VISUALIZATION IN THE SYNTHETIC BIOLOGY DESIGN CYCLE

JAMES SCOTT-BROWN
Department of Engineering Science
& Linacre College

A thesis submitted for the degree of
Doctor of Philosophy

Dedicated to my family.

ABSTRACT

Synthetic biology is an emerging field with the ultimate aim of designing and building biological systems with particular functions. Achieving this goal requires parallel developments in several areas: experimental methods, libraries of reusable parts mined from natural systems, approaches to mathematical modelling, and design tools. As the biological circuits that are designed and built become more complex, automation becomes increasingly important at every stage.

Appropriate use of visualization can facilitate the development of synthetic biology by aiding both the understanding of data and interaction with automation tools. This thesis therefore introduces visual tools for several tasks in the synthetic biology design-build-test-cycle, using diagrammatic representations that are designed to be clear to human readers whilst also having a precise meaning. These tools are intended to enable the expression of temporal logic specifications, the identification of models and parameters that cause these specifications to be satisfied, and the expression of laboratory protocols that could be used to experimentally test the resulting designs.

In this thesis we first develop two diagrammatic representations: the `TimeRails` representation of specifications provides an abstraction for representing *sets* of signals, and the `List of Liquids` representation provides a representation of laboratory protocols for conducting experiments.

This thesis also describes two approaches for the automated design of a circuit to meet a specification expressed using `TimeRails`. The first approach constructs a satisfaction problem *modulo* the theory of Ordinary Differential Equations. The second approach uses Approximate Bayesian Computation to sample from the posterior distribution over parameters, given that the system meets the specification; the results of this inference process are presented in a visual tool that can also be applied to model selection and parameter estimation more generally.

PUBLICATIONS

The work described in this thesis has led to the following publications:

[1] **James Scott-Brown** & Antonis Papachristodoulou. "sbml-diff: A Tool for Visually Comparing SBML Models in Synthetic Biology." In: *ACS Synthetic Biology* (2017). DOI: 10.1021/acssynbio.6b00273.

[2] **James Scott-Brown** & Antonis Papachristodoulou. "Bayesian Designer." In: *In preparation* (2018).

[3] **James Scott-Brown** & Antonis Papachristodoulou. "Visual Representation of Experimental Protocols." In: *Submitted* (2018).

[4] Marta Kwiatkowska, Antonis Papachristodoulou, James Scott-Brown, and Max Whitby. "CRNSynth: A Tool for the Automated Synthesis of Chemical Reaction Networks." In: *In preparation* (2018).

Work at earlier stages was also presented at conferences:

[1] **James Scott-Brown** and Antonis Papachristodoulou. "Visual Expression of Experimental Protocols." In: *9th International Workshop on Bio-Design Automation*. 2017.

[2] **James Scott-Brown** and Antonis Papachristodoulou. "Visualization of Temporal Logic Specifications." In: *EuroVis 2017 - Posters*. Ed. by Anna Puig Puig and Tobias Isenberg. The Eurographics Association, 2017. ISBN: 978-3-03868-044-4. DOI: 10.2312/eurp.20171183.

[3] **James Scott-Brown**, Thomas Prescott, and Antonis Papachristodoulou. "TEBio - Tools for Engineering Biology." In: *8th International Workshop on Bio-Design Automation*. 2016.

My name is written in bold, except for publications that follow the convention of alphabetical author order.

ACKNOWLEDGMENTS

Personally, I would like to thank:

My parents, for their support whilst I was conducting the work described in this thesis, particularly whilst our house was being concurrently rebuilt.

My wife, Rachel Croft, for all of her help and support throughout my time in Oxford (and before).

All of my teachers and lecturers during the 22 years that I have spent as a student: they have all helped me to get to where I am now.

Madan Babu Mohan, whose suggestion that I do a summer project in his group as an undergraduate turned into my first real experience of biological visualization design.

Chris Myers, for many useful conversations and suggestions, which in particular led to the collaboration with Max Whitby that resulted in Chapter 3.

My supervisor, Professor Antonis Papachristodoulou, for providing help and advice when it was needed, whilst also allowing me a remarkable degree of freedom to follow a research direction that has extended far beyond what would be expected given the location of my desk on the Control Corridor.

My industrial co-supervisor at Dstl, Jon David, for his help.

FUNDING

This work was funded by the EPSRC/BBSRC Centre for Doctoral Training in Synthetic Biology (EP/L016494/1), and by Dstl.

I would also like to thank the following organisations for financially supporting my participation in their events:

- the Institute for Mathematics and its Applications, for the IMA Workshop on Biological Systems and Networks in November 2015
- the BBSRC Biological Visualisation Network, for the 3rd BiVi Annual Meeting in April 2017
- the Bio-Design Automation Consortium, for the 9th International Workshop on Bio-Design Automation in August 2017
- the Leibniz-Zentrum fur Informatik, for Dagstuhl Seminar 18082 (Formal Methods for the Synthesis of Biomolecular Circuits) in February 2018

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

LIST OF ALGORITHMS

See also the sampling algorithms listed in Table 4.1, and the DPLL algorithm represented in Table 3.1.

ACRONYMS

ABC Approximate Bayesian Computation

CRN Chemical Reaction Network

CSV Comma Separated Values

CUDA Complete Unified Device Architecture

DNA Deoxyribonucleic Acid

DPLL Davis–Putnam–Logemann–Loveland algorithm

GPU Graphical Processing Unit

IDE Integrated Development Environment

IPSep-CoLa Incremental Procedure for Separation Constraint Layout of graphs

MathML Mathematical Markup Language

mRNA messenger RNA

ODE Ordinary Differential Equation

PCR Polymerase Chain Reaction

PDE Partial Differential Equation

RBS Ribosome Binding Site

RNA Ribonucleic Acid

SAT (Boolean) SATisfaction

SBGN Systems Biology Graphical Notation

SBML Systems Biology Markup Language

SBO Systems Biology Ontology

SBOL Synthetic Biology Open Language

SDE Stochastic Differential Equation

SMC Sequential Monte Carlo

SMT SATisfaction *Modulo* Theory

SPLOM ScatterPLOt Matrix

STL Signal Temporal Logic

SVG Scalable Vector Graphic

VNODE-LP Validated Numerical ODE solver developed using Literate Programming

XML eXtensible Markup Language

NOTATION

$\vee$	Logical disjunction (or)
$\wedge$	Logical conjunction (and)
$\neg$	Logical negation (not)
$\Longrightarrow$	Logical implication
$\Leftrightarrow$	Logical equivalence
$\models$	Satisfies/Models
$\forall$	For all
$\exists$	There exists
::=	symbol (left) may be replaced by expression (right)
|	represents an alternative
U	Until
$\Box$	Globally
$\diamond$	Finally

See also the explanation of probability notation in Appendix A.

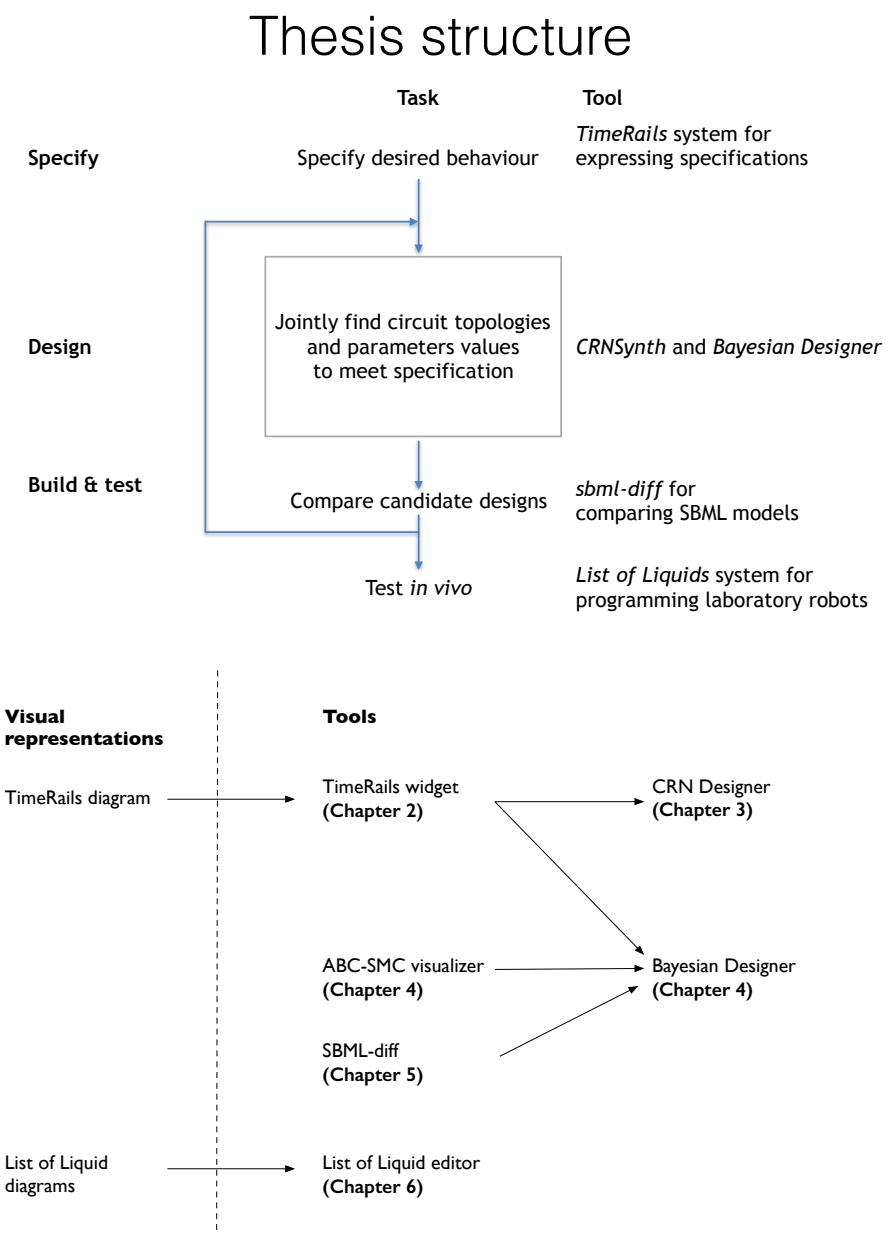


Figure 0.1: Two views of the structure of this thesis.

1

INTRODUCTION

The overarching question considered by this thesis is: how can tasks in the synthetic biology design-build-test cycle be supported by the combination of diagrams with precise semantics and tools that work with these diagrams?

Synthetic biology has the potential to produced engineered organisms that are able to address major challenges, but many tasks in synthetic biology are made unnecessarily difficult by the lack of good visual representations. The relationship between the representations and tools described in different sections of this thesis is summarised in Figure 0.1.

This introductory chapter provides background information that is relevant to all chapters of this thesis: Section 1.1 introduces the field of Synthetic Biology, Section 1.2 describes (in molecular detail) the biology that is manipulated by synthetic biologists, and Section 1.3 reviews the different kinds of mathematical models that are typically applied to biochemical networks. In addition to this general introduction, which is relevant to all of the work described in this thesis, further introductory material that is relevant only to a particular chapter is included in the corresponding chapter. This includes an overview of formal methods and specifications in Chapter 2, SAT and SMT constraint-satisfaction problems in Chapter 3, Approximate Bayesian Computing in Chapter 4, SBML in Chapter 5, and laboratory automation in Chapter 6. The chapter concludes in Section 1.7 with a description of the structure of this thesis, and a summary of its contributions.

1.1 SYNTHETIC BIOLOGY

The term 'synthetic biology' was first used in the early twentieth century by Stèphane-Armand Nicolas Leduc, who wrote that "The task of synthetic biology is the recognition of those physicochemical conditions which can produce forms and structures analogous to those of living beings" [14], a definition that would exclude most of what is today considered synthetic biology. Such differences between the initial and current meanings of a subfield have also occurred in other areas of biology—for example, the original definition of 'bioinformatics' as the study of informatic processes in biotic systems is very different to its current use, which refers mostly to the analysis of genetic or protein sequence data [66].

