refer to more sophisticated methods for doing automatic evaluation.

Step 2: Smoothing graph

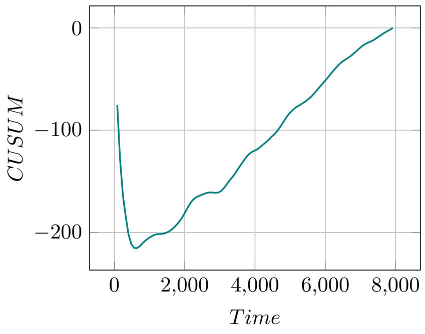
Simple moving average (SMA) is a simple method for smoothing graphs. SMA is the unweighted mean of the last n data points $\{X_1, X_2, ..., X_n\}$ in a time-series. To accommodate for the shift in time, often the mean is taken from an equal amount of data around the central data point:

$$SMA = \frac{X_{1-n/2} + X_{2-n/2} + ... + X_{n-n/2}}{n}$$

The larger n the more fluctuations are removed. Fig. 8.14a is OR-gate #11 with high output and two SMAs with different n applied.



(a) OR-gate #11 with high output. The two SMA graphs with $n = 4$ and $n = 10$ illustrate how more fluctuations are removed as n gets larger.



(b) A CUSUM chart of the SMA $n = 10$ graph.

Figure 8.14

Step 3: Identify change-points

To identify change-points the change in concentration per time should be observed, we assume that huge changes indicate change-points. A *cumulative sum* (CUSUM) chart can be used to identify these points, Taylor (2000). The cumulative sums $S_0, S_1, ..., S_n$ are calculated using the data-points $X_1, X_2, ..., X_n$ (here n represents *all* data-points):

1. Calculate the average $\overline{X}$.
2. Set $S_0 = 0$.
3. Calculate S_i by summing the previous cumulative sum and the difference between the current data-point and the average: $S_i = S_{i-1} + (X_i - \overline{X})$

Refer to Fig. 8.14b for the CUSUM chart based on the SMA data-points for OR-gate #11 with $n = 10$. The data-points used for the graphs in Fig. 8.14 and the corresponding SMAs and CUSUM can be found in Appendix C. The average used for calculations of CUSUM is $\overline{X} = 83.87$.

A section with downward slope on CUSUM charts indicates that the data is below average at that point, and a section with upward slope indicates data above the average. A sudden change in direction indicates a change-point. Fig. 8.14b clearly indicates a change-point at time ~ 700, but we cannot really be sure that this is the only change-point. A *bootstrap analysis* can be carried out to establish a confidence level for the change-point.

A bootstrap analysis reorders the data-points randomly and CUSUM charts are created from these orderings. The idea is that these CUSUM charts most likely will contain no significant change-points, thus the CUSUM chart for a bootstrapping sample will be more horisontal flat. The difference S_{diff} between the minimum S_i and maximum S_i for $i = \{1, 2, ..., n\}$ for the original data-points can be used as estimator of the change. For each bootstrapping sample, let S_i^0 be the CUSUMs and S_{diff}^0 be the equivalent to S_{diff} for the original data-points. Now for each bootstrapping sample where $S_{diff}^0 < S_{diff}$ we are confirmed that the original data-points have a change-point.

To establish a confidence-level typically around 1000 bootstrapping samples should be investigated. The confidence-level can be established by:

$$100 \cdot \frac{N}{1000}\%$$

Where N is the number of bootstrapping samples where $S_{diff}^0 < S_{diff}$. Typically a confidence-level above 90-95% is required to be sure that a significant change occured.

A simple method to identify the change-point from this data is by finding the absolute maximum cumulative sum:

$$|S_m| = \max_{i=0,\ldots,n} |S_i|$$

For OR-gate #11 this corresponds to $time = 640$ where $m = 8$ and $|S_m| = 215.24$. Now this step should be repeated by splitting the data by the change-point identified into two sets and perform this analysis on both sets again. When no more change-points can be identified proceed with the next step.

Step 4: Characterise design

A part or design with well-defined input and output can have a different number of change-points (with the assumption that all output concentrations start at 0):

0: No change at all: steady-low.

1: Changing from rising to steady-high, assuming the rising starts immediately at time 0.

2: Changing from rising to falling and steady-low, rising for a short period and then repressed.

The above only applies in best-case simulations, one could easily imagine that devices with >2 change-points still exhibit a steady-state, but it just takes some time to stabilise.

The number of change-points for OR-gate #11 is only one, thus we can conclude that there is a rise in output. At last the stable output concentration level needs to be found. A simple method for doing so is to find the mean for all data-points after the change-point, i.e. $m + 1$ from the previous step, this corresponds to $time > 640$. The mean based on data-points where $time > 640$ can then easily be calculated:

$$\frac{X_{m+1} + X_{m+2} + ... + X_n}{n - m} = \frac{85.75 + 86.71 + ... + 85.74}{99 - 8} \approx 86.24$$

The mean above applies for the row in the truth-table where Ara and aTc are high, to make a complete characterisation all other rows should be simulated and evaluated as well. Two other interesting numbers in this context are $max1$ and $min1$:

$$max1 = \max_{i=m+1,...,n} (X_i) = 88.93$$

$$min1 = \min_{i=m+1,...,n} (X_i) = 83.66$$

Usually an error-margin should be applied to these values to be certain that output concentration levels are interpreted correctly.

8.5 Implementation

A simplified prototype of the GDA tool-chain proposed in Sec. 8.2 has been made using the library based technology mapping approach explained in Sec. 8.3 where the Quine-McKlusky minimisation is performed using a third-party implementation[4].

This tool depends on the DTU-SB Framework, leveraging its I/O capabilities and simulation engine. The independent GUI explained in Ch. 7 has been

[4] https://github.com/qtstc/Circuit-Simulation/tree/master/Quine-McCluskeyJavaSampleCode.

altered to support this extension. A tutorial describing how to use this tool can be found in Appendix D.

The key limitations of this prototype are:

- The input truth-table do not support *don't cares*.
- Only a single minimal form is used even though several may exist for a given truth-table.
- Partitioning is not implemented, thus only expressions with a single output protein are allowed.
- Due to the non-canonical nature of AIGs only SoP expressions with at most two products with at most two literals each are solved deterministically. Larger expressions may be decomposed into different AIG layouts each time, hence it may not necessarily find solutions even though such exists.
- The AIG based approach suffers from structural bias so finding a satisfying composition, if such exists, cannot be guaranteed.
- Currently our prototype does not support automatic evaluation of proposed designs, so the quality of the synthesised devices needs to be manually assessed.

Each of these points are naturally subject to further work. The implementation contains a small library of parts as well.

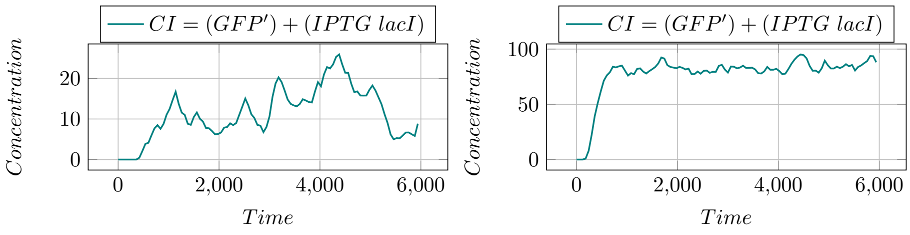
8.6 Discussion

The proposed synthesis heuristic makes no guarantees about optimality or even finding a solution if such exists. Although these genetic circuits are asynchronous in their very nature we have here focused on using technology mapping techniques developed for *synchronous circuits* as it is well-tested, comprehensible and easy to implement. Using theory of synchronous circuits will ultimately – compared to the theory of *asynchronous circuits* – result in additional (invalid) design candidates that need to be assessed by the simulator and therefore increase the overall complexity of the process. Further research could look into how the mapping techniques developed for asynchronous circuits, e.g. Nelson (2004); chun Chou et al. (1999); Siegel et al. (1993), can be applied.

8.6.1 Parts characterisation

It should also be mentioned that this approach naturally will only work for parts reaching a steady-state meaning they can be summarised in a truth-table. Parts with no steady-sate, such as oscillators, cannot be handled properly although future research could look into extending the input format to also support arbitrary mathematical functions such as $GFP = Ara - sin(aTc)$ for synthesis.

Consider the case where a composition of parts, each logical describable, cannot work in conjunction as the output concentration of one part never reaches a level to fully activate another part. Fig 8.15a shows the behaviour of such an example where the library parts #3: $Ara = (lacI\ GFP)$, #4 $aTc = (GFP')$ and #11: $CI = aTc + Ara$, do not work as intended as part #4 do not produce enough aTc to fully activate part #11.



(a) Parts #3, #4 and #11. **(b)** Parts #3, #4 twice and #11.

Figure 8.15: Composite behaviour of different quantities of library parts #3, #4 and #11 during absence of GFP, $lacI$ and $IPTG$.

With knowledge about the required activation- and output-levels it will be possible to extend the technology mapping phase to easily account for this by simply putting duplicate library parts in the design until the required activation level has been reached. Fig. 8.15b shows the composite behaviour using part #4 twice which is enough to drive the output to a steady-state.

Duplicating parts with the theory used in the simulator will just result in amplification of the respective concentrations without introducing side-effects. In practice there might be issues with concentrations much higher than required as it is uncertain if the *Brownian dynamics* is still applicable and whether it will introduce additional cross-talk. Wet-lab experiments should reveal if these issues should be accounted for.

Even though this can dramatically increase the amount of possible design candidates for a given target function, there will still be cases where this theory is inadequate as in the case of the oscillator from Fig. 6.2, where three parts easily describable by logical functions in composition no longer are describable by simple Boolean logic, see its oscillating behaviour in Fig. 8.7. Further, only output proteins in library parts can be amplified, it cannot solve the problem of too low initial concentrations of input protein(s) to activate any of the available library parts.