There are various contemporary definitions of 'synthetic biology' [157], but the following, from [78], is typical:

> Synthetic biology is the design and construction of new biological entities, such as enzymes, genetic circuits, and cells, or the redesign of existing biological systems. The goal of synthetic biology, which builds on advances in molecular, cellular, and systems biology, is to transform biology in the same way that synthesis transformed chemistry and integrated circuit design transformed computing.

The field is broad: it encompasses a wide range of research questions, approaches, and applications, and includes such diverse areas as protein engineering and Deoxyribonucleic Acid (DNA) nanotechnology, metabolic engineering, and the design of biological circuits to function inside cells. However, these are unified by a defining characteristic: the use of a *rational engineering approach* to manipulate biology. For a review of the early history of the field [1961–2013] see [22].

Synthetic biology has attracted a great deal of interest. David Willets, as Minister of State for Universities and Science, stated that "Synthetic biology has huge potential. Indeed it has been said that it will heal us, feed us and fuel us"[118]. It does indeed have this potential:

metabolic engineering has the potential to produce drugs, foodstuffs, and biofuels, whilst synthetic circuits have the potential to provide diagnostic devices and cellular therapeutics.

Metabolic engineering

The area of synthetic biology that has the greatest impact so far is arguably metabolic engineering, which is concerned with modifying a host organism (typically a bacterium or a yeast) so that it produces a specific chemical that it would not naturally produce. This provides an alternative to producing fine chemicals using more traditional synthetic chemistry, potentially at lower cost or with reduced environmental impact. Metabolic engineering as a subfield of synthetic biology is distinguished from earlier 'genetic engineering' by its increased ambition and number of modifications made to the host organism: an early highlight, the production of artemisinic acid in engineered yeast [126], required directly upregulating three genes, indirectly upregulating many more by upregulating a transcription factor, and introducing 2 genes from the plant *A. annua*.

The development of enabling technologies has enabled faster progress in metabolic engineering: the DARPA pressure test found that an organism capable of producing 6 of 10 challenge molecules could be developed within 90 days [24].

Genetic circuits

Another major subfield of synthetic biology is concerned with the creation of *synthetic circuits* that perform some desired function. This rose to prominence with the 20th January 2000 issue of *Nature*, which included two seminal papers describing the construction of a genetic ring-oscillator (the 'repressilator') [39] and a genetic toggle-switch [50]. The repressilator circuit is analogous to a ring-oscillator in electronics, in which oscillations arise when an odd number of NOT gates are connected in a loop. It consists of three genes (*tetR*, λcI, *lacI*), each of

which is a transcription factor that binds to a corresponding sequence of DNA (a promoter) and prevents the downstream gene from being transcribed. These are arranged such that LacI protein represses the production of TetR protein, which represses the production of λcI protein, which represses the production of LacI protein, resulting in the concentrations of each protein oscillating in turn. The toggle switch consisted of two transcription factors, each of which represses production of the other unless a specific small molecule was added to prevent this repression. Addition of one of these input molecules would 'toggle' the switch into the state where it was expressing one transcription factor at a high concentration, and the other at a low concentration; it would remain in this state after the input was removed, until the opposite input was provided. Models of both the repressilator and toggle switch as both shown in Figure 5.3, where they provide an illustration of the capabilities of the `sbml-diff` tool introduced in Chapter 5.

Transcriptional regulation is not the only mechanism by which genetic circuits can operate: alternatives include the use of regulatory Ribonucleic Acid (RNA) molecules, and the use of integrase proteins that can either excise or reverse the orientation of a section of DNA between two specific sequences (recombination sites) depending on the orientation of the recombination sites. For a broad review of the types of genetic circuits that can be built, see [21].

Genetic circuits attracted the interest of many researchers from an engineering background, and many analogies have been made between synthetic biology and electrical engineering (such as the desire for abstraction, well-characterised parts, signal isolation/cross-talk minimization, and new biological equivalents to existing Electronic Design Automation tools).

Genetic circuits have not yet had the same industrial impact that metabolic engineering has, but there is the potential for practical applications in the future. One potential application is to medical diagnostics and therapeutics, where it may be desirable to diagnose a particular disease state using a circuit implementing a multi-input decision rule [161], cyclically release a therapeutic agent by lysis of

the chassis organism controlled by a synchronized oscillator [32], or directionally move towards a pathogen [72] or tumor. For reviews of the potential of engineered-cell therapies produced using a synthetic-biology approach see [82, 138, 163].

Aside from practical applications, a major motivation for creating synthetic circuits is to increase our understanding of how natural circuits function in biology, either to explore 'design principles' generally [97] or in the specific context of development [29]. As well as allowing the construction and experimental investigation of minimal genetic circuits, synthetic biology allows otherwise impossible experiments to be performed on natural circuits, by allowing precise temporal control of the concentration of particular proteins within the cell [43, 146].

1.2 MOLECULAR BIOLOGY

Space constraints limit this section to a brief overview; for a detailed introduction to molecular biology, please consult standard undergraduate textbooks such as [1 📖, 15 📖, 88 📖, 156 📖].

The functional unit of life is the cell, which is delineated by a *cell membrane* composed of a bilayer of amphipathic phospholipids. Within the cell, hereditary information is stored in the form of DNA, a polymer consisting of four different monomers (bases): adenine (commonly denoted A), thymine (T), cysteine (C), and guanine (G). DNA adopts a double helical structure, in which two complementary strands are held together in an anti-parallel orientation, with pairing between complementary bases (A pairs with T, and C pairs with G).

Proteins, which are polymers of amino acids, are another important class of biological macromolecules. There are 20 proteogenic amino acids: because they are made from a more chemically diverse set of monomers than are nucleic acids, proteins adopt a wider range of structures, and fulfill a wider range of functions. Almost all chemical reactions within the cell are catalysed by proteins known as enzymes: this catalysis not only increases the rate at which reactions occur, but also provides specificity, preventing side-reactions. Proteins also per-

form structural functions (e.g., collagen or keratin), and fulfill a variety of other roles, such as transporting substances across membranes, sensing the external environment, and producing movement. The sequence of amino acids determines the structure into which a protein will fold, and hence the function it will perform.

The DNA is organised into genes, and the sequence of base pairs in the gene encodes the amino acid sequence of a protein; the mapping from nucleotide triplets to amino acids is known as the *genetic code*, and is almost identical across all organisms. A triplet of bases specifies a single amino acid (there are $4^3 = 64$ such triplets, allowing the specification of 20 amino acids with some redundancy), or a 'start' or 'stop' codon. Other sections of DNA are important for the regulation of gene expression: they contain sequences that are recognised by specific proteins (*transcription factors*), which either *activate* or *repress* expression of particular genes, by recruiting or blocking the RNA polymerase enzyme.

RNA polymerase is an enzyme that uses the double-stranded DNA as a template to produce a complementary single strand of messenger RNA (mRNA). RNA is chemically similar to DNA, but has a hydroxyl group instead of hydrogen atom in the 2′ position (hence the *deoxy* in the name DNA), which makes it chemically less stable; additionally, it uses a uracil (U) in place of thymine (T). The *ribosome*, which is a complex containing protein and RNA components, then binds to a region of the mRNA called the Ribosome Binding Site (RBS), 'reads' the mRNA, and produces the corresponding protein by sequentially adding amino acids to the end of an elongating chain.

The *central dogma of molecular biology* states that there is an irreversible flow of information from genetic sequence to protein sequence. Reverse transcriptase can produce RNA from a DNA template, but there is no biological mechanism to produce either DNA or RNA from a protein template.

Since gene products are able to regulate gene expression, it is possible to form feedback loops that give rise to useful behaviours, including bistability (e.g., the lac operon in nature and Gardner's synthetic

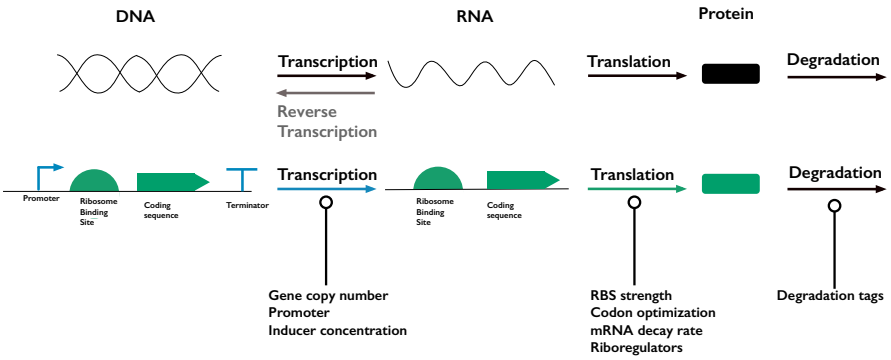


Figure 1.1: DNA is transcribed into RNA, which is translated into protein. At each stage there is the potential for regulation. Some of the factors affecting rates are shown in the diagram; for a further discussion of how rates can be tuned see [4].

toggle switch [50]), oscillations (e.g., Elowitz's repressilator [39]), pulsing/excitation, or adaptation.

1.3 MODELING

> *To make progress in understanding all this, we probably need to begin with simplified (oversimplified?) models and ignore the critics' tirade that the real world is more complex. The real world is always more complex, which has the advantage that we shan't run out of work.*
>
> John Ball [6]

For synthetic biology to become a true engineering discipline, it is necessary to have quantitative mathematical models that can be used to predict the behaviour of a system before it is built, and to predict the effect of making specific changes such as altering the values of particular parameters. In order to be useful, a model must capture a sufficient level of detail to enable accurate predictions (at least within some domain of validity), whilst also remaining simple enough for analysis to be feasible. All models are necessarily simplifications that omit much detail.

For an introduction to mathematical models of gene networks, see the review articles [16, 58, 62, 74, 103, 119, 120], and the textbook [155 📖]. For an overview of stochastic modeling, see the monographs [159 📖, 167 📖] and the article [38].

One important choice is where to draw the boundary between the system being modeled and its environment. There have been attempts at constructing whole cell models (most notably [76]), but such models will inevitably contain a huge amount of uncertainty. Many models of genetic circuits consider only the circuit itself, ignoring (or treating as constant) the remainder of the cell. This has the advantage of simplicity, but fails to capture effects like host burden or competition for ribosomes.

A second choice is what species and reactions to include. For example, the process of producing a single RNA molecule by transcription (or protein molecule by translation) consists of a very large number of chemical reactions that occur sequentially, but is often treated as a single reaction (sometimes with a delay).

Another key choice is how to handle stochasticity. A natural representation of the state of a (bio-)chemical reaction system is a vector containing the concentration of each chemical species. The chemical species are created and destroyed in chemical reactions, which occur at rates determined by the concentration of their reactants. As Gillespie observes in his classic paper [53], to predict the full evolution of the system we would need knowledge of the position and velocity of every molecule; the projection of the system onto this vector of concentrations thus cannot evolve deterministically. There are several choices of how to handle this (Figure 1.2): as the number of molecules being modelled increases, it becomes possible to use simpler approximations whilst maintaining an acceptable degree of accuracy. The most extreme approaches are to either perform a spatial simulation that keeps track of the position of each molecule in space, or to construct a Boolean models by discretising species concentrations into 'high' or 'low' levels, but most models adopt the intermediate approach of treating the concentrations of species as real numbers (or the molecule counts of species as integers).