8.6.2 Technology mapping

The mapping approach suffers from *structural bias* which means that the combination of parts making up a solution is very dependent on the structural representation, and not necessarily the actual semantics, of the AIG. Other issues include the non-canonical nature of the AIG as well as its lack of support for *don't cares*. Some of these issues could be eliminated by storing/considering parts with bridged or short-circuited inputs leading to new AIG representations.

Alternative solutions overcoming many of these shortcomings in EDA, though $\mathbb{NP}$-hard, are surveyed in Benini and De Micheli (1997). These are all based on the canonical *binary decision Diagram* (BDD) representation instead of using AIG but requires more sophisticated algorithms for mapping library parts. It is not immediately clear how to modify these to support the protein-compatibility requirement during matching.

BDDs do not receive as much attention as AIGs as they can make the layout calculation intractable for even the simplest microprocessor design, but we believe they are worth investigating further as the complexity of genetic layouts are orders of magnitudes simpler than that of even legacy microprocessors and they will be able to solve the shortcomings mentioned here, i.e. that the decomposition to AIGs are non-canonical and that the mapping suffers from structural bias and thus possible solutions (to more complex layouts) are not guaranteed to be found.

8.6.3 Automatic evaluation

Concerning automatic evaluation of parts and new designs, there exists several alternative methods to the proposed naïve method. The proposed method is composed of several ideas and is rather expensive as several thousand random samples of the data need to be analysed to be able to establish a confidence level about the given characterisation. In the consideration of alternative methods it should be noted that often a method is well-suited for special cases, e.g. some methods work very well on simulations where a steady-state can be found, Goldsman (2010). When evaluating genetic designs we cannot be certain that the simulation has a steady-state.

The most important – and probably the hardest – task of automatic evaluation is deciding whether a simulation has a steady-state or not. Development of a more sophisticated approach should involve *sequential change-point detection algorithms* to identify the number of change-points and on the basis of that decide whether a steady-state is present. If it turns out that the simulation most likely is steady *steady-state analysis algorithms* such as the *batch means method* should be used to find the output concentration level. If the automatic evaluation gives a negative answer, we cannot be completely certain that the answer is correct, as the simulation can turn out differently when the simulation time is increased. Evaluation of designs is an important, time-consuming step in a GDA-tool so developing sophisticated automatic evaluation methods is of great importance.

Chapter 9

Conclusion

In this thesis we establish a biological foundation to give an insight into the world of synthetic biology and better understand some of the challenges that are inherit in this area. The class of stochastic petri nets (SPNs) is formally introduced to ensure the reader prerequisites the necessary knowledge for modelling the biological pathways as networks.

SPNs are analysed using *stochastic simulation algorithms* (SSAs). Gillespie's SSA that generalises the *chemical master equation* (CME) and works for systems describable by *Brownian dynamics* has been found useful in this context, as interactions of proteins in biological systems are often described by Brownian dynamics. Several, mainly performance, improvements to this SSA are surveyed in Sec. 5.4.

Usually when developing models, regardless of the scientific discipline, the models are compared to some *real-world* measurements to asses their quality. As we did not have access to perform *wet-lab* experiments ourselves we have spend weeks searching for comparisons of such real-world behaviour and pre-modelled behaviour without any luck. Finally we contacted Chris J. Myers author of Myers (2011) whom told us that the modelling of synthetic biology is still in such a early stage that this is even far from possible, see Appendix B. At the current stage the models are created from experimental data, not vice versa.

This indicates that many great difficulties and challenges lie ahead of establishing more accurate models. The choice of the very generic, extensible and intuitively easy-understandable SPN representation for our models has ensured that future detailed findings for specifying models can easily be supported either by embedding new transition formulas or altering the SPN networks, and thereby laying the foundation for a long-living framework. Using these SPNs we have surveyed several approaches to model the biological pathways in Ch. 6, in the remainder of the thesis we assumed that the modelling technique presented is usable for modelling genetic devices.

On the basis of the research summarised above, we have made a sound and modular framework, *DTU-SB Framework*, that implements many of the ideas and theories presented in this thesis including a prototype of the proposed toolchain from Fig. 8.1 for genetic logic synthesis. The framework accepts the widely used SBML format as input and can thereby easily be compared to similar tools and frameworks. We verified that the implementation of Gillespie's SSA corresponded to Snoopy's implementation by comparing simulation outputs. All simulation graphs shown in this thesis are created with the DTU-SB Framework.

Referring to the discussion on modelling, evaluations of predictive-simulations and real-world experiments still show too low cohesion to be of use. Although the simulated behaviour and compositions obtained using the DTU-SB Framework will not necessarily work in practice, the framework can still be used to give guidance and identify some compositions that will clearly not work and thereby reduce the amount of required wet-lab experiments to be performed.

9.1 Logic synthesis

Assuming a correct model we have proposed how to automate the synthesis of new biological systems by leveraging behavioural knowledge of existing parts obtained by evaluating simulations. This behaviour is specified using Boolean logic, we are thereby able to draw parallels to *electronic design automation* (EDA) and thus in theory support synthesis of arbitrary complex systems. Naturally this approach have its limitations as not all parts can be sufficiently described by just Boolean logic. We call this *genetic design automation* (GDA), which involves minimisation of Boolean expressions, finding designs with the desired behaviour and simulation, characterisation and evaluation of these designs.

Technology mapping algorithms used in EDA has been deemed unsatisfactory as they do not account for challenges inherit in connecting different parts, thus we suggested a new technology mapping method taking these challenges into account. When researching for the construction of this new method we saw that the *and-inverter graph* (AIG)-data structure received a lot of positive attention and that nearly all commercial EDA tools made use of this representation. Based on this the decision was made to employ this data-structure as it required little effort, using this representation, to alter the existing technology mapping algorithms developed for EDA to enable GDA.

The AIG structure have some drawbacks resulting in not necessarily all possible solutions to a target function can be found. The canonical AIG- *With choice* and *binary decision diagram* (BDD)-data structure should be further researched to see if their associated technology mapping algorithms can be altered to support GDA. If this is the case, all the issues of our proposed approach can be eliminated and hence guarantee finding solutions if such exists.

The technology mapper needs well-defined library parts, which is why we have outlined the important characteristics that need to be identified for these parts, and how these characteristics can been identified using automatic evaluation. Evaluation also plays an important role in deciding whether proposed designs adhere to desired Boolean behaviour specified. The technology mapper can be further improved by incorporating even more different characteristics, e.g. activation levels of parts can help the technology mapper decide if a certain protein concentration level should be amplified by using the same library part several times.

9.2 Related work

Numerous tools and modelling frameworks for use in the field of synthetic biology already exists where some of the most noticeable are the following:

iBioSim Myers et al. (2009) A framework for analysis and design of genetic circuits developed by *Myers Research Group* from University of Utah. All research carried out in this group have been implemented in this framework and have by now a comprehensive list of advanced features: GDA, alternative simulation strategies, parameter estimation, genetic classification and automated evaluation of genetic circuits.

Snoopy Heiner et al. (2012a) A lightweight petri net editor with a built-in stochastic simulator. This tool supports a myriad of petri net variants that can be exported using the SBML standard.

GEC A programming language developed at *Microsoft Research* including a prototype compiler- and simulator-suite. The programming language operates on parts level, see Sec. 3.3, and supports specification of protein variables that can be resolved in a compilation step using a database of parts.

Eugene Bilitchenko et al. (2011) A CAD tool using a mixture between imperative- and constraint-programming that operates on both devices and individual parts, see Sec. 3.3. Eugene is greatly inspired by EDA tools and uses design space exploration to come up with candidate compositions.

COPASI A simulator for general biological devices with built-in support for parameter estimation and optimisation as well as many visualisation options.

Tinkercell Chandran et al. (2009) A comprehensive CAD tool with a very user-friendly design and an exposed API for integration.

The implementation of the theory in chapters 2-7, corresponding to the first milestone, is very similar to the functionality found in any of the above tools. DTU-SB differs from these tools by the way we perform genetic logic synthesis, explained in Ch. 8.

9.3 Reflections

In this cross-disciplinary approach to synthetic biology we have encountered some noticeable differences in how to approach scientific problems: the world of biology is much more driven by empirical experiments (top-down) whereas the area of computer science engineering typically aim for being able to explain every phenomenon (bottom-up). This difference has been very apparent in the biological literature we have read and have at times caused minor frustrations over having to accept some behaviour without further explanation.

The development of this framework and thesis have been driven by the desire to have it being of practical use and educational value to other computer scientists who currently do not have the time or courage to review biological literature in detail, but that wish to emerge into the field of synthetic biology.

This thesis takes a classical approach to managing the complexity of the models by assuming the existing circuits in the cell are always orthogonal to the model we wish to implement and thereby never exhibit external interference. In practice this can be obtained by creating a *minimal cell*; a cell acting as a container stripped of all non-essential behaviour. Noticeable alternatives to managing the complexity includes:

- Using a Lab-On-Chip where the genetic circuit is split into multiple cells placed in different micro-fluid chambers which are then routed to give the desired behaviour while introducing a minimum of interference, Huang et al. (2013).
- By rewiring, so instead of designing these genetic circuits from scratch, simply rewire the existing circuits of the cells. Naturally selecting cells with behaviour close to the target function requires less effort and risk, Nandagopal and Elowitz (2011).

We believe that computer scientists and software engineers in the future will be able to contribute greatly to research of areas within synthetic biology, especially by applying the typical bottom-up approach where models are created and simulated and by that predictions can be made with some certainty before realising designs in wet-lab experiments.

9.4 Future directions

Each chapter discusses areas that need further work, here are the most important parts summarised:

- Improve the precision of the technology mapping by investigating the applicability of the BDD-data and AIG-with choice structures with their related mapping techniques to overcome the issues mentioned in Sec. 8.5.
- Research applicability of EDA techniques developed for *asynchronous circuits* to GDA.
- Implement automatic evaluation from Sec. 8.4 to automatically asses the quality of the simulations of the synthesised circuits using their desired behaviour as target and thereby making the framework fully automated.
- Increase the simulation performance by e.g. GPGPU based simulation algorithms as in Komarov and D'Souza (2012), employing τ-leaping and the use of hybrid petri net that dynamically switches between using a stochastic solver for small concentrations and a continuous solver for high concentrations as proposed in e.g. Herajy and Heiner (2012).

- In the context of the framework being of educational value a future extension could involve a SPN editor for direct interaction with and understanding of the models. Currently we create models in Snoopy and export them to SBML.
- Implement various SPN-based model checkers to analyse the models and give guarantees about properties such as *liveness* and timings.
- Further investigate whether employing translational regulation with sRNA can construct even more complex devices, Marchisio and Stelling (2011), possibly overcoming the *size* requirement in Sec. 8.2. The SPN models can easily be extended to account for this.

Appendix A

Modelling examples details

The rate function Eq. (6.7) on page 60 for the oscillator is the Hill equation describing binding. The following variables are used in the equation:

$a0_{tr}$	is the transcription rate from a fully repressed promotor.
a_{tr}	is the transcription rate from a free promotor.
KM	is the number of repressor molecules per cell giving half maximal repression. *It represents the concentration of an inhibitor substance that is required to suppress 50% of an effect.*[1]
n	is the Hill coefficient describing cooperativity.
cI	is the protein repressing this transcription.

All of these definitions can be found on the BioModels page for this device: `http://www.ebi.ac.uk/biomodels-main/BIOMD0000000012`.

[1]From `http://www.ebi.ac.uk/sbo/main/SBO:0000288`.

Appendix B

Email from Chris J. Myers

Hi,

Thanks for your interest in my textbook. You are correct that it is hard to find many good studies that compare models to experiments. Unfortunately, there are very few biological systems that we understand well enough to construct complete and accurate models. The phage lambda is one of these examples where there is sufficient knowledge to build and test such models, as was done by Arkin and repeated by us using improved abstractions. There are some recent promising developments, see for example the Karr et al whole cell model from Stanford that was recently published in Cell. Most studies like that of lambda and the whole cell model construct models that yield results that compare with existing experimental knowledge. While this can be valuable as it gives you some insight into the mechanisms that may be producing the observed behavior, there are few studies which actually have shown how a model can make a prediction that is "later" validated in the laboratory. I think the main challenge to this effort continues to be the difficulty in gaining sufficient understanding of the system to construct a complete model that includes all important interactions, not to mention getting rate parameters which is very challenging.

I believe the reason for this is that evolution produces extremely complex systems. When I presented my phage lambda results, the speaker before me, Drew Endy, commented that he had become frustrated that his models produced results that did not agree with his models. He decided that perhaps these systems were not designed very well, and he could re-engineer them to agree with his models. This comment he used to motivate his work in synthetic biology. While certainly meant as a bit of a joke, there is some truth in this. Developing models of things we have not designed is very difficult, but models of things we have designed is much easier. For this and other reasons, I've focused my recent modeling work to help support synthetic biology where we have a better chance of building predictive models. However, I still think Systems Biology is very important and will continue to improve its modeling efforts as our experimental methods improve.

I hope this helps. Please feel free to contact me with further questions as your work progresses.

Best of luck.

Chris

APPENDIX C

Evaluation data for OR-gate 11

i	Time	CI	CI ($n=4$)	CI ($n=10$)	CUSUM
1	80	8.383409	8.4	8.4	-75.47138162
2	160	31.117693	31	31	-128.3427632
3	240	48.948135	37.958391	49	-163.2141449
4	320	63.384327	54.25020125	63	-184.0855265
5	400	73.55065	67.6166215	66.2071648	-201.7497433
6	480	84.583374	77.32013525	74.3706275	-211.2504974
7	560	87.76219	83.8082235	80.0486612	-215.0732178
8	640	89.33668	87.256186	83.7065167	-215.2380828
9	720	87.3425	88.026015	85.7470574	-213.362407
10	800	87.66269	88.5899765	86.7122234	-210.5215652
11	880	90.018036	88.230314	86.435519	-207.9574278
12	960	87.89803	87.7763615	86.0138456	-205.8149638
13	1040	85.52669	86.8081225	85.6122086	-204.0741369
14	1120	83.789734	85.104191	85.3551796	-202.5903389
15	1200	83.20231	83.583766	84.7593836	-201.7023369
16	1280	81.81633	83.0884575	84.062962	-201.5107565
17	1360	83.545456	83.4711015	84.0326355	-201.3495026
18	1440	85.32031	83.8635765	84.547775	-200.6731093
19	1520	84.77221	83.8356765	84.9989936	-199.5454973

20	1600	81.70473	83.7127675	85.5735596	-197.8433193
21	1680	83.05382	84.28138125	86.1217041	-195.5929968
22	1760	87.594765	85.75785	86.6418579	-192.8225205
23	1840	90.678085	87.4071475	87.1492129	-189.5446893
24	1920	88.30192	88.880685	87.6562579	-185.759813
25	2000	88.94797	88.8064375	88.6293373	-181.0018573
26	2080	87.297775	88.32366475	88.9315853	-175.9416536
27	2160	88.746994	88.84664975	88.4672978	-171.3457374
28	2240	90.39386	89.07032225	87.3004063	-167.9167128
29	2320	89.84266	90.1047595	86.1139643	-165.6741301
30	2400	91.435524	89.437086	85.1366753	-164.4088364
31	2480	86.0763	87.5765935	85.0441163	-163.2361017
32	2560	82.95189	84.868221	84.9811044	-162.1263789
33	2640	79.00917	81.118715	84.7952814	-161.2024792
34	2720	76.4375	79.39341	84.2381134	-160.8357474
35	2800	79.17508	80.24848375	83.715094	-160.992035
36	2880	86.372185	82.52541	83.663264	-161.2001526
37	2960	88.116875	85.5499425	84.158757	-160.9127772
38	3040	88.53563	86.8239175	85.415652	-159.3685069
39	3120	84.27098	86.78220375	86.778247	-156.4616415
40	3200	86.20533	86.142485	87.666916	-152.6661071
41	3280	85.558	85.9852825	87.5575015	-148.9799872
42	3360	87.90682	87.8120675	87.120692	-145.7306768
43	3440	91.57812	88.7765975	87.508714	-142.0933445
44	3520	90.06345	89.40254	88.195552	-137.7691741
45	3600	88.06177	88.745345	87.988576	-133.6519797
46	3680	85.27804	86.78801	87.8377565	-129.6856048
47	3760	83.74878	87.37611	87.66898	-125.8880064
48	3840	92.41585	88.1455075	86.747691	-123.0116971
49	3920	91.13936	87.85989	85.791472	-121.0916067
50	4000	84.13557	87.93514625	85.1810584	-119.7819299
51	4080	84.049805	86.3859475	85.2634704	-118.3898411
52	4160	86.219055	84.192415	86.1129894	-116.1482333
53	4240	82.36523	83.2838375	86.2547059	-113.7649091
54	4320	80.50126	82.76079475	86.0837519	-111.5525388
55	4400	81.957634	82.731571	86.4068839	-109.0170365
56	4480	86.10216	85.201256	86.7433174	-106.1451007
57	4560	92.24397	88.53419475	86.4562289	-103.5602534
58	4640	93.833015	90.40224125	86.9852979	-100.4463372
59	4720	89.42982	90.71842375	87.6728153	-96.64490348
60	4800	87.36689	89.51096625	88.1922399	-92.3240452
61	4880	87.41414	86.889755	88.1411334	-88.05429342
62	4960	83.34817	86.44628	87.6485984	-84.27707664
63	5040	87.65592	86.448666	86.7882529	-81.36020536

64	5120	87.376434	86.383101	86.3635303	-78.86805668
65	5200	87.15188	86.94383225	85.9174869	-76.8219514
66	5280	85.591095	86.85950725	85.3827395	-75.31059352
67	5360	87.31862	86.32278875	85.561956	-73.62001914
68	5440	85.22956	85.83046725	85.7465524	-71.74484836
69	5520	85.182594	85.1593075	86.0530916	-69.56313838
70	5600	82.906456	83.846319	86.2277546	-67.2067654
71	5680	82.066666	83.82401275	86.9186451	-64.15950192
72	5760	85.140335	84.90383525	87.0409591	-60.98992444
73	5840	89.501884	86.78767775	87.1448105	-57.71649556
74	5920	90.441826	88.49563875	86.9880251	-54.59985208
75	6000	88.89851	90.335555	86.9911175	-51.4801162
76	6080	92.5	90.095524	87.1813589	-48.17013892
77	6160	88.54176	89.052086	87.2949374	-44.74658314
78	6240	86.268074	87.7311435	87.13376	-41.48420476
79	6320	83.61474	85.3404885	86.8943254	-38.46126098
80	6400	82.93738	84.1973185	86.8716039	-35.4610387
81	6480	83.96908	84.19933	86.2372373	-33.09518302
82	6560	86.27612	85.2681725	85.7603953	-31.20616934
83	6640	87.89011	86.5456975	85.5732309	-29.50432006
84	6720	88.04748	87.72125125	85.8058519	-27.56984978
85	6800	88.671295	87.69130475	86.1321419	-25.3090895
86	6880	86.156334	86.66211225	86.2411659	-22.93930522
87	6960	83.77334	85.74934975	86.4329349	-20.37775194
88	7040	84.39643	85.0667635	86.2872909	-17.96184266
89	7120	85.94095	85.07775	85.7617319	-16.07149238
90	7200	86.20028	85.399245	85.2965578	-14.6463162
91	7280	85.05932	86.34859	85.0010824	-13.51661542
92	7360	88.19381	86.47177	85.2148824	-12.17311464
93	7440	86.43367	85.6196725	85.6710654	-10.37343086
94	7520	82.79189	85.359731	85.7385824	-8.50623008
95	7600	84.019554	84.1116735	85.68728267	-6.690329033
96	7680	83.20158	83.981091	85.765778	-4.795932653
97	7760	85.91134	85.5226835	85.41891629	-3.248397988
98	7840	88.95826	86.171825	85.24979067	-1.869988941
99	7920	86.61612	87.16190667	85.7413708	2.39048E-07