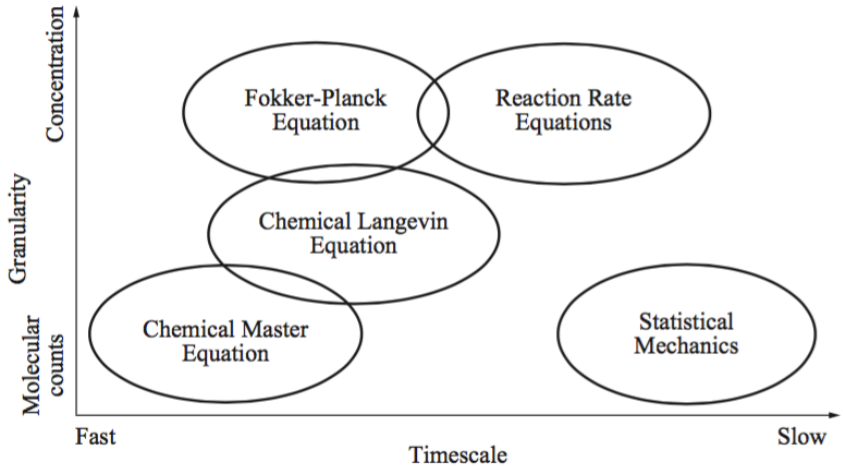


Figure 1.2: The relationship between modelling formalisms for biomolecular systems, from Figure 2.1 of [155 📖].

To avoid keeping track of the locations of each molecule, the molecule counts can be considered to evolve stochastically, as a Markov process[1].

In the following discussion, we represent the vector of molecule counts as q, and assume that the are M different reactions that can occur: the j'th reaction has a probability $a_j(q,t)\ dt$ of occurring in a short time-step dt, and results in a net change of $\xi_{i,j}$ in the i'th species (where ξ is written with a single subscript ξ_j, it represents the vector of stoichiometry changes for each species in the j'th reaction).

The evolution of the probability distribution over concentration vectors is given by a Partial Differential Equation (PDE) called the **Chemical Master Equation (CME)** or Kolmogorov equation:

$$\frac{\partial}{\partial t}P(\mathbf{q},t) = \sum_{j=1}^{M}\left(a_j(\mathbf{q}-\xi_j)P(\mathbf{q}-\xi_j,t) - a_j(\mathbf{q})P(\mathbf{q},t)\right)$$

In principle, this can be solved to obtain the probability distribution over states at each time $P(\mathbf{q},t)$, but this is feasible only for simple systems. A trajectory of the system can be exactly sampled using

1 In discrete time, a Markov process is a stochastic process in which the probability of the state having a particular value at the next time, given the state at the current time, is conditionally independent of the state at earlier times: $P(x_{t+1}|x_t) = P(x_{t+1}|x_t, x_{t-1}, x_{t-2}, \ldots, x_0)$. In other words, a Markov process is *memoryless*.

the Gillespie stochastic simulation algorithm[2] [53]; at each iteration, this samples the time until the next reaction occurs from an exponential distribution, and the identity of this reaction from a categorical distribution, then advances the simulation to that time and updates molecule counts accordingly. Other methods allow faster simulations, at the cost of decreased accuracy, by introducing approximations. One such method is tau-leaping, which estimates how many times each reaction will occur during an interval of duration τ using propensities calculated using the concentrations at the start of the interval.

The Chemical Master Equation can be approximated by a **Fokker-Planck Equation**. This is also a PDE that gives an formula for $\frac{\partial}{\partial t}P(\boldsymbol{q}, t)$, but it treats the state as a set of continuous variables and includes the derivatives of $P(\cdot)$ with respect to these on the right hand side.

At higher volumes, the system can be simulated using the **Chemical Langevin Equation (CLE)**, a Stochastic Differential Equation [54] that includes white-noise processes $\Gamma_j(t)$:

$$\frac{dX_i(t)}{dt} = \overbrace{\sum_{j=1}^{M} \xi_{ji} a_j(\mathbf{X}(t))}^{A_i(X(t))} + \sum_{j=1}^{M} \overbrace{\xi_{ji}\sqrt{a_j(\mathbf{X}(t))}}^{B_{ij}(\mathbf{X}(t))} \Gamma_j(t)$$

Simulated solutions to this Stochastic Differential Equation (SDE) are often obtained using the *Euler–Maruyama method*:

$$\mathbf{X}(t+\tau) = \mathbf{X}(t) + \tau \overbrace{\sum_{j=1}^{M} \xi_{ji} a_j(\mathbf{X}(t))}^{A_i(\mathbf{X}(t))} + \sqrt{\tau} \sum_{j=1}^{M} \overbrace{\xi_{ji}\sqrt{a_j(\mathbf{X}(t))}}^{B_{ij}(\mathbf{X}(t))} \Gamma_j(t)$$

where each element of $\Gamma_j(t)$ is randomly sampled from a zero-mean unit-variance normal distribution at each iteration.

Alternatively, it is possible to construct and solve an ODE that describes the evolution of the mean and variances for the concentration of each species (the **Linear Noise Approximation**).

At higher volumes still, the effect of the noise term becomes negligible and it becomes reasonable to perform a deterministic simula-

2 Also referred to as the Gillespie algorithm, or as the Stochastic Simulation Algorithm (SSA).

tion using Ordinary Differential Equations (the **Rate Reaction Equations**) [89]:

$$\frac{dX_i(t)}{dt} = \sum_{j=1}^{M} \xi_{ij} a_j(\mathbf{X}(t))$$

It is common to simplify models by exploiting time-scale separation to eliminate some species whose dynamics are assumed to be 'fast': some common examples of model reductions are listed in Table 1.1. However, care must be taken when applying such assumptions to stochastic models (rather than the deterministic Rate Reaction Equations) [80].

The work described in this thesis uses Rate Reaction Equations and the Linear Noise Approximation. Crnsynth (Chapter 3) requires an ODE model, so cannot be used with other modelling formalisms. However, Bayesian Designer (Chapter 4) uses interchangeable modular simulators, and as an alternative to solving these equations with an ODE solver, it can instead use the Euler-Maruyama method to solve the Chemical Langevin equation, or sample trajectories using the Gillespie algorithm. An SBML model specifies a `kineticLaw` for each reaction, but not how this should be simulated, so `sbml-diff` 5 is unaffected by the choice of modelling formalism.

1.4 APPROACHES TO AUTOMATED SYNTHETIC CIRCUIT DESIGN

Sequential design

One approach to designing a synthetic circuit is to first design the circuit topology, and then to assign specific parts from a library to each abstract part in the design (this second stage is sometimes referred to as technology mapping).

Cello [113] designs combinatorial logic circuits that are constructed using transcriptional NOR gates: a NOR-inverter graph is produced by either replacing "A AND B" with "(NOT A) NOR (NOT B)" in the AND-inverter graph produced by the synthesis tool ABC, or by transforming the Product of Sums/Conjunctive Normal Form expression

	Full Reaction scheme	Full ODE model	Assumption	Reduced model
Michaelis-Menten	$S + E \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} ES$ $ES \xrightarrow{k_2} E + P$	$\frac{d[S]}{dt} = -k_1[E][S] + k_{-1}[ES]$ $\frac{d[E]}{dt} = -k_1[E][S] + (k_{-1} + k_2)[ES]$ $\frac{d[ES]}{dt} = k_1[E][S] - k_{-1}[ES]$ $\frac{d[P]}{dt} = k_2[ES]$ $[E] + [ES] = E_{tot}$	$\frac{d[ES]}{dt} \approx 0$; $\frac{d[S]}{dt} \approx 0$ gives similar approximation but without the k_2 factor.	$\frac{d[P]}{dt} = \frac{k_2 E_{tot} + [S]}{(k_{-1} - k_2)/k_1 + [S]}$
Fast mRNA dynamics	$\emptyset \xrightarrow{k_1} mRNA$ $mRNA \xrightarrow{k_2} \emptyset$ $mRNA \xrightarrow{k_3} mRNA + protein$ $protein \xrightarrow{k_4} \emptyset$	$\frac{d[mRNA]}{dt} = k_1 - k_2[mRNA]$ $\frac{d[protein]}{dt} = k_3[mRNA] - k_4[protein]$	$k_2 >> k_1$	$\frac{d[protein]}{dt} = k_3 k_1 / k_2 - k_4[protein]$
Hill activation/ repression	$X + nS \rightleftharpoons C$	$\frac{dC}{dt} = k_1 X S^n - k_{-1} C$ $X_T = X + C$	$\frac{dC}{dt} \approx 0$	$C = \frac{X_T S^n}{S^n + (k_{-1}/k_1)}$ $X = \frac{X_T (k_{-1}/k_1)}{S^n + (k_{-1}/k_1)}$

Table 1.1: Common model reductions for deterministic models.

produced by Espresso [19 📖][3]. Specific parts are then assigned from a library using Simulated Annealing to maximise a 'circuit score' that is the ratio between the lowest predicted output level that should be ON and the highest predicted output level that should be OFF.

CALIN [57] designs multicellular circuits to implement combinatorial logical circuits that are split across multiple cells. It first re-writes the desired logic function in Disjunctive Normal Form, and then implements each term with a separate circuit in a distinct strain using integrases. The disjunction between these terms is achieved by having each strain release a signaling molecule: if any strain produces this, it will be detectable in the growth medium.

Simultaneous design

An alternative approach is to design the circuit in a single phase, by trying to directly connect specific components into a functional circuit.

One natural formulation for this problem is to consider (gene, promoter) pairs, so that a circuit is represented by a set of indicator variables $Y(i,j)$, where $Y(i,j) = 1$ if there is a copy of the j'th protein under the control of the i'th promoter. Denoting the index of the protein that regulates promoter i as $g(i)$, and making some assumptions about kinetics, the rate of transcription of protein j can be written as:

$$\sum_i Y(i,j) \frac{\alpha_i}{1 + K_i p_{g(i)}^{n_i}}$$

This gives rise to a Mixed Integer Dynamic Program: the *integers* are the $Y(i,j) \in \{0,1\}$, and the *dynamic* refers to the fact the optimization uses an objective function evaluated over the solution of a differential equation.

3 A formula in CNF consists of the conjunction of terms, each containing only negation and disjunction; a formula in DNF consists of a disjunction of terms, each containing only negation and conjunction; a NOR-inverter graph includes only NOR and negation; an AND-inverter graph includes only conjunction and negation. As a concrete example, consider the logical formula A XOR B, which is true when exactly one of A or B is true. This can be written in CNF as $(A \vee B) \wedge (\neg A \vee \neg B)$, in DNF as $(A \wedge \neg B) \vee (\neg A \wedge B)$, in AND-inverter graph form as $\neg(\neg A \wedge \neg B) \wedge \neg(A \wedge B)$, and in NOR-inverted graph form as $((A \text{ NOR } A) \text{ NOR } (B \text{ NOR } B)) \text{ NOR } (A \text{ NOR } B)$.

This approach was taken by the OptCircuit method [28], which uses a decomposition procedure to bracket a solution by generating a converging sequence of upper and lower bounds. The same modeling framework was taken by SYNBADm [115–117], which applied metaheuristics/stochastic optimization methods (enhanced Scatter Search, Ant Colony Optimization, Variable Neighborhood search, Mixed Integer Tabu Search) to multi-objective problems (in the examples in the paper, one of the objective functions was a measure of circuit performance, and the second was a measure of protein production).

A coarse-grained approach can be used to categorise promoters and ribosomes as low/medium/high strength [165]; this allows optimization of steady-state behaviour to be formulated as an integer linear program and solved with a solver such as IBM ILOG CPLEX.

Genetdes [127] designs circuits using simulated annealing [81]. The proposed moves at each iteration are of one of several kinds: either a kinetic parameter is altered, or a network interaction is added or removed, a gene is removed, or a new gene is added (under a constitutive promoter). This work was continued under the new name AutoBioCAD [128], which uses parameters from a parts library, and can output a genetic sequence for the final design in GenBank format (as well as a model in SBML format that can be used for simulations).

1.5 SBOL VISUAL

Key to communication in synthetic biology is the schematic diagram showing the structure of engineered nucleic acid sequences, and the interactions that enable them to act as functional circuits. These diagrams are standardised by the SBOL Visual community standard [123, 148], which tries to produce a 'coherent language for the structure and function of genetic circuits', largely by standardising emerging practices. This effort is important, as without a shared understanding of the semantics of diagrams, their meaning is unclear and there is the potential for confusion.

Since the earliest papers in synthetic biology [39, 50], researchers have attempted to produce diagrams showing the structure and in-

tended function of their designs, but these have not always been clear or well thought through. The SBOL Visual project effort has thus been able to proceed largely by identifying and standardising emerging best practices and conventions, and produce a result that helps to guide researchers towards producing better diagrams..

In contrast, some of the objects considered in this thesis have not traditionally been represented using structured diagrams, and so a major effort has been to design new graphical representations, rather than standardising and systematizing existing ones.

1.6 SOFTWARE IMPLEMENTATION CHOICES

Many of the prototype systems described in this thesis are implemented as web-applications, rather than as desktop applications. One reason for this design decision is that web applications are inherently cross-platform; it would be more difficult to implement these applications as native desktop applications that work on multiple operating systems, especially as they involve relatively complex user interactions. Developing a web application also allows us to take advantage of the existing JavaScript libraries for data visualization and user interaction. Where it is necessary to persist state in a database, integrate with separate commandline applications, or perform computations that would be inconvenient to carry out in JavaScript, we use a Python backend.

A web application can also be centrally hosted, rather than needing to be installed on every user's machine. This seems particularly appropriate for some applications: if a liquid handling automation facility is centralised, the corresponding software may as well be too.

However, the web applications can also be run on user's desktops. The provision of Dockerfiles allows end users to use a single command to create an image containing the application and all necessary dependencies, without affecting the configuration of the host machine. A Docker image could also be pre-built and distributed as a single file with no dependencies other than Docker itself.

1.7 THESIS STRUCTURE AND CONTRIBUTIONS

This introductory chapter has provided background information about synthetic biology and mathematical modelling.

Chapter 2 introduces `TimeRails`, a new graphical representation for specifications expressed in Signal Temporal Logic. It also describes an interactive editor that can be used to create, edit, and interact with such formulae. In contrast to previous work, the notation proposed in this chapter can represent a wider range of Signal Temporal Logic specification in a consistent way.

Chapters 3 and 4 describe tools that use `TimeRails` as an interface for design tools in synthetic biology, using two very different approaches.

Chapter 3 describes new a tool that allows a user to automatically synthesize a CRN that meets a specification expressed using `TimeRails`, by formulating an SMT-ODE problem that can be solved by existing solvers, using the method originally described in [23]. It then extends this method to handle input species whose concentrations have a given profile over time, enabling the synthesis of CRNs with particular responses to a given input function. The user can either enter a fixed topology, and synthesize only the parameter values, or enter a parameteric *sketch* of the CRN topology and synthesize the topology and parameter values simultaneously.

Chapter 4 describes visual analytics tools to visualise the results of parameter estimation performed using ABC-SMC, enabling the investigation of how parameter values affect the behaviour of a system. The chapter demonstrates that the ABC-SMC method can be combined with the robustness degree of a specification expressed in Signal Temporal Logic to estimate the posterior distribution of parameters, *given that a specification is met*. The tool allows a user to visually draw a specification using `TimeRails`, and then visually explore the sets of parameters for which this specification is satisfied.

Chapter 5 describes `sbml-diff`, a tool for visually representing and comparing the structure of SBML files describing mathematical models of biochemical reaction systems. Unlike some existing tools, it is

able to represent `rules` and `events`, not just `reactions`, and produce diagrams showing a range of levels of detail. This is integrated into the Bayesian design tool described in Chapter 4.

Chapter 6 describes a new visual representation for experimental protocols, and a software tool implementing it that can be used to control laboratory robots. The usefulness of the tool is evaluated by directly comparing the representation of the same protocol in List of Liquids and an earlier tool (Section 6.2.1), and showing that this the representation is able to capture a number of protocols that are in current use.

While the work described in this thesis was motivated by problems arising in synthetic biology, many of these problems also arise in other domains, to which the tools developed also apply. For example, temporal logic specifications are used in other engineering fields, SBML models are used in systems biology as well as synthetic biology, and liquid handling robots are used across the chemical and life sciences.

Part I

DESIGN

2

FORMAL SPECIFICATIONS

2.1 MOTIVATION & CONTRIBUTION

When designing a system, an engineer's intention is to produce a design that will perform as intended, as judged by compliance with a specification.

As well as reducing the ambiguity inherent in natural language, expressing such specifications in a precise mathematical form has the advantage of potentially allowing two things:

- confirming that a system (or, more precisely, a a mathematical model of the system) will perform as intended: either by *testing* to check that a property holds in some specific circumstances that are simulated; or by performing *system verification* to construct a mathematical proof that some property always holds (or find a counter-example for which the property is violated)
- automatically designing (*synthesizing*) a system that meets the specification

The traditional applications of formal methods have been to computer software and electronic hardware in domains where failures are likely to be expensive or endanger human life. For example, verifying the correctness of a new CPU design is justified by the potential cost of a recall (the Pentium FDIV bug is said to have cost Intel $465 million in 1994, and the Sandy Bridge bug was estimated to cost Intel $700 million in 2011). In contrast, the application of formal methods to synthetic biology has so far been relatively limited. One reason for this is the difficulty of constructing a reliably predictive model of a synthetic biological system. In verifying a software system, we must assume that a program will be interpreted using the semantics defined for the language in which it is written; in verifying a syn-

thetic biological circuit, we must assume that it functions as described by a particular mathematical model. Whilst translating from the real world to the abstracted world in which the verification is performed requires assumptions to be made in both cases, the validity of the assumptions is less clear in the case of synthetic biology. However, this uncertainty is expected to decrease as further experiments increase our knowledge and increase our predictive ability. We therefore press on and develop methods as best we can, in the hope that improved models developed in the future can be inserted in a modular way.

Whilst formal methods are in principle widely applicable, their use requires specifications to be expressed in a precise format, and the difficulty of reading and writing specifications is one limitation to their use. Written conventionally, these specifications are typically an intimidating string of symbols (including $\wedge$, $\vee$, $\Box$, $\diamond$, $\neg$). The problem of writing specifications could in principle be avoided by instead having a user annotate trajectories as satisfying or violating a desired property, and attempting to learn the corresponding specification from these examples [86]. However, this does not help the user to verify that the learned specification actually captured what was intended, rather than some irrelevant distinction between the specific training examples provided for the two classes of trajectory. Additionally, communication of the specification to someone else requires a way to directly represent the specification.

An alternative approach, which we adopt, is to represent specifications visually. Whilst there has been some previous work on visualising specifications expressed in temporal logics, most considered systems that are characterized by discrete states rather than continuous signals. An exception is the VisSpec system [68], which represents each temporal operator, or combination of temporal operators, as a single block labelled with the corresponding property (e.g., 'always' or 'at least once'). Users pick a block from a list of 'templates', then replace placeholders with numerical values. The main innovation of the work presented here is to map each operator of the symbolic logic expression directly onto a *graphical* element that can be directly

manipulated. We are also able to represent a different set of specifications, including some that VisSpec cannot (e.g., $\Box \diamond \Box \phi$).

This chapter begins by reviewing ways in which specifications can be formulated (Section 2.2). It then introduces the `TimeRails` approach for writing specification (Section 2.3), and describes its implementation. Section 2.4 describes some applications, and Section 2.5 concludes.

Whilst the original motivation for creating `TimeRails` was to express specifications for systems to be used in design, it can also be applied to other problems such as searching through a collection of timeseries (Section 2.4.1).

2.2 INTRODUCTION

2.2.1 *Ways of expressing specifications*

Natural language

The easiest way to describe desired behaviour is as a description written in natural language (for example, requiring that 'the concentration of protein X should rise above a threshold within 30 minutes of the concentration of input A rising above a threshold, and not decrease until after input B is supplied'). However, whilst such descriptions may seem to be easily understood they have the potential for ambiguity, which increases as they become more complicated, and they cannot be directly processed by automated tools. Ambiguity could be removed by using a sufficiently constrained form of natural language, but (unlike a diagram) this would still not allow the direct comparison of a specification with a specific system trajectory obtained from a simulation or experiment.

Distance to a reference

We could allow the user to specify a *reference curve*, and require the system's output to be sufficiently close to it, in the sense that the distance measured using an l_p metric[1] is less than some threshold.

However, this has the disadvantage of being sensitive to potentially irrelevant transformations: for example, a user may want a system's output to be a Gaussian curve, and not care about the precise time at which the peak occurs, but the distance between two identical Gaussian curves that are offset in time will be large. A related difficulty for synthesis is that the precise curve that was drawn may not be realizable, despite curves sharing the qualitative features of interest being realizable.

2.2.1.1 *Ad hoc specifications*

It is possible to use a general-purpose programming language to write a function that takes a simulation trace as an input, performs some computation, and returns a result indicating whether the corresponding specification has been met (potentially together with an indication of *how close* the specification was to being met).

This approach has been used for several studies in systems biology, which have randomly sampled combinations of network topologies and parameters, and checked which give rise to particular behaviours: some of these studies are listed in Table 2.1.

However, this approach has two main disadvantages: it requires the user to be comfortable expressing their specification in a programming language, and as the specification is expressed as a computer program rather than as a logical formula many existing tools cannot be applied to it.

1 For real-valued p, the l_p norm of a vector is defined as $\|x\|_p = \sqrt[p]{\sum_i |x_i|^p}$, with the special cases that $\|x\|_\infty = \max_i \|x_i\|$ and the l_0 "norm" $|x|_0$ is defined as the number of non-zero elements in x.
The l_p-metric is then a norm metric defined as $d_{l_p}(x, y) = \|x - y\|_p$. The Euclidian metric is the l_2-metric.

Property	Specification
Robust Adaptation [99]	The input is a step function from I_1 to I_2; the output has steady-state values O_2 and O_1, and a transient with peak value O_{peak}. The specification requires both precision and sensitivity to be above a threshold. $\text{Precision} = 1/\text{steady-state error} = \dfrac{\lvert I_2 - I_1\rvert / I_1}{\lvert O_2 - O_1\rvert / O_1}$ $\text{Sensitivity} = \dfrac{\lvert O_{\text{peak}} - O_1\rvert / O_1}{\lvert I_2 - I_1\rvert / I_1}$
Stripe formation [134]	"Here, we define a stripe as a region of at most 36 cells of high expression, preceded and followed by a region of low expression, where 'low' refers to values lower than 30% of MaxConcentration, and 'high' refers to values higher than 30% of MaxConcentration."
Oscillation [160]	$d_1 = (n_{\max} - n_O)^2$ $d_2 = (n_{\min} - n_O)^2$ $d_3 = (\langle \Delta t_{\max} \rangle - n_t / n_O)^2$ $d_4 = (\langle \Delta t_{\min} \rangle - n_t / n_O)^2$ $d_5 = 1/(\langle x_{\max,i} \rangle - \langle x_{\min,i} \rangle)$ After smoothing the signal, $n_{\min}$, $n_{\max}$ are the number of minima and maxima; $\langle x_{\min,i} \rangle$ and $\langle x_{\max,i} \rangle$ are the average values of the minima and maxima; $\langle \Delta t_{\min} \rangle$ and $\langle \Delta t_{\max} \rangle$ are the mean number of time points between successive minima and maxima, n_t is the total number of time points. n_O is the number of desired oscillations. The objective is satisfied if each distance function is below a threshold.
Switch-like responses (bistability) [61]	At 'a particular stimulus concentration', the outputs must differ by at least 0.1 (and at least a factor of 5), depending on whether the stimulus had previously been high or low. $\text{maximum local response} = \max\left(\dfrac{d \ln \text{Output}}{d \ln S}\right)$
Switch-like responses (ultrasensitivity) [61]	$n_H = \dfrac{\log 81}{\log \frac{S_{90\%}}{S_{10\%}}}$ Where $S_{10\%}$ and $S_{90\%}$ are the input values at which the output reaches 10% or 90% of its maximum output value.