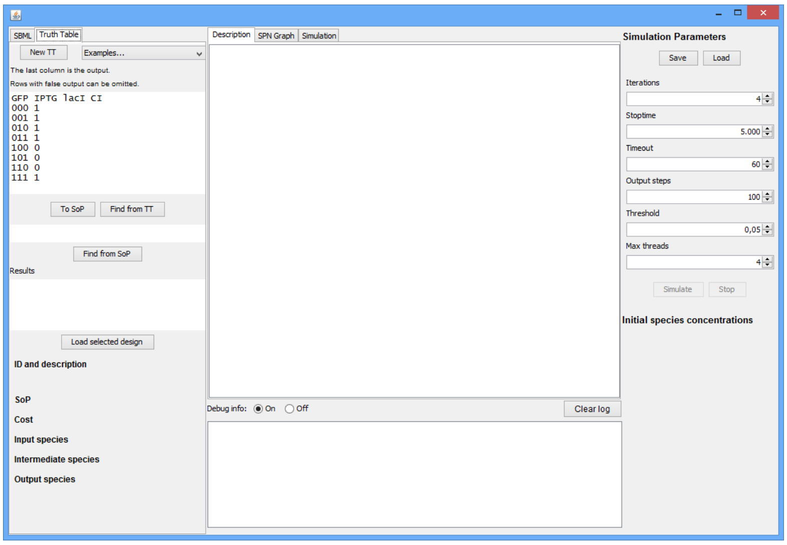
APPENDIX D

DTU-SB GDA Tutorial

This tutorial will explain how to perform GDA synthesis with the DTU-SB framework.

Fig. 1: Open the `Truth Table` tab on the left where a default example truth-table is shown. We will use that example as well as the default library (that can be viewed on the `SBML` tab).

Fig. 2: Clicking the `To SoP` button shows the minimised SoP expression: $CI = (GFP') + (IPTG\ lacI)$. Clicking `Find from SoP` yields four different design candidates for this SoP ordered by their cost. Details about e.g. the library parts used to realise the selected design can be seen on the bottom left. Here we select *Design 1* composed of *Part 8* and *Part 3* and click `Load selected design`.



The last column is the output.
Rows with false output can be omitted.

```
GFP IPTG lacI CI
000 1
001 1
010 1
011 1
100 0
101 0
110 0
111 1
```

Results

ID and description

SoP

Cost

Input species

Intermediate species

Output species

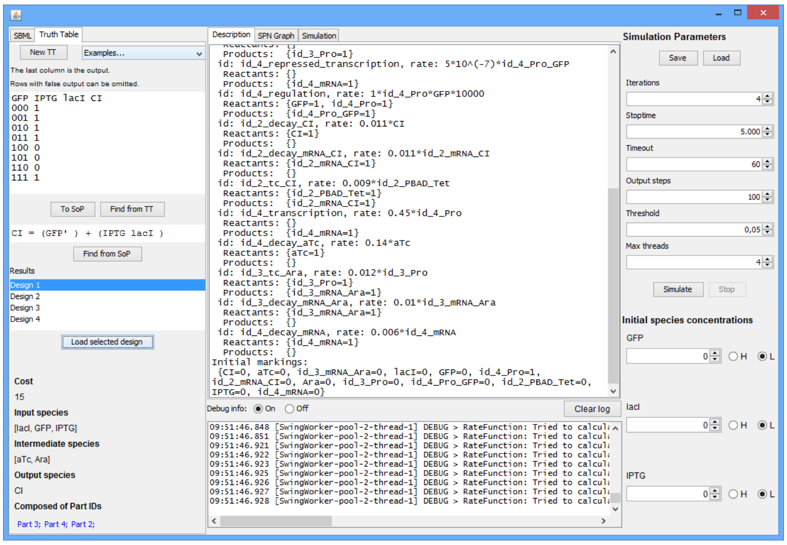
Debug info: [✓ On] Off

Simulation Parameters

Iterations: 4
Stoptime: 5.000
Timeout: 60
Output steps: 100
Threshold: 0,05
Max threads: 4

Initial species concentrations

Figure 1



The last column is the output.
Rows with false output can be omitted.

```
GFP IPTG lacI CI
000 1
001 1
010 1
011 1
100 0
101 0
110 0
111 1
```

```
CI = (GFP' ) + (IPTG lacI )
```

Results

Design 1
Design 2
Design 3
Design 4

Cost
15

Input species
[lacI, GFP, IPTG]

Intermediate species
[aTc, Ara]

Output species
CI

Composed of Part IDs
Part 3; Part 4; Part 2;

```
 Products:  {id_3_Pro=1}
id: id_4_repressed_transcription, rate: 5*10^(-7)*id_4_Pro_GFP
 Reactants: {}
 Products:  {id_4_mRNA=1}
id: id_4_regulation, rate: 1*id_4_Pro*GFP*10000
 Reactants: {GFP=1, id_4_Pro=1}
 Products:  {id_4_Pro_GFP=1}
id: id_2_decay_CI, rate: 0.011*CI
 Reactants: {CI=1}
 Products:  {}
id: id_2_decay_mRNA_CI, rate: 0.011*id_2_mRNA_CI
 Reactants: {id_2_mRNA_CI=1}
 Products:  {}
id: id_2_tc_CI, rate: 0.009*id_2_PBAD_Tet
 Reactants: {id_2_PBAD_Tet=1}
 Products:  {id_2_mRNA_CI=1}
id: id_4_transcription, rate: 0.45*id_4_Pro
 Reactants: {}
 Products:  {id_4_mRNA=1}
id: id_4_decay_aTc, rate: 0.14*aTc
 Reactants: {aTc=1}
 Products:  {}
id: id_3_tc_Ara, rate: 0.012*id_3_Pro
 Reactants: {id_3_Pro=1}
 Products:  {id_3_mRNA_Ara=1}
id: id_3_decay_mRNA_Ara, rate: 0.01*id_3_mRNA_Ara
 Reactants: {id_3_mRNA_Ara=1}
 Products:  {}
id: id_4_decay_mRNA, rate: 0.006*id_4_mRNA
 Reactants: {id_4_mRNA=1}
 Products:  {}
Initial markings:
 {CI=0, aTc=0, id_3_mRNA_Ara=0, lacI=0, GFP=0, id_4_Pro=1,
id_2_mRNA_CI=0, Ara=0, id_3_Pro=0, id_4_Pro_GFP=0, id_2_PBAD_Tet=0,
IPTG=0, id_4_mRNA=0}
```

Debug info: [✓ On] Off

```
09:51:46.848 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
09:51:46.851 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
09:51:46.921 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
09:51:46.922 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
09:51:46.923 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
09:51:46.925 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
09:51:46.926 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
09:51:46.927 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
09:51:46.928 [SwingWorker-pool-2-thread-1] DEBUG > RateFunction: Tried to calcula
```

Simulation Parameters

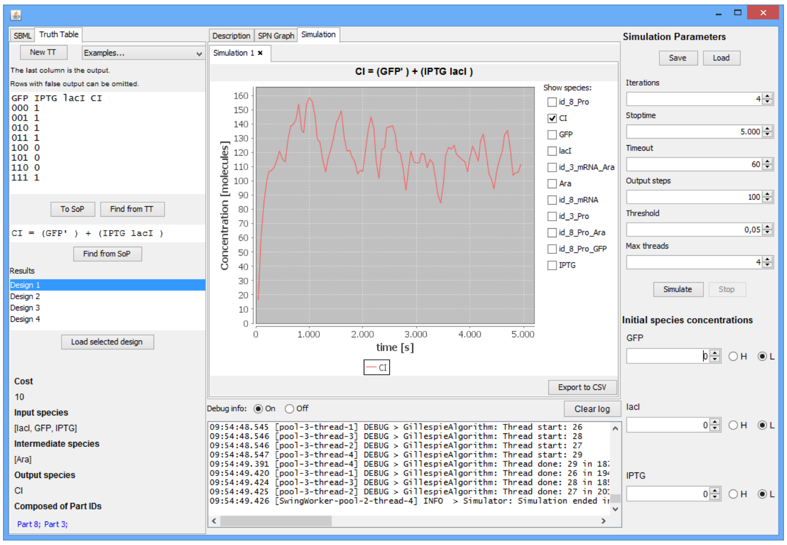
Iterations: 4
Stoptime: 5.000
Timeout: 60
Output steps: 100
Threshold: 0,05
Max threads: 4

Initial species concentrations

GFP: 0 H [✓ L]
lacI: 0 H [✓ L]
IPTG: 0 H [✓ L]

Figure 2

Pressing `Simulate` using the default species concentrations (absence of all species) defined on the lower right, yields a graph similar to that of Fig. 3, which clearly indicates a high level of CI as expected due to the (GFP') implicant.