Table 2.1: Some *ad hoc* specifications found in the literature. Each was implemented precisely in software, but they are typically described more vaguely in the corresponding paper. No attempt was made to fit each specification into any larger framework. Note that these specifications require only a particular qualitative behavior, not a behavior that quantitatively matches some specific requirement.

2.2.1.2 *Temporal logics*

Temporal logics are a family of logics that include a notion of time, and allow the truth values of a logic formula to change as time evolves.

Broadly, these can be categorised according to whether they treat time as linear or branching, whether time is metric or simply ordinal, and whether they can handle signals that are real-valued rather than Boolean. The relationship between many of these is shown in Figure 2.1.

A branching temporal logic considers a *tree of states*: from each state, there may be many possible future states, and possible trajectories of the system correspond to different paths through this tree. A specification in a branching temporal logic applies quantifiers over paths, which can denote that a property holds *for all paths* from a particular state, or that *there exists a path* from a particular state for which a property holds. In contrast, a linear temporal logic considers only a single possible successor state for each current state. In some temporal logics, time is ordinal—the system simply progresses through states in order—whereas in others time is metric, making it possible to refer to time differences and durations (for example, to require that a response should occur within a certain time of an input being provided). Some specifications refer to states that are simply Boolean, whereas others allow variables that are real-valued: this is convenient for referring to physical values and signals, and allows a more meaningful interpretation of how close a trajectory is to meeting a specification (see Section 4.2.1).

2.2.1.3 *Signal temporal logic*

Signal Temporal Logic (STL) is a linear temporal logic, in which time is metric, and variables are real-valued functions of time ('signals'). It allows the expression of statements about time by introducing three temporal operators: Until ($U_{[a,b]}$), Globally ($\Box_{[a,b]}$) and Finally ($\Diamond_{[a,b]}$). The subscript indicates the time interval to which the operator applies. The **until** operator applies to two predicates: it is true if there is some time within the interval such that the first predicate is true

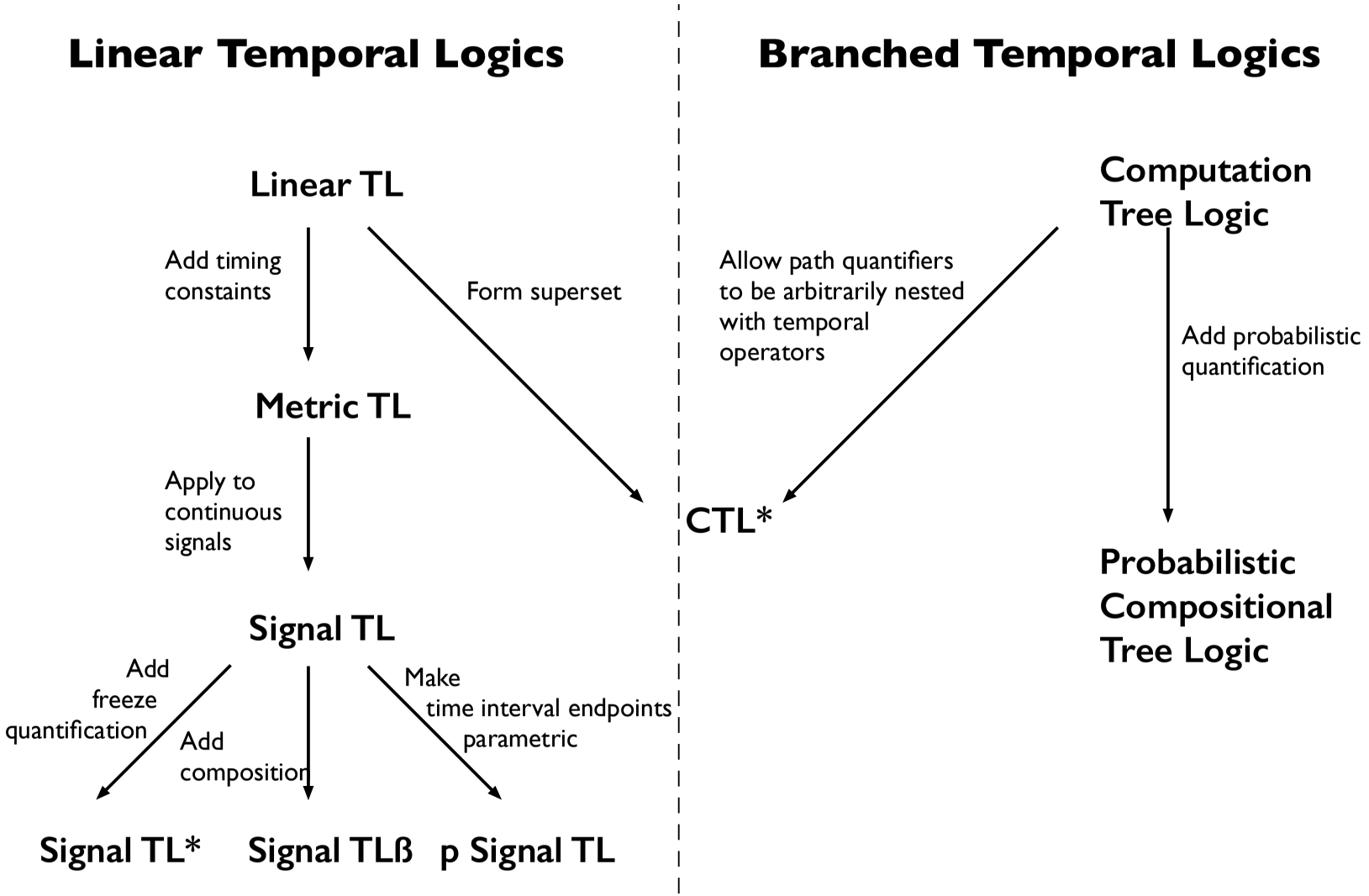


Figure 2.1: This diagram shows the relationship between some of the many different temporal logics that are in use. It is also sometimes useful to work with restricted *fragments of* these. Temporal logics divide broadly into branched and linear temporal logics: both are capable of expressing things that the other cannot. CTL* combines the features of CTL and LTL into a logic that is more expressive than either [5 📖, Sec 6.8].

from the start of the interval until that time, and the second predicate is true from that time until the end of the interval. The globally and finally operators apply to a single predicate. The **globally** operator is true if the predicate is true for all times in the interval. The **finally** operator is true if there is some time within the interval at which the predicate is true.

To make this description precise, we will describe the syntax and semantics of STL. It has the grammar:

$$\varphi ::= \mu | \neg\varphi | \varphi_1 \vee \varphi_2 | \varphi_1 \wedge \varphi_2 | \varphi_1 U_{[a,b]} \varphi_2$$

where $[a, b]$ is a time interval, and μ is a numeric predicate in the form of an inequality.

Using the notation $s[t] \vDash \varphi$ to mean that the signal s satisfies the STL formula φ at time t, the qualitative semantics of STL can be written as:

$$s[t] \vDash \neg\varphi \Leftrightarrow s[t] \nvDash \varphi \tag{2.1}$$

$$s[t] \vDash \varphi_1 \vee \varphi_2 \Leftrightarrow s[t] \vDash \varphi_1 \text{ or } s[t] \vDash \varphi_2 \tag{2.2}$$

$$s[t] \vDash \varphi_1 \wedge \varphi_2 \Leftrightarrow s[t] \vDash \varphi_1 \text{ and } s[t] \vDash \varphi_2 \tag{2.3}$$

$$s[t] \vDash \varphi_1 U_{[a,b]} \varphi_2 \Leftrightarrow \exists t' \in [a+t, b+t] : s[t'] \vDash \varphi_2, s[t''] \vDash \phi_1 \forall t'' \in [t+a, t')] \tag{2.4}$$

$$\diamond_{[a,b]} \varphi \equiv \text{ true } U_{[a,b]} \varphi \tag{2.5}$$

$$\square_{[a,b]} \varphi \equiv \neg \diamond_{[a,b]} (\neg\varphi) \tag{2.6}$$

$$\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2 \tag{2.7}$$

Equation 2.1 states that satisfying a specification is equivalent to not satisfying its negation. Equations 2.2 and 2.3 state that conjunction and disjunction behave in the obvious way: satisfying (ϕ_1 or ϕ_2) is equivalent to satisfying ϕ_1 or satisfying ϕ_2; satisfying (ϕ_1 and ϕ_2) is equivalent to satisfying ϕ_1 and satisfying ϕ_2. Equation 2.4 defines the temporal operator *Until* (U): $\varphi_1 U_{[a,b]} \varphi_2$ is satisfied if there is some time t' between a and b such that φ_1 holds from time a until time t', and φ_2 holds at time t'. Equations 2.5-2.7 provide 'syntactic sugar', defining useful operators whose semantics can be easily defined in

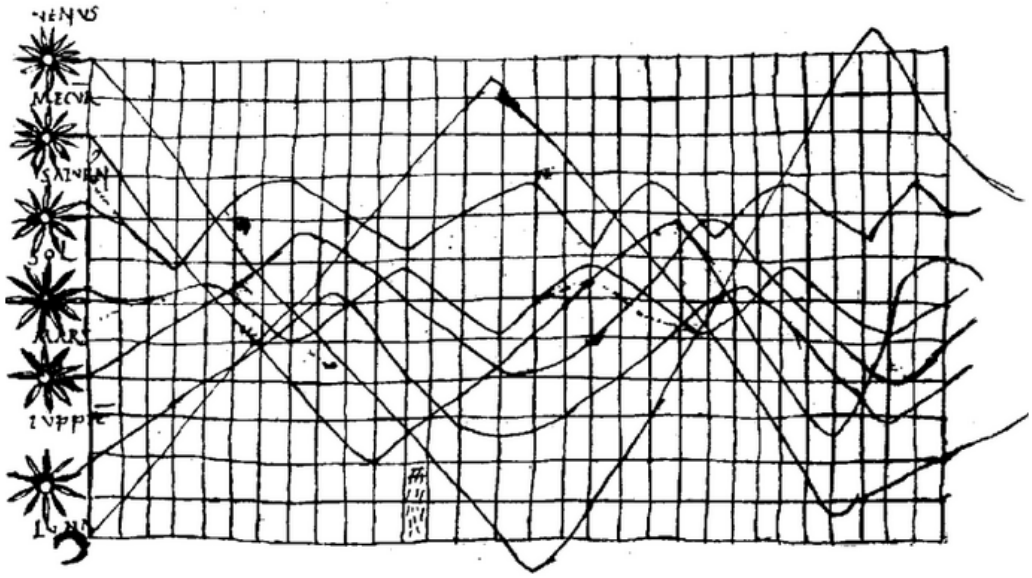


Figure 2.2: The earliest known line graph, dating from the 10th Century [47].

terms of previously defined operators. Equations 2.5 and 2.6 define two additional temporal operators: $\Diamond_{[a,b]}\varphi$ ("eventually" or "finally") ensures that φ will hold at some time between a and b time units in the future; $\Box_{[a,b]}\varphi$ ("globally") ensures that φ will hold at all times between a and b time units in the future. Equation 2.7 defines implication in terms of conjunction and negation, saying that $\varphi_1 \rightarrow \varphi_2$ is true unless φ_1 is true and φ_2 is false. Some authors also choose to remove disjunction from the core grammar and semantics, and define it as $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$.

STL can also be interpreted using a *quantitative* semantics, which gives a numerical indication of how robustly the specification is met (or how close it was to having been met); this is discussed further in Chapter 4, where it is used to search for system parameters that cause a specification to be satisfied.

2.2.1.4 *Alternative viewpoint*

It is straightforward to visualize a single function $\mathbb{R} \rightarrow \mathbb{R}$ as a line-graph. Such graphs are very familiar, and quite old, with a known example dating from the 10th Century [47] (Figure 2.2).