The last column is the output.
Rows with false output can be omitted.

```
GFP IPTG lacI CI
000 1
001 1
010 1
011 1
100 0
101 0
110 0
111 1
```

```
CI = (GFP' ) + (IPTG lacI )
```

Results
Design 1
Design 2
Design 3
Design 4

Cost
10
Input species
[lacI, GFP, IPTG]
Intermediate species
[Ara]
Output species
CI
Composed of Part IDs
Part 8; Part 3;

CI = (GFP') + (IPTG lacI)

Show species: [✓ CI]

Debug info: [✓ On] Off

```
09:54:48.545 [pool-3-thread-1] DEBUG > GillespieAlgorithm: Thread start: 26
09:54:48.546 [pool-3-thread-3] DEBUG > GillespieAlgorithm: Thread start: 28
09:54:48.546 [pool-3-thread-2] DEBUG > GillespieAlgorithm: Thread start: 27
09:54:48.547 [pool-3-thread-4] DEBUG > GillespieAlgorithm: Thread start: 29
09:54:49.391 [pool-3-thread-4] DEBUG > GillespieAlgorithm: Thread done: 29 in 18
09:54:49.420 [pool-3-thread-1] DEBUG > GillespieAlgorithm: Thread done: 26 in 19
09:54:49.424 [pool-3-thread-3] DEBUG > GillespieAlgorithm: Thread done: 28 in 18
09:54:49.425 [pool-3-thread-2] DEBUG > GillespieAlgorithm: Thread done: 27 in 20
09:54:49.426 [SwingWorker-pool-2-thread-4] INFO  > Simulator: Simulation ended i
```

Simulation Parameters

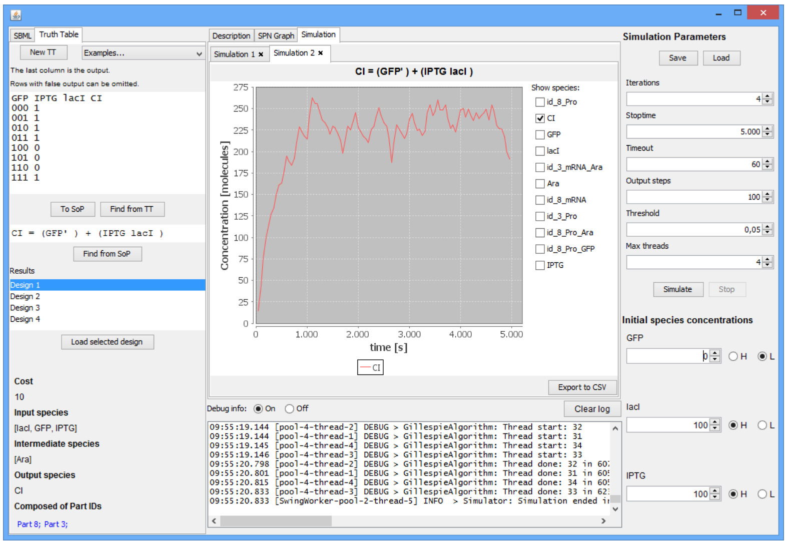
Iterations	4
Stoptime	5.000
Timeout	60
Output steps	100
Threshold	0,05
Max threads	4

Initial species concentrations

GFP	0	H [✓ L]
lacI	0	H [✓ L]
IPTG	0	H [✓ L]

Figure 3

Selecting absence of GFP but presence of both $lacI$ and $IPTG$ yields a behaviour similar to that of Fig. 4. Here we see that the CI level is higher than that of Fig. 3, this is due to both implicants (GFP') and $(IPTG\ lacI)$ evaluate to true where the given design just emits twice the CI.



The last column is the output.
Rows with false output can be omitted.

```
GFP IPTG lacI CI
000 1
001 1
010 1
011 1
100 0
101 0
110 0
111 1
```

```
CI = (GFP' ) + (IPTG lacI )
```

Results
Design 1
Design 2
Design 3
Design 4

Cost
10
Input species
[lacI, GFP, IPTG]
Intermediate species
[Ara]
Output species
CI
Composed of Part IDs
Part 8; Part 3;

CI = (GFP') + (IPTG lacI)

Show species: [✓ CI]

Debug info: [✓ On] Off

```
09:55:19.144 [pool-4-thread-2] DEBUG > GillespieAlgorithm: Thread start: 32
09:55:19.144 [pool-4-thread-1] DEBUG > GillespieAlgorithm: Thread start: 31
09:55:19.145 [pool-4-thread-4] DEBUG > GillespieAlgorithm: Thread start: 34
09:55:19.146 [pool-4-thread-3] DEBUG > GillespieAlgorithm: Thread start: 33
09:55:20.798 [pool-4-thread-2] DEBUG > GillespieAlgorithm: Thread done: 32 in 60
09:55:20.801 [pool-4-thread-1] DEBUG > GillespieAlgorithm: Thread done: 31 in 60
09:55:20.815 [pool-4-thread-4] DEBUG > GillespieAlgorithm: Thread done: 34 in 60
09:55:20.833 [pool-4-thread-3] DEBUG > GillespieAlgorithm: Thread done: 33 in 62
09:55:20.833 [SwingWorker-pool-2-thread-5] INFO  > Simulator: Simulation ended i
```

Simulation Parameters

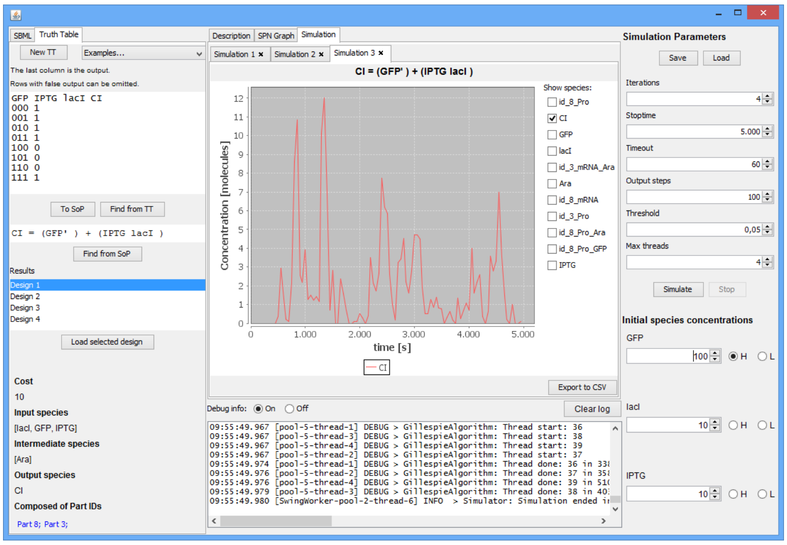
Iterations	4
Stoptime	5.000
Timeout	60
Output steps	100
Threshold	0,05
Max threads	4

Initial species concentrations

GFP	0	H [✓ L]
lacI	100	[✓ H] L
IPTG	100	[✓ H] L

Figure 4

Finally selecting presence of *GFP* but very low concentrations (10) of both *lacI* and *IPTG* yields something similar to Fig. 5, which can be interpreted as absence of *CI* due to the very low concentrations.



The last column is the output.
Rows with false output can be omitted.

```
GFP IPTG lacI CI
000 1
001 1
010 1
011 1
100 0
101 0
110 0
111 1
```

```
CI = (GFP' ) + (IPTG lacI )
```

Results
Design 1
Design 2
Design 3
Design 4

Cost
10
Input species
[lacI, GFP, IPTG]
Intermediate species
[Ara]
Output species
CI
Composed of Part IDs
Part 8; Part 3;

CI = (GFP') + (IPTG lacI)

Show species: id_8_Pro, [✓ CI], GFP, lacI, id_3_mRNA_Ara, Ara, id_8_mRNA, id_3_Pro, id_8_Pro_Ara, id_8_Pro_GFP, IPTG

Debug info: [✓ On] Off

```
09:55:49.967 [pool-5-thread-1] DEBUG > GillespieAlgorithm: Thread start: 36
09:55:49.967 [pool-5-thread-3] DEBUG > GillespieAlgorithm: Thread start: 38
09:55:49.967 [pool-5-thread-4] DEBUG > GillespieAlgorithm: Thread start: 39
09:55:49.967 [pool-5-thread-2] DEBUG > GillespieAlgorithm: Thread start: 37
09:55:49.974 [pool-5-thread-1] DEBUG > GillespieAlgorithm: Thread done: 36 in 33
09:55:49.976 [pool-5-thread-2] DEBUG > GillespieAlgorithm: Thread done: 37 in 35
09:55:49.976 [pool-5-thread-4] DEBUG > GillespieAlgorithm: Thread done: 39 in 51
09:55:49.979 [pool-5-thread-3] DEBUG > GillespieAlgorithm: Thread done: 38 in 40
09:55:49.980 [SwingWorker-pool-2-thread-6] INFO  > Simulator: Simulation ended i
```

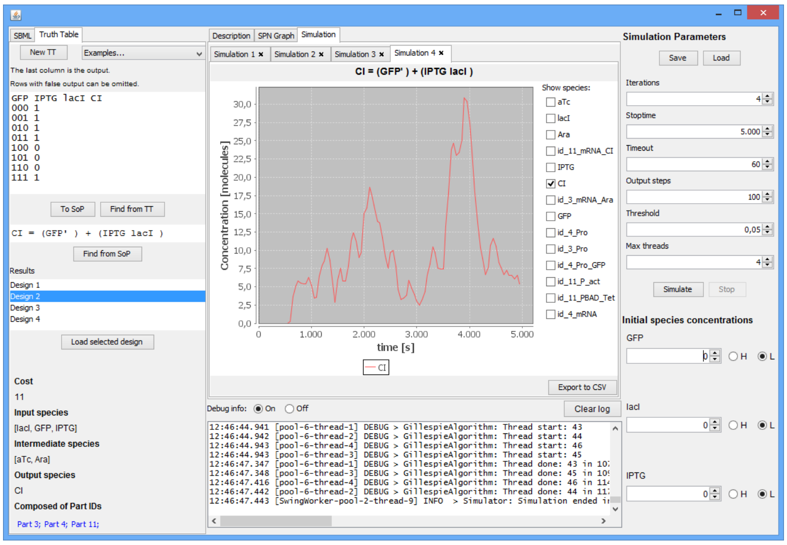
Simulation Parameters

Iterations: 4
Stoptime: 5.000
Timeout: 60
Output steps: 100
Threshold: 0,05
Max threads: 4

Initial species concentrations

GFP: 100 [✓ H] L
lacI: 10 H L
IPTG: 10 H L

Figure 5



The last column is the output.
Rows with false output can be omitted.

```
GFP IPTG lacI CI
000 1
001 1
010 1
011 1
100 0
101 0
110 0
111 1
```

```
CI = (GFP' ) + (IPTG lacI )
```

Results
Design 1
Design 2
Design 3
Design 4

Cost
11
Input species
[lacI, GFP, IPTG]
Intermediate species
[aTc, Ara]
Output species
CI
Composed of Part IDs
Part 3; Part 4; Part 11;

CI = (GFP') + (IPTG lacI)

Show species: aTc, lacI, Ara, id_11_mRNA_CI, IPTG, [✓ CI], id_3_mRNA_Ara, GFP, id_4_Pro, id_3_Pro, id_4_Pro_GFP, id_11_P_act, id_11_PBAD_Tet, id_4_mRNA

Debug info: [✓ On] Off

```
12:46:44.941 [pool-6-thread-1] DEBUG > GillespieAlgorithm: Thread start: 43
12:46:44.942 [pool-6-thread-2] DEBUG > GillespieAlgorithm: Thread start: 44
12:46:44.943 [pool-6-thread-4] DEBUG > GillespieAlgorithm: Thread start: 46
12:46:44.943 [pool-6-thread-3] DEBUG > GillespieAlgorithm: Thread start: 45
12:46:47.347 [pool-6-thread-1] DEBUG > GillespieAlgorithm: Thread done: 43 in 10
12:46:47.348 [pool-6-thread-3] DEBUG > GillespieAlgorithm: Thread done: 45 in 10
12:46:47.416 [pool-6-thread-4] DEBUG > GillespieAlgorithm: Thread done: 46 in 11
12:46:47.442 [pool-6-thread-2] DEBUG > GillespieAlgorithm: Thread done: 44 in 11
12:46:47.443 [SwingWorker-pool-2-thread-9] INFO  > Simulator: Simulation ended i
```

Simulation Parameters

Iterations: 4
Stoptime: 5.000
Timeout: 60
Output steps: 100
Threshold: 0,05
Max threads: 4

Initial species concentrations

GFP: 0 H [✓ L]
lacI: 0 H [✓ L]
IPTG: 0 H [✓ L]

Figure 6

These three cases behaved as expected but if we instead selected *Design 2* and absence of all initial concentrations we get the low and fluctuating *CI* behaviour in Fig. 6, which is somewhat unexpected and do not adhere to the target function. This can be explained by compatibility problems between *Part 11* and *Part 4*, although they both work fine in isolation. *Part 11* is never fully triggered due to the long settling time of *Part 4* which is something that logical analysis cannot immediately reveal. This stresses the importance of simulating every combination of inputs in order to asses the quality of a given design.

The oscillator in Fig. 8.7 on page 93 is another example where three inverter parts, each describable by truth-tables, give rise to oscillating behaviour that cannot be described by a logic function.

Bibliography

ABC. ABC: A system for sequential synthesis and verification, release 70930. http://www.eecs.berkeley.edu/~alanmi/abc/. URL http://www.eecs.berkeley.edu/~alanmi/abc.

James J. Collins Ahmad S. Khalil. Synthetic biology: applications come of age. *Nature Reviews Genetics*, (5):367–379, 2010. doi: 10.1038/nrg2775. URL http://www.nature.com/nrg/journal/v11/n5/full/nrg2775.html.

Aurélien Alfonsi, Eric Cancès, Gabriel Turinici, Barbara Di Ventura, and Wilhelm Huisinga. Exact simulation of hybrid stochastic and deterministic models for biochemical systems. Rapport de recherche RR-5435, INRIA, 2004. URL http://hal.inria.fr/inria-00070572.

Charles J. Alpert and Andrew B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: The VLSI Journal*, 19:1–81, 1995.

Altera. Quartus ii. http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html.

J. Alves. Genetic code bw. http://openclipart.org/detail/95197/genetic-code-bw-by-j_alves, November 2010.

A. Arkin, J. Ross, and HH McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected escherichia coli cells. *GENETICS*, 149(4):1633–1648, 1998. ISSN 00166731, 19432631.

G. Baldwin, P.S. Freemont, and R.I. Kitney. *Synthetic Biology: A Primer*. Imperial College Press, 2012. ISBN 9781848168633. URL http://books.google.dk/books?id=Vo0CywAACAAJ.

Jacob Beal, Andrew Phillips, Douglas Densmore, and Yizhi Cai. High-level programming languages for biomolecular systems. In Heinz Koeppl, Gianluca Setti, Mario di Bernardo, and Douglas Densmore, editors, *Design and Analysis of Biomolecular Circuits*, pages 225–252. Springer New York, 2011. ISBN 978-1-4419-6765-7. doi: 10.1007/978-1-4419-6766-4_11. URL http://dx.doi.org/10.1007/978-1-4419-6766-4_11.

Jacob Beal, Ron Weiss, Douglas Densmore, Aaron Adler, Evan Appleton, Jonathan Babb, Swapnil Bhatia, Noah Davidsohn, Traci Haddock, Joseph Loyall, Richard Schantz, Viktor Vasilev, and Fusun Yaman. An end-to-end workflow for engineering of biological networks from high-level specifications. *ACS Synthetic Biology*, 1(8):317–331, 2012. doi: 10.1021/sb300030d. URL http://pubs.acs.org/doi/abs/10.1021/sb300030d.

Luca Benini and Giovanni De Micheli. A survey of boolean matching techniques for library binding. *ACM Trans. Des. Autom. Electron. Syst.*, 2(3):193–226, July 1997. ISSN 1084-4309. doi: 10.1145/264995.264996. URL http://doi.acm.org/10.1145/264995.264996.

Swapnil Bhatia and Douglas Densmore. Pigeon: A design visualizer for synthetic biology. *ACS Synthetic Biology*, 2(6):348–350, 2013.

Lesia Bilitchenko, Adam Liu, Sherine Cheung, Emma Weeding, Bing Xia, Mariana Leguia, J. Christopher Anderson, and Douglas Densmore. Eugene - a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS ONE*, 6(4):e18882, 04 2011. doi: 10.1371/journal.pone.0018882. URL http://dx.doi.org/10.1371%2Fjournal.pone.0018882.

MA Blätke, M Heiner, and W Marwan. Tutorial - Petri Nets in Systems Biology. Technical report, Otto von Guericke University Magdeburg, Magdeburg Centre for Systems Biology, August 2011.

F. R. Blattner, G. Plunkett, C. A. Bloch, N. T. Perna, V. Burland, M. Riley, J. Collado-Vides, J. D. Glasner, C. K. Rode, G. F. Mayhew, J. Gregor, N. W. Davis, H. A. Kirkpatrick, M. A. Goeden, D. J. Rose, B. Mau, and Y. Shao. The complete genome sequence of Escherichia coli K-12. *Science (New York, N.Y.)*, 277(5331):1453–1462, September 1997. ISSN 0036-8075. URL http://dx.doi.org/10.1126/science.277.5331.1453.

Jerome Bonnet, Peter Yin, Monica E. Ortiz, Pakpoom Subsoontorn, and Drew Endy. Amplifying Genetic Logic Gates. *Science*, 340(6132):599–603, May 2013. ISSN 1095-9203. doi: 10.1126/science.1232758. URL http://dx.doi.org/10.1126/science.1232758.

Dmitri Bratsun, Dmitri Volfson, Lev S. Tsimring, and Jeff Hasty. Delay-induced stochastic oscillations in gene regulation. *Proceedings of the National Academy* *of Sciences of the United States of America*, 102(41):14593–14598, 2005. URL http://www.pnas.org/content/102/41/14593.abstract.

Robert King Brayton, Alberto L. Sangiovanni-Vincentelli, Curtis T. McMullen, and Gary D. Hachtel. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1984. ISBN 0898381649.

C.F.A Bryce and D. Pacini. *The Structure and Function of Nucleic Acids*. The Biochemical Society, 1998. ISBN 0904498344.

Kevin Burrage, Pamela Burrage, Andre Leier, Tatiana Marquez-Lago, and Dan Nicolau Jr. Stochastic simulation for spatial modelling of dynamic process in a living cell. In H. Koeppl, D. Densmore, G. Setti, and M. di Bernardo, editors, *Design and Analysis of Biomolecular Circuits: Engineering Approaches to Systems and Synthetic Biology*, pages 43–62. Springer Science+Business Media, 2011. URL http://eprints.qut.edu.au/45889/.

Guilhem Chalancon, Charles N. J. Ravarani, S. Balaji, Alfonso Martinez-Arias, L. Aravind, Raja Jothi, and M. Madan Babu. Interplay between gene expression noise and regulatory network architecture. *TRENDS IN GENETICS*, 28 (5):221–232, 2012. ISSN 01689525.

Deepak Chandran, Frank Bergmann, and Herbert Sauro. TinkerCell: modular CAD tool for synthetic biology. *Journal of Biological Engineering*, 3(1):19+, 2009. ISSN 1754-1611. doi: 10.1186/1754-1611-3-19. URL http://dx.doi.org/10.1186/1754-1611-3-19.

Abhijit Chatterjee, Dionisios G. Vlachos, and Markos A. Katsoulakis. Binomial distribution based Ï„-leap accelerated stochastic simulation. *The Journal of Chemical Physics*, 122(2):024112, 2005. doi: http://dx.doi.org/10.1063/1.1833357. URL http://scitation.aip.org/content/aip/journal/jcp/122/2/10.1063/1.1833357.