This representation can be easily extended to a vector-valued function $\mathbb{R} \rightarrow \mathbb{R}^n$ by representing each of the n dimensions of the range

by a separate line, either superimposed on a single plot or on separate sub-plots that are vertically aligned to have a common x-axis. A set that is explicitly defined as consisting of a finite number of such functions can be represented by drawing each as a line that is visually distinguished from the others (e.g., by line colour or stroke style): this is conceptually straight-forward, though practical problems due to occlusion will be encountered if the cardinality of the set is large.

A more difficult problem is to graphically represent a set whose membership is defined implicitly as the (potentially infinitely many) functions satisfying a condition: in this case we must represent the *condition* that defines set membership, rather than the members themselves. This problem is the subject of this chapter. The central problem is to develop an abstraction that is clear and easy to understand, is internally consistent, and is able to generalise to the wide range of sets that may be encountered in practice.

As well as representing a system specification for use in testing, formal verification, or synthesis, there are several other reasons why we may want to represent such a set:

- we may be given a set of functions, and want to *search* or *filter* these
- we may want to show the results of some abstraction of a set of functions.

For a small set of functions, the second of these goals may be met by simply superimposing plots, but this quickly becomes hopeless as the number of superimposed lines grows.

Temporal logics have the advantage of already having been successfully applied to engineering problems, and provide a consistent overarching framework that encompasses many different specifications, in the form of a language with a precisely defined syntax and semantics that is supported by existing tools. They are therefore a good foundation on which to build a visual representation for specifications: the bulk of this chapter takes STL as a symbolic language as a starting point, and attempts to design a graphical equivalent.

2.2.1.5 *Visual representation of specifications*

Diagrammatic representations of logical formulae have a long history: the progression from Euler diagrams (c. 1786), via Venn diagrams (c. 1881), to Pierce diagrams (c. 1933) culminates in a diagrammatic system that is equivalent to monadic predicate logic [141 📖]. In Boolean logic, the Marquand diagram (c. 1881) was rediscovered as the Veitch chart (c. 1952), and refined as the Karnaugh map (c. 1953), resulting in a practical diagrammatic tool for simplifying Boolean functions.

Diagrammatic representations for *temporal* logics have a shorter history. The most complete system is ViSpec[2], described in [33, 69, 70] and in Bardh Hoxha's PhD thesis [67 🎓]. This provides a menu-based interface in which the user first selects the form of specification that they wish to add from a list of choices ('None', 'Always', 'At Least Once','Eventually Always', 'Repeatedly Often and Finally'). Having selected such a template, they are shown a form that expresses the specification as a sentence containing empty input boxes into which they can enter values for times and threshold values (Figure 2.3). It is not possible to directly manipulate the diagram representing the specification (e.g., by dragging to adjust thresholds).

The ViSpec system is really a form-based editor with a schematic diagrammatic viewer, rather than being a diagrammatic editor *per se*. Multiple templates can be added, and are represented by a separated diagram aligned on a common time-axis; it is not possible to display two templates that constrain the same variable with a shared y-axis. Templates can be grouped in three ways: by *implication* (this is the default), by *conjunction* (depicted by a black box), or by nesting temporal operators (indicated by a green wedge joining the templates—see bottom panel of Figure 2.5).

The publications describing ViSpec claim that this enables it to represent specifications with following grammar:

2 The name 'TemporalGUI' also seems to have been used, and is the name of the example application made available at `https://sites.google.com/a/asu.edu/s-taliro/vispec`.

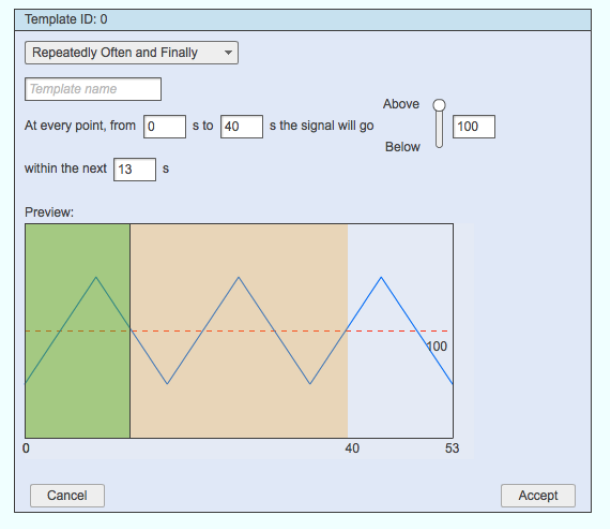


Figure 2.3: The editor used within ViSpec to create a property by selecting a template and populating the corresponding fields.

$$S ::= \neg T | T$$

$$T ::= A | B | C$$

$$A ::= P | (P \wedge A) | (P \implies A)$$

$$B ::= \Box_{\mathcal{I}} D | \Diamond_{\mathcal{I}} D$$

$$C ::= \Box_{\mathcal{I}} \Diamond_{\mathcal{I}} D | \Diamond_{\mathcal{I}} \Box_{\mathcal{I}} D$$

$$D ::= (p \implies A) | (p \wedge A) | (p \implies B) | (p \wedge B)$$

$$P ::= p | \Box_{\mathcal{I}} p | \Diamond_{\mathcal{I}} p$$

p is an atomic proposition

However, they also present an additional example $\Diamond_{[0,30]} \Box_{[0,10]} (speed < 100)$, which is of the form $\Diamond_{\mathcal{I}} \Box_{\mathcal{I}} p$, and does not conform to this grammar: the nesting of temporal operators can be achieved only through the application of definitions B or C, which apply temporal operators to D, but D cannot be an atomic proposition.

This grammar is unable to represent disjunction, either directly (it does not include a $\vee$ symbol) or indirectly ($\neg$ can only be applied to the complete specification, so the expression $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ cannot be written).

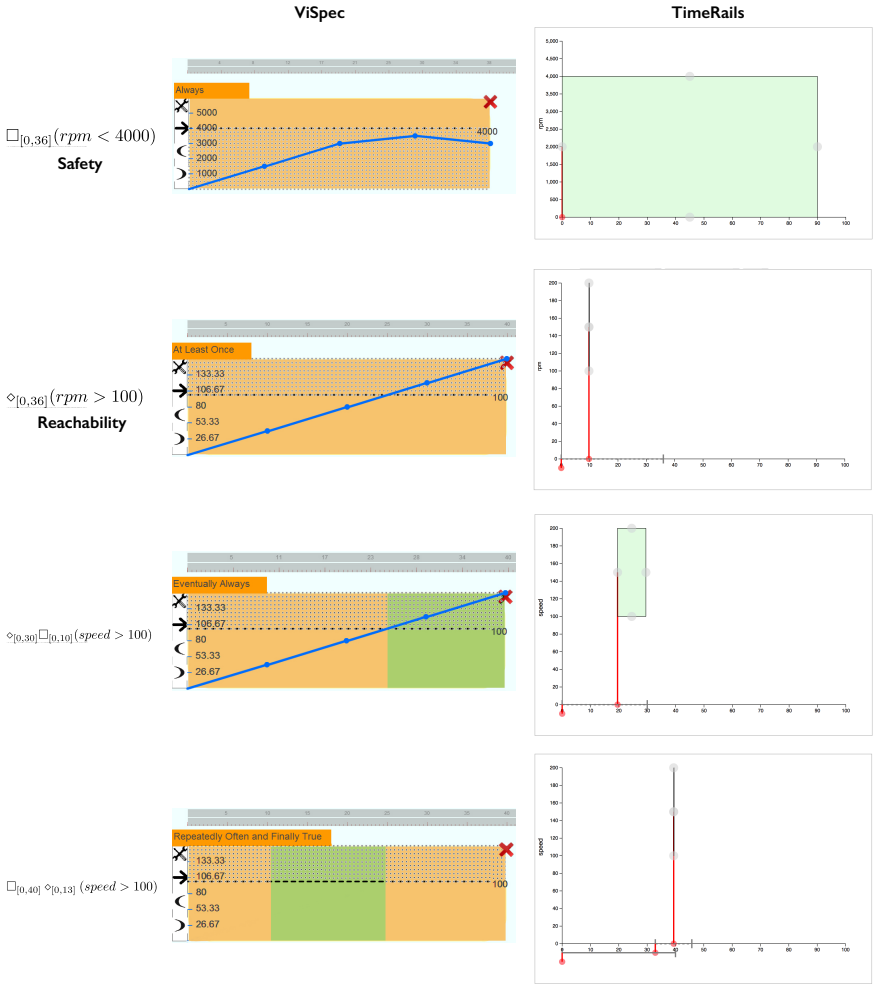


Figure 2.4: Example properties and their representations in ViSpec (left) and `TimeRails` (right). The screenshots of ViSpec appear as Figures 4-10 in [70]. Note that in ViSpec the diagrammatic representations of the *safety* and *reachability* properties are identical, and they are distinguished only by labels, whereas in TimeRails they are distinct.
These specifications require that: rpm remain below 4000 for all times between 0 and 36; rpm increases above 100 at some time between 0 and 36; that at some time between time 0 and 30 speed will increase above 100 and remain above 100 for 10 time units; and that at all times between 0 and 40, speed will increase above 100 within 13 time units of that time.

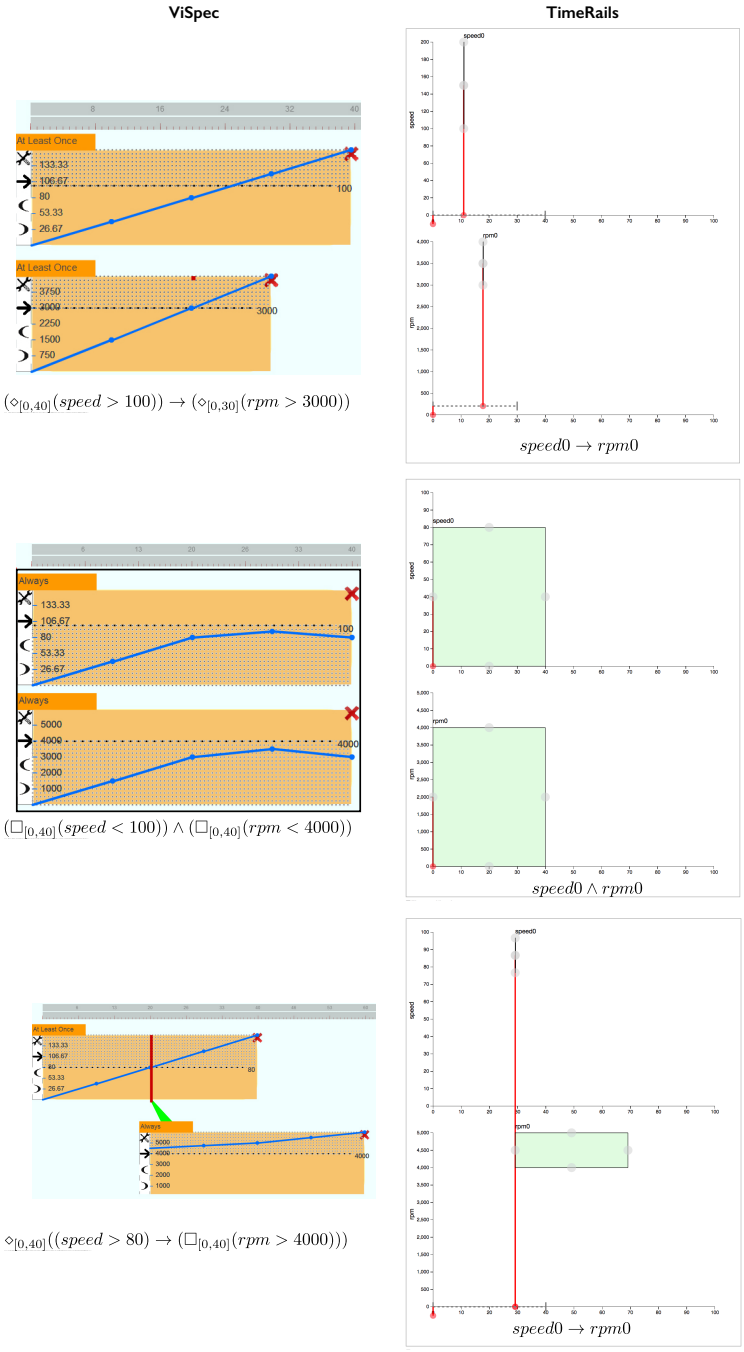


Figure 2.5: Example properties and their representations in ViSpec (left) and `TimeRails` (right). The screenshots of ViSpec appear as Figures 4-10 in [70].
These specifications require that: if speed increases above 100 in the first 40 time units, then the rpm must be above 3000 at some point in the first 30 time units; the speed must be less than 100 for the first 40 time units, and the rpm must be less than 4000 for the first 40 time units; and if the speed increases above 80 at some time in the first 40 time units, then the rpm must be greater than 4000 at some time within 50 time units of this time.

Note that $\Box_{[a,b]}\Box_{[c,d]}(\cdot)$ and $\Diamond_{[a,b]}\Diamond_{[c,d]}(\cdot)$ are degenerate and can be simplified to $\Box_{[a+c,b+d]}(\cdot)$ and $\Diamond_{[a+c,b+d]}(\cdot)$ respectively. Note also that specifications including logical implication (Reactive Request-Response and Non-strict sequencing) implicitly contain quantifiers over traces so cannot be meaningfully applied to single traces or trajectories, but only to systems: they assert that *for all* trajectories meeting one condition, and second condition is also met.

2.3 TIMERAILS CONCEPT

We introduce a diagrammatic representation of specifications expressed in STL. In contrast to previous systems, it is able to represent the complete grammar of STL, rather than just a fragment of it.

2.3.1 *Visual representation*

The basic approach we adopt is to draw the axes that would be used to plot trajectories of the system, and to overlay a representation of the constraints imposed by the specification.

Our goal is to produce a diagrammatic representation of a specification, on top of which a specific trajectory can be superimposed. The first design choice is what coordinate system to use for plotting trajectories. As we are using a temporal logic with a metric notion of time, it is necessary to explicitly represent time as a dimension (rather than simply plotting a projection of the trajectory onto the just the axes corresponding to the state variables, which would give a phase-space plot or connected scatterplot). We could use an orthogonal coordinate system, but we run out of spatial dimensions with which to represent more than 2 state variables alongside time (and 3 dimensional representations already encounter problems due to occlusion and depth perception). Points in a higher-dimensional space can be represented using either radial or parallel coordinate systems[3],

3 Typically a point in multidimensional space is drawn as a point at the location that is the intersection of all surfaces individually defined by a coordinate. Typically, the value of each coordinate defines a surface, and a single data-point is represented by a circular mark drawn at the intersection of all of these surfaces. Alternatively, it is

but it then becomes challenging to represent a time-series. A natural solution, which we adopt, is to use multiple orthogonal coordinate systems, by separating out each state variable into a distinct subplot. Multiple dimensions could be plotted on the same axes, but this has associated problems: over-plotting/occlusion/clutter due to drawing multiple lines in the same area, and the need for multiple y-axes with different scales if state variables have different physical units or have very different ranges.

A second question is how to represent a constraint on a variable's value between certain times. Again, this has a natural solution: a rectangle. `TimeRails` represent a constraint on the values that a variable may take using a rectangle; for the specification to be satisfied by the specification, the system must remain within each rectangle at all times between its start and end times. This provides an advantage over ViSpec, which represents threshold by a horizontal dashed line, requiring an additional textual label to indicate whether a variable is required to be *above* or *below* this threshold.

A more interesting question is how to represent the temporal operators globally and finally. ViSpec represents finally operators as green rectangles that should be imagined to *slide within* larger yellow rectangles[4], but cannot actually be made to slide by dragging.

We represent the globally and finally operators by rails along which rectangles or other rails can slide. The graphical components (rectangles and rails) compose in the same way as the corresponding symbols, and successive elements are linked. The finally operator indicates uncertainty in the time at which a proposition is true; this is represented diagrammatically by a dashed rail, and for a trajectory to satisfy the specification it must be possible to *slide along* the rail

possible to exploit a *point-line duality* and represent each data-point using a line: a circular mark can be drawn on each axis, at the position defined by the corresponding coordinate, and the marks representing the same data-point can be joined by a polyline. This means that the axes no longer need to be orthogonal, and so more than three of them can be drawn; they can be either radial (often called a 'radar chart' or 'star chart') or parallel (a parallel coordinates plot, as used in Section 4.3.2).

4 Actually they can slide out of the right end of the yellow rectangle, as long as the rectangles remain in contact. A property $\Box_{[a,b]} \diamond_{[0,c]} (x < 10)$ is represented by a green rectangle of width c, inside a yellow rectangle extending from $t = a$ to $t = b$ (see bottom panel of Figure 2.4); however, for the property to hold at $t = b$, the green rectangle would need to start where the yellow rectangle ends. This is potentially confusing.

into a position where the trajectory remains inside the rectangle. The globally operator represents a set of constraints; it indicates that a proposition is true for every time in a range. This is represented diagrammatically by a solid rail, and for a trajectory to satisfy the specification the trajectory must remain inside the rectangle for every position along the rail.

These features are illustrated in Figures 2.4 and 2.5, which shows how the same properties can be represented in ViSpec and `TimeRails`.

Two rails or rectangles may slide along the same parent rail, implying that their start times lie in the same interval. However, two elements may also be linked more directly than this (Figure 2.8). Two or more elements may start at the same time: this is represented visually by the elements sharing a vertical start line. In the editor, this relationship is created by right-clicking the circular mark on the start line of one rectangle, selecting "Link start times", then clicking on the circular mark on the start line of another. Alternatively, one rectangle may be specified as starting at the same time that another ends. This is indicated by a start time line that joins these two rectangles. In the interactive editor, dragging one rectangle will also drag linked rectangles.

The axes provides a good way to represent intervals or boxes in state-space and the *temporal relationships* between them, but they are not well suited to representing *logical relationships* such as conjunction, disjunction, and implication. `TimeRails` therefore allows each rectangle to be automatically labeled with a name, so that the user can write a separate logical formula (Figures 2.4 and 2.6). This logical formula is much simpler than the full Signal Temporal Logic specification, because the temporal operators and inequalities applied to state variables have been separated out and presented on the diagram instead.

2.3.2 *Interactive editing*

A user can add a new subplot (corresponding to a new variable) by pressing a button.

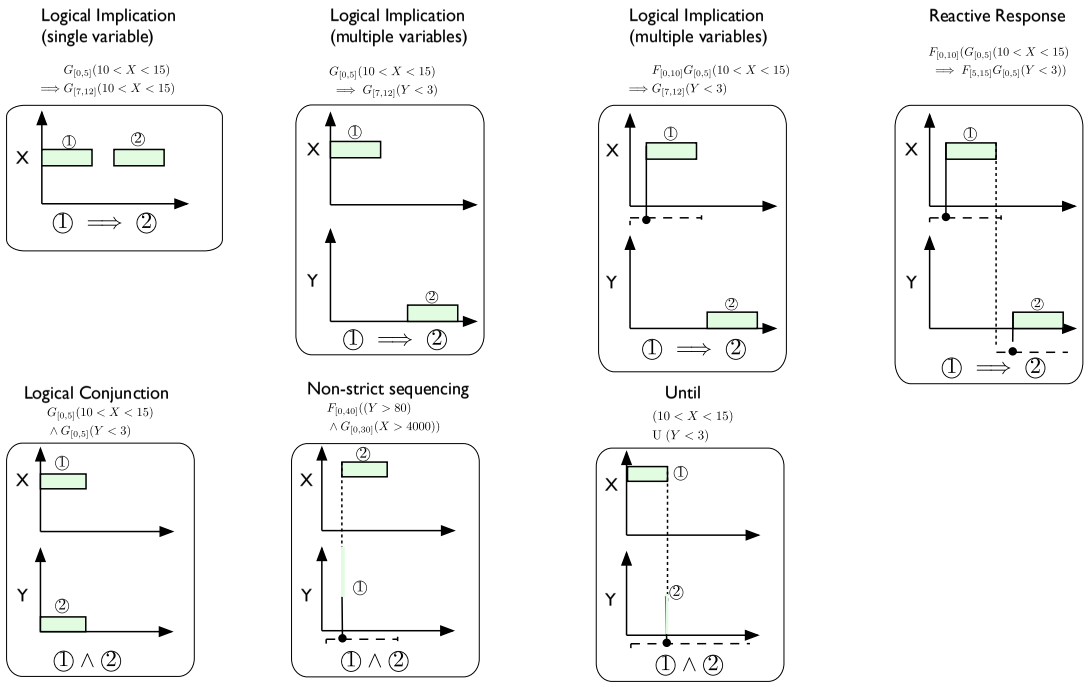


Figure 2.6: Sketches showing the representation of logical relationships in `TimeRails` notation. The diagram and logic formula enclosed in each of the rounded-rectangular boxes are together equivalent to the temporal logic formula written above it.

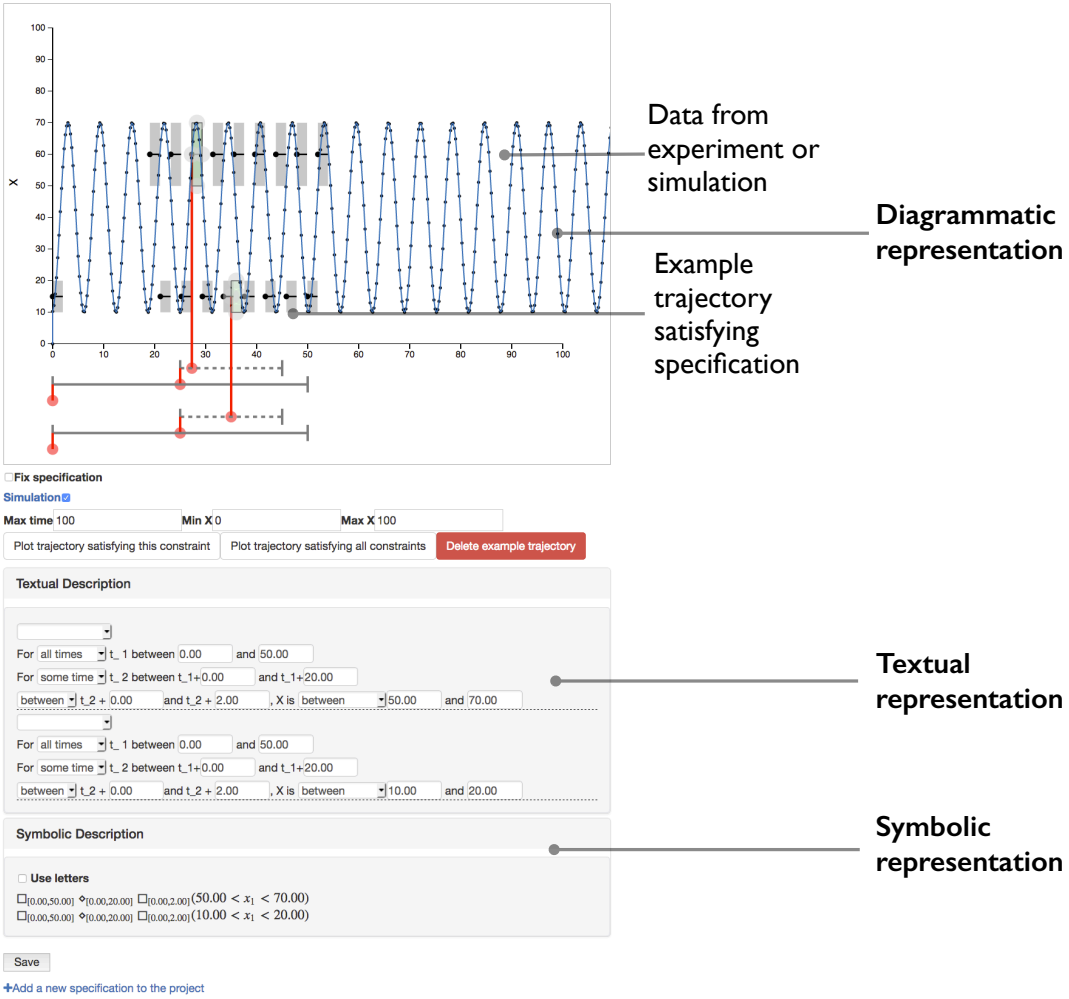
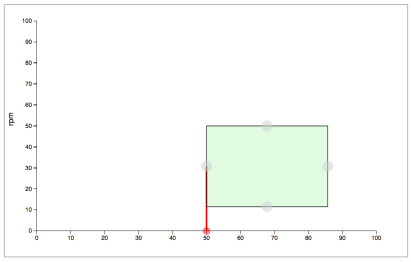
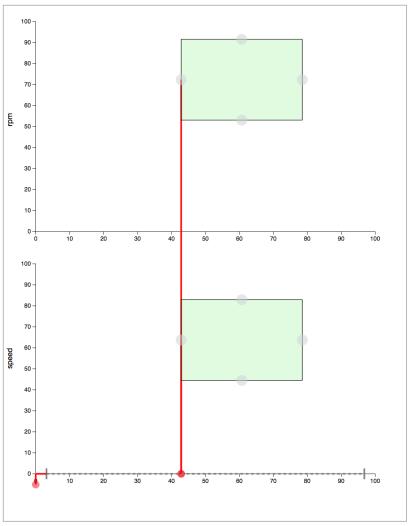