Satrajit Chatterjee. *On Algorithms for Technology Mapping*. PhD thesis, EECS Department, University of California, Berkeley, Aug 2007. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-100.html.

Wei chun Chou, Peter A. Beerel, and Kenneth Y. Yun. Average-case technology mapping of asynchronous burst-mode circuits. *IEEE TRANS. ON CAD*, 18: 143–4, 1999.

F. H. C. Crick. Central dogma of molecular biology. *Nature*, 227(5258):561–563, 1970. doi: 10.1038/227561a0. URL http://dx.doi.org/10.1038/227561a0.

D. Densmore and S. Hassoun. Design automation for synthetic biological systems. *Design Test of Computers, IEEE*, 29(3):7–20, 2012. ISSN 0740-7475. doi: 10.1109/MDT.2012.2193370.

Andreas Dräger, Nicolas Rodriguez, Marine Dumousseau, Alexander Dörr, Clemens Wrzodek, Nicolas Le Novère, Andreas Zell, and Michael Hucka. Jsbml: a flexible java library for working with sbml. *Bioinformatics - Oxford*, 27(15):2167, 2013. ISSN 13674803, 13674811.

M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *NATURE -LONDON-*, (6767):335–338, 2000. ISSN 0028-0836, 00280836, 14764687.

Access Excellence. Restriction enzyme - action of ecori. http://www.accessexcellence.org/RC/VL/GG/restriction.php.

Andrew Martin Finney, Mike Hucka, Herbert M. Sauro, John Doyle, Hiroaki Kitano, and Hamid Bolouri. The systems biology markup language. *Molecular Biology of the Cell*, 12(Supplement):130a, 2001. ISSN 10591524, 19394586.

Forluvoft. Simple transcription elongation. https://en.wikipedia.org/wiki/File:Simple_transcription_elongation1.svg, October 2007.

Kurt Frenzel. Lecture 9 from ece 349 - background study in digital computer fundamentals. URL http://www.mrc.uidaho.edu/mrc/people/jff/349/lect.09.

M.A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A*, 104(9):1876–1889, 2000. ISSN 1089-5639. URL http://pubs3.acs.org/acs/journals/doilookup?in_doi=10.1021/jp993732q.

D. T. Gillespie. Stochastic simulation of chemical kinetics. *ANNUAL REVIEW OF PHYSICAL CHEMISTRY*, 58:35–56, 2007. ISSN 0066-426X, 0066426x, 15451593.

Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977. doi: 10.1021/j100540a008. URL http://pubs.acs.org/doi/abs/10.1021/j100540a008.

F. T. Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *J Chem Phys*, 115:1716–1733, 2001.

Dave Goldsman. Simulation output analysis, lecture slides module 9, 2010. URL http://www2.isye.gatech.edu/~sman/courses/Mexico2010/Module09-OutputAnalysis_100526.pdf.

Michael J. Gregory. Course: General biology 1 (bio 101). http://faculty.clintoncc.suny.edu/faculty/michael.gregory/files/Bio%20101/bio_1_menu.htm, September 2013.

M Heiner, D Gilbert, and R Donaldson. *Petri Nets for Systems and Synthetic Biology*, volume 5016 of *LNCS*, pages 215–264. Springer, 2008. URL http://dx.doi.org/10.1007/978-3-540-68894-5_7.

Monika Heiner, Mostafa Herajy, Fei Liu, Christian Rohr, and Martin Schwarick. Snoopy - a unifying petri net tool. In *Petri Nets*, pages 398–407, 2012a.

Monika Heiner, Mostafa Herajy, Fei Liu, Christian Rohr, and Martin Schwarick. Snoopy a unifying petri net tool. In Serge Haddad and Lucia Pomello, editors, *Application and Theory of Petri Nets*, volume 7347 of *Lecture Notes in Computer Science*, pages 398–407. Springer Berlin Heidelberg, 2012b. ISBN 978-3-642-31130-7. doi: 10.1007/978-3-642-31131-4_22. URL http://dx.doi.org/10.1007/978-3-642-31131-4_22.

Mostafa Herajy and Monika Heiner. Hybrid representation and simulation of stiff biochemical networks. *Nonlinear Analysis: Hybrid Systems*, 6(4):942–959, November 2012. URL http://dx.doi.org/10.1016/j.nahs.2012.05.004.

R. Hershberg, S. Altuvia, and H. Margalit. A survey of small rna-encoding genes in escherichia coli. *NUCLEIC ACIDS RESEARCH*, 31(7):1813–1820, 2003. ISSN 03051048, 13624962. doi: 10.1093/nar/gkg297.

Thomas P. Howard, Sabine Middelhaufe, Karen Moore, Christoph Edner, Dagmara M. Kolak, George N. Taylor, David A. Parker, Rob Lee, Nicholas Smirnoff, Stephen J. Aves, and John Love. Synthesis of customized petroleum-replica fuel molecules by targeted modification of free fatty acid pools in Escherichia coli. *Proceedings of the National Academy of Sciences*, 110(19): 7636–7641, May 2013. ISSN 1091-6490. doi: 10.1073/pnas.1215966110. URL http://dx.doi.org/10.1073/pnas.1215966110.

Haiyao Huang, Swapnil Bhatia, Ahmad Khalil, and Douglas Densmore. Fluigi: a computer aided design framework for combining microfluidics and synthetic biology. poster presented at the 6th International Meeting on Synthetic Biology (SB6.0), July 2013. URL http://cidarlab.org/wp-content/uploads/2013/09/HuangSB62013.pdf.

Michael Hucka, Frank T. Bergmann, Stefan Hoops, Sarah M. Keating, Sven Sahle, James C. Schaff, Lucian P. Smith, and Darren J. Wilkinson. The systems biology markup language (sbml): Language specification for level 3 version 1 core. 2010.

National Human Genome Research Institute. Fact sheets about genetic and genomic science. http://www.genome.gov/10000202#al-1, September 2013.

Gao Jiang Jiang, M. Gao, J h. Jiang, Y. Jiang, Y. Li, A. Mishchenko, S. Sinha, T. Villa, and R. Brayton. Optimization of multi-valued multi-level networks, 2002.

Wetterstrand KA. Dna sequencing costs. http://www.genome.gov/sequencingcosts/, 2014.

Gerald Karp. *Cell and Molecular Biology: Concepts and Experiments (Karp, Cell and Molecular Biology)*. Wiley, 2009. ISBN 0470483377.

Jay D. Keasling, Dae-Kyun Ro, Eric M. Paradise, Mario Ouellet, Karl J. Fisher, Karyn L. Newman, John M. Ndungu, Kimberly A. Ho, Rachel A. Eachus, Timothy S. Ham, James Kirby, Michelle C. Y. Chang, Sydnor T. Withers, Yoichiro Shiba, and Richmond Sarpong. Production of the antimalarial drug precursor artemisinic acid in engineered yeast. *Nature*, 440(7086):940–943, 2006.

Kurt Keutzer. Dagon: Technology binding and local optimization by dag matching. In *DAC*, pages 341–347, 1987. URL http://dblp.uni-trier.de/db/conf/dac/dac87.html#Keutzer87.

Ivan Komarov and Roshan M. D'Souza. Accelerating the gillespie exact stochastic simulation algorithm using hybrid parallel execution on graphics processing units. *PLoS ONE*, 7(11), 11 2012.

Erel Levine and Terence Hwa. Stochastic fluctuations in metabolic pathways. *Proceedings of the National Academy of Sciences*, 104(22):9224–9229, May 2007. URL http://dx.doi.org/10.1073/pnas.0610987104.

Gabriele Lillacci and Mustafa Khammash. Parameter estimation and model selection in computational biology. *PLoS Computational Biology*, 6(3):–, 2010. ISSN 1553734X, 15537358. doi: 10.1371/journal.pcbi.1000696.

Lin Liu, Yinhu Li, Siliang Li, Ni Hu, Yimin He, Ray Pong, Danni Lin, Lihua Lu, and Maggie Law. Comparison of next-generation sequencing systems. *Journal of biomedicine & biotechnology*, 2012:1–11, 2012. ISSN 1110-7251. URL http://dx.doi.org/10.1155/2012/251364.

Huma M. Lodhi and Stephen. Muggleton. *Elements of computational systems biology*. Wiley, 2010. ISBN 0470180935, 9780470180938. URL http://vcp.med.harvard.edu/papers/misb.pdf.

Ting Lu, Dmitri Volfson, Lev Tsimring, and Jeff Hasty. Cellular growth and division in the gillespie algorithm. In *IEE Systems Biology*, pages 121–128, 2004.

Mario A. Marchisio and Jörg Stelling. Automatic design of digital synthetic gene circuits. *PLoS Computational Biology*, 7(2), 2011. URL http://dblp.uni-trier.de/db/journals/ploscb/ploscb7.html#MarchisioS11.

H. H. McAdams and A. Arkin. Stochastic mechanisms in gene expression. *PROCEEDINGS- NATIONAL ACADEMY OF SCIENCES USA*, 94(3):814–819, 1997. ISSN 0027-8424.

L.J. McBride and M.H. Caruthers. An investigation of several deoxynucleoside phosphoramidites useful for synthesizing deoxyoligonucleotides. *Tetrahedron Letters*, 24(3):245 – 248, 1983. ISSN 0040-4039. doi: http://dx.doi.org/10.1016/S0040-4039(00)81376-3. URL http://www.sciencedirect.com/science/article/pii/S0040403900813763.

E. J. Mccluskey. Minimization of Boolean functions. *The Bell System Technical Journal*, 35(5):1417–1444, November 1956.

U.S. National Library Of Medicine. Handbook - help me understand genetics. http://ghr.nlm.nih.gov/handbook.pdf, September 2013.

S. Menz, J. Latorre, C. Schatte, and W. Huisinga. Hybrid stochastic–deterministic solution of the chemical master equation. *Multiscale Modeling & Simulation*, 10(4):1232–1262, 2012. doi: 10.1137/110825716. URL http://epubs.siam.org/doi/abs/10.1137/110825716.

Ron Milo, Paul Jorgensen, Uri Moran, Griffin Weber, and Michael Springer. Bionumbers-the database of key numbers in molecular and cell biology. *NUCLEIC ACIDS RESEARCH*, 38(Suppl. 1):D750–D753, 2010. ISSN 03051048, 13624962.

Munsky and Khammash. The finite state projection algorithm for the solution of the chemical master equation. *Journal of Chemical Physics*, 124(4):44104–1–13, 2006. ISSN 00219606, 10897690.

Saburo Muroga. *Logic Design and Switching Theory*. John Wiley & Sons, Inc., New York, NY, USA, 1979. ISBN 0471044180.

Chris J. Myers, Nathan Barker, Kevin Jones, Hiroyuki Kuwahara, Curtis Madsen, and Nam-Phuong D. Nguyen. ibiosim. *Bioinformatics*, 25(21):2848–2849, November 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp457. URL http://dx.doi.org/10.1093/bioinformatics/btp457.

C.J. Myers. *Engineering Genetic Circuits*. Chapman & Hall/CRC Mathematical and Computational Biology. Taylor & Francis, 2011. ISBN 9781420083255. URL http://books.google.dk/books?id=UoCq4FeTeHkC.

Nagarajan Nandagopal and Michael B. Elowitz. Synthetic biology: Integrated gene circuits. 333(6047):1244–1248, 2011. doi: 10.1126/science.1207084.

Carl Rod Nave. Translation or protein synthesis. http://hyperphysics.phy-astr.gsu.edu/hbase/organic/translation.html, October 2013.

C.A. Nelson. *Technology Mapping of Timed Asynchronous Circuits*. Department of Electrical and Computer Engineering, University of Utah, 2004. URL http://books.google.dk/books?id=I86XYgEACAAJ.

Michael Pedersen and Andrew Phillips. Towards programming languages for genetic engineering of living cells. *Journal of The Royal Society Interface*, 6, April 2009. URL http://rsif.royalsocietypublishing.org/content/early/2009/04/14/rsif.2008.0516.focus.

C. Adam Petri and W. Reisig. Petri net. 3(4):6477, 2008.

Carl Adam Petri. *Communication with automata*. PhD thesis, Universität Hamburg, 1966.

Jacqueline Quinn, Jacob Beal, Swapnil Bhatia, Patrick Cai, Joanna Chen, Kevin Clancy, Nathan Hillson, Michal Galdzicki, Akshay Maheshwari, Umesh P, Matthew Pocock, Cesar Rodriguez, Guy-Bart Stan, and Drew Endy. Synthetic biology open language visual (sbol visual), version 1.0.0. bbf rfc 93. http://hdl.handle.net/1721.1/78249, 2013.

T A RAJU. Lac operon. http://en.wikipedia.org/wiki/Regulation_of_gene_expression.

R. Rauhut and G. Klug. mrna degradation in bacteria. *FEMS MICROBIOLOGY REVIEWS*, 23(3):353–370, 1999. ISSN 01686445, 15746976. doi: 10.1016/S0168-6445(99)00012-1.

Virgil A. Rhodius, Thomas H. Segall-Shapiro, Brian D. Sharon, Amar Ghodasara, Ekaterina Orlova, Hannah Tabakh, David H. Burkhardt, Kevin Clancy, Todd C. Peterson, Carol A. Gross, and Christopher A. Voigt. Design of orthogonal genetic switches based on a crosstalk map of sigmas, anti-sigmas, and promoters. *Molecular Systems Biology*, 9(1), October 2013. URL http://dx.doi.org/10.1038/msb.2013.58.

Andre Ribeiro, Rui Zhu, and Stuart A. Kauffman. A general modeling strategy for gene regulatory networks with stochastic dynamics. *JOURNAL OF COMPUTATIONAL BIOLOGY*, 13(9):1630–1639, 2006. ISSN 10665277, 15578666. doi: 10.1089/cmb.2006.13.1630.

Marc R. Roussel and Rui Zhu. Stochastic kinetics description of a simple transcription model. *BULLETIN OF MATHEMATICAL BIOLOGY*, 68(7):1681–1713, 2006. ISSN 00928240, 15229602. doi: 10.1007/s11538-005-9048-6.

R. Sarpeshkar. Brain power - borrowing from biology makes for low power computing [bionic ear]. *Spectrum, IEEE*, 43(5):24–29, 2006. ISSN 0018-9235. doi: 10.1109/MSPEC.2006.1628504.

Yishai Shimoni, Gilgi Friedlander, Guy Hetzroni, Gali Niv, Shoshy Altuvia, Ofer Biham, and Hanah Margalit. Regulation of gene expression by small non-coding rnas: a quantitative view. *Molecular Systems Biology*, 3:138, 2007. ISSN 17444292.

Polly Siegel, Giovanni De Micheli, and David L. Dill. Automatic technology mapping for generalized fundamental-mode asynchronous designs. In *DAC*, pages 61–67, 1993. URL http://dblp.uni-trier.de/db/conf/dac/dac93.html#SiegelMD93.

Alexander Slepoy, Aidan P. Thompson, and Steven J. Plimpton. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks. *The Journal of Chemical Physics*, 128(20):205101, 2008. doi: http://dx.doi.org/10.1063/1.2919546. URL http://scitation.aip.org/content/aip/journal/jcp/128/20/10.1063/1.2919546.

Stricker, Cookson, Bennett, Mather, Tsimring, and Hasty. A fast, robust and tunable synthetic gene oscillator. *Nature*, 456(7221):516–519, 2008. ISSN 00280836, 14764687.

Alvin Tamsir, Jeffrey J. Tabor, and Christopher A. Voigt. Robust multicellular computing using genetically encoded nor gates and chemical 'wires'. *NATURE*, 469(7329):212–215, 2011. ISSN 00280836, 14764687. doi: 10.1038/nature09565.

Wayne A. Taylor. Change-point analysis: A powerful new tool for detecting changes, 2000. URL http://www.variation.com/cpa/tech/pattern.html.

Tianhai Tian and Kevin Burrage. Binomial leap methods for simulating stochastic chemical kinetics. *The Journal of Chemical Physics*, 121(21), 2004.

P. Waage and CM Gulberg. Studies concerning affinity. *JOURNAL OF CHEMICAL EDUCATION*, 63(12):1044–1047, 1864. ISSN 00219584, 19381328. doi: 10.1021/ed063p1044.

Ron Weiss, George E. Homsy, and Thomas F. Knight. *Toward in vivo Digital Circuits*, pages 275–279. DIMACS Workshop. Springer, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.9888.

Ron Weiss, Subhayu Basu, and David Braun. Parameter estimation for two synthetic gene networks: A case study. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, V:V769–V772, 2005. ISSN 15206149. doi: 10.1109/ICASSP.2005.1416417.

Jialiang Wu, Brani Vidakovic, and Eberhard O. Voit. Constructing stochastic models from deterministic process equations by propensity adjustment. *BMC systems biology*, 5(1):187+, 2011. ISSN 1752-0509. doi: 10.1186/1752-0509-5-187. URL http://dx.doi.org/10.1186/1752-0509-5-187.

Dennis C. Wylie, Yuko Hori, Aaron R. Dinner, and Arup K. Chakraborty. A hybrid deterministicâ^'stochastic algorithm for modeling cell signaling dynamics in spatially inhomogeneous environments and under the influence of external fields. *The Journal of Physical Chemistry B*, 110(25):12749–12765, 2006. doi: 10.1021/jp056231f. URL http://pubs.acs.org/doi/abs/10.1021/jp056231f. PMID: 16800611.

Xilinx. Ise. http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm.

D.A. Zaitsev. Toward the minimal universal petri net, 2013. ISSN 2168-2216.

Hornung Ákos. A simple explanation of how the dna can be read by the transcription machinery. https://en.wikipedia.org/wiki/File:Reading_Frame.png, December 2011.

Glossary

Asynchronous circuits A digital circuit that operates without clock signals. In theory these circuits can be faster and more power efficient but practical issues concerning hazard-handling often remove these advantages. 103, 111

Bimolecular reaction Reactions with two reactants, i.e. $S_i + S_j \to products$. 39

Biochemical reactions Chemical reactions happening in the cell. 46

Brownian dynamics A simplified model of molecular motion that can be described solely by radius, volume and speed. 39, 104, 107

DeMorgan's theorem Rules of propositional negation that states *The negation of a conjunction is the disjunction of the negations* and *The negation of a disjunction is the conjunction of the negations.* So for example any OR-gate is equivalent to an AND-gate with its inputs and output negated. 88

Enzyme A protein that catalyses chemical reactions. 12

IGEM The *international genetically engineered machine competition* a student competition in synthetic biology. 2

Markov process A stochastic memory-less process that can make prediction of the future based solely on the current state. 40

Monomolecular reaction Reactions with one reactant, i.e. $S_i \to products$. 39

Real-world Typically real-world experiments refer to experiments in real living organisms. 25, 107

Sum-of-products A convenient way to express boolean expression using only AND, OR or NEG operators in a specific arrangement. 80

Synchronous circuits A digital circuit that is synchronised by a clock so that every execution is performed in clock-steps. This type of circuit is very common and is used in nearly every consumer product. 103

Turing complete A term meaning that this programming language or automata can simulate any other programming language or automata. 30

Wet-lab Typically wet-lab experiments refer to experiments in glass, i.e. in an artificial laboratory setting where the unit of interest is isolated. 22, 49, 80, 107