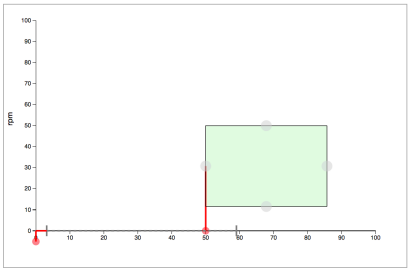
Figure 2.7: Screenshot of `TimeRails` interface. As well as the diagrammatic and symbolic representations, this includes a textual representation that describes the specification in words, with form elements that allow temporal operators and numerical values to be changed. This textual representation was present in an initial prototype but later removed, because it was challenging to scale it to more complex specifications.



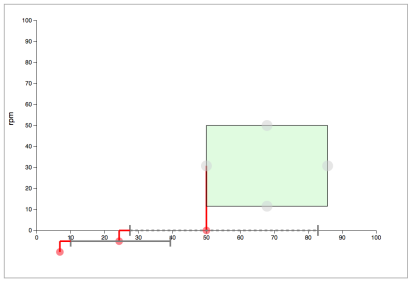
Fixed time



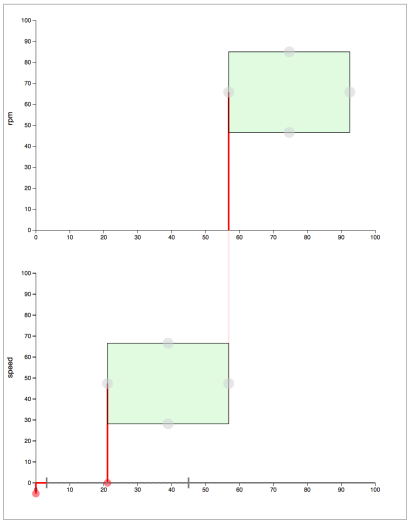
Linked start times



Single temporal operator



Nested temporal operators



Linked start and end times

Figure 2.8: Timing relationships expressible in `TimeRails`.

Right-clicking on a subplot opens a context-menu from which a rectangle can be added.

Right-clicking on the circle attached to each rail or rectangle opens a context menu that allows it to be attached to a newly created rail (if "Applies at *some* time in range" or "Applies at *all* times in range" is selected), or remove it from a rail to which it is currently attached ("Starts at fixed time").

Alternatively, the user can attach a rail to an existing rail, enabling two or more entities to be attached to the same parent rail (enabling the construction of branching terms like $\diamond_{\mathcal{I}}(\square_{\mathcal{I}}(\cdot) \wedge \square_{\mathcal{I}}(\cdot))$).

The user can adjust the specification by direct manipulation of the diagram: rectangles can be dragged to new positions, or resized by dragging one of the circular handles on their edges; rails can be adjusted by dragging their tick marks. Alternatively, right-clicking on a rail or rectangle and selecting `Edit Values` from the context menu that appears opens a modal dialog box containing a form that allows exact numerical values to be modified.

However, not all adjustments of the diagram correspond to changes in the specification that it represents: in particular, sliding a rectangle or rail along a rail representing a Finally term does not affect the underlying specification. A checkbox allows the user to 'lock' the specification, allowing them to perform such manipulations but preventing them from accidentally modifying the specification.

2.3.2.1 *Example trajectories*

To help the user check that they have specified what they intended to, `TimeRails` can automatically generate example trajectories that satisfy a specification. One way to do this would be to try to synthesize a *single trajectory*, or to synthesize a few individual trajectories: this is the approach taken by STLInspector [102, 130], which also synthesises trajectories that satisfy various mutated specifications based on the original specifications[5].

5 Specifically, STLInspector generates various mutated forms of the original specification by applying particular mutation rules. For each mutated specification, it generates an example signal that satisfies the mutant but not the original specification; if a human reports that this signal does not meet the intent of the specification,

We adopt a different approach, and try to represent a set of trajectories, obtained by fixing values of the times in the specification (but not precise values of the signal).

From the initial specification, which defines a set of rectangles that are able to slide along rails, we obtain a set of TimeBoxes at fixed locations, such that any curve that remains inside all of these will satisfy the original specification (curves that do not remain inside all of these may or may not satisfy the original specification); see Figure 2.7. A horizontal line segment is drawn through each box: any trajectory obtained by joining these with curves will satisfy the constraint. If the spaces between rectangles were filled by rectangles of unbounded height, the rectangles would represent an enclosure of a set of trajectories satisfying the specification.

The algorithm used to obtain the set of fixed TimeBoxes is described in Algorithm 1.

Globally terms are discretised and replaced by a conjunction of terms, each holding at one timepoint in the interval spanned by the original globally term; this results in an increase in the number of rectangles. After Globally terms have been expanded, the (multi-)tree of rails is then transversed upwards from the bottom-level rail towards the rectangles. For each Finally term, a new time variable is introduced to represent the time for which the term becomes true, and a constraint is added restricting the difference between this time and the start time of the parent rail to lie in the specified interval. If two rectangles have linked times, so that they both start at the same time or one starts when the second ends, then this is recorded by an equality constraint. Additional constraints are added to ensure that any pair of rectangles that do not overlap vertically also do not overlap horizontally; such an overlapping pair of TimeBoxes would not be satisfiable, as a signal could not simultaneously be inside both rectangles.

The resulting set of constraints forms an SMT problem in the theory of linear differences which is solved by the SMT solver Z3 [108] to ob-

it concludes that a particular error that could have been made in constructing the specification was not made.

tain a starting and ending time for each rectangle. Z3 was chosen because it provides a Python interface that can be used directly, without having to use a package such as `pySMT` as a wrapper layer. SMT problems, and the Davis–Putnam–Logemann–Loveland algorithm (DPLL) method for solving them, are discussed in Section 3.2.1.

This procedure for representing example trajectories using a set of rectangles does not currently handle specifications including quantifiers, implication, and conjunction/disjunction, but could be extended to include them.

A further extension would be to try to synthesize a several sets of boxes (each corresponding to a separate set of trajectories meeting the specification). This could be done by repeatedly solving the SMT problem, at each stage adding an additional constraint that the distance between the vector of rectangle start times being synthesized and the vectors of rectangle start times synthesized at previous iterations should be maximised. This maximisation could be achieved be adding a constraint that the distance should be larger than some threshold, and increasing the value of this threshold until the problem becomes unsatisfiable.

2.3.3 *TimeRails implementation*

As a proof-of-concept, we implemented `TimeRails` as a reusable JavaScript component using the `D3.js` library.

This can be embedded in other web pages and applications by simply calling the constructor, providing the `id` of the `<div>` HTML element in which the `TimeRails` editor should be rendered. This generates a `TimeRails` diagram object, which has methods for creating a subplot (for either an input or non-input variable), deleting or renaming a subplot, and loading and plotting data from a Comma Separated Values (CSV) file (either to compare experimental measurements or simulation results against a specification, or for use as a 'template' when constructing a specification). These allow control of the `TimeRails` editor from an application from which it is embedded (for example, in the `CRN Designer` application described in Chapter 3,

Algorithm 1 Synthesizing example trajectories.

```
1:  procedure GENERATE_EXAMPLES(specification)
2:      create SMT variable t_0
3:      ADD_CONSTRAINT(t_0 = 0)
4:
5:      for subplot in specification["subplots"] do
6:          for rail in subplot["rails"] do
7:              if rail has no parent then
8:                  create SMT variable t
9:                  ADD_CONSTRAINT(t = rail["start_time"])
10:                 PROCESS_RAIL(rail, t, subplot["variable_name"])
11:
12:         for rectangle in subplot["rectangles"] do
13:             if rectangle has no parent then
14:                 create time variable t
15:                 ADD_CONSTRAINT(t = rail["start_time"])
16:                 PROCESS_RAIL(rect, t0, subplot["variable_name"])
17:     CONSTRUCT_OVERLAPPING_CONSTRAINTS
18:     solve the SMT problem
19:
20: procedure PROCESS_RAIL(rail, parent_time, variable_name)
21:     if rail["kind"] == "globally" then
22:         for t in np.arange(start_time, end_time, dt) do
23:             create SMT variable new_time
24:             ADD_CONSTRAINT(new_time = (parent_time + t))
25:             PROCESS_RAIL_CHILDREN(rail, new_time, variable_name)
26:     else if rail["kind] = "Finally" then
27:         create SMT variable t
28:         ADD_CONSTRAINT((t − parent_time) >= start_time))
29:         ADD_CONSTRAINT((t − parent_time) <= end_time)
30:         PROCESS_RAIL_CHILDREN(rail, t, variable_name)
31:
32: procedure PROCESS_RAIL_CHILDREN(rail, parent_time, variable_name)
33:     for child in rail["children"] do
34:         if child["kind"] == "rectangle" then
35:             PROCESS_RECT(child, parent_time, variable_name)
36:         else
37:             PROCESS_RAIL(child, parent_time, variable_name)
38:
```

```
39: procedure PROCESS_RECT(rect, parent_time, variable_name)
40:     ADD_RECTANGLE(start_time + parent_time, end_time − start_time, min_val, max_val, variable_name)
41:
42: procedure CONSTRUCT_OVERLAPPING_CONSTRAINTS
43:     for r1 in rectangles do
44:         for r2 in rectangles do
45:             if rectangles apply to same variable then
46:                 if rectangles do not overlap vertically then
47:                     ADD_CONSTRAINT((r2.start > r1.end) or (r2.end < r1.start))
```

adding a new input chemical species triggers addition of a new input subplot to the `TimeRails` diagram).

`TimeRails` provides several options to customize its behaviour. Some of these affect appearance (`show_formula` determines whether the specification should be displayed as a logical formula (by being converted to LaTeX and rendered with MathJax), `generateExampleTrajectories` determines whether example trajectories satisfying the specification should be generated and drawn), or allow integration with a larger application (e.g., `saveURL` specifies the URL to which the diagram's state should be sent when the 'Save' button is pressed), but most allow the forms of specification that can be constructed to be constrained:

- `max_depth` sets a limit on how deeply temporal operators can be nested
- `allow_logic` determines whether the input box that allows specification of (non-temporal) logic operators should be created
- `allow_branching` determines whether more than one entity should be allowed to slide along the same parent rail
- `allow_modes` allows/forbids the drawing of modes or until relationships
- `allow_globlly` allows/forbids the addition of rectangles representing Globally terms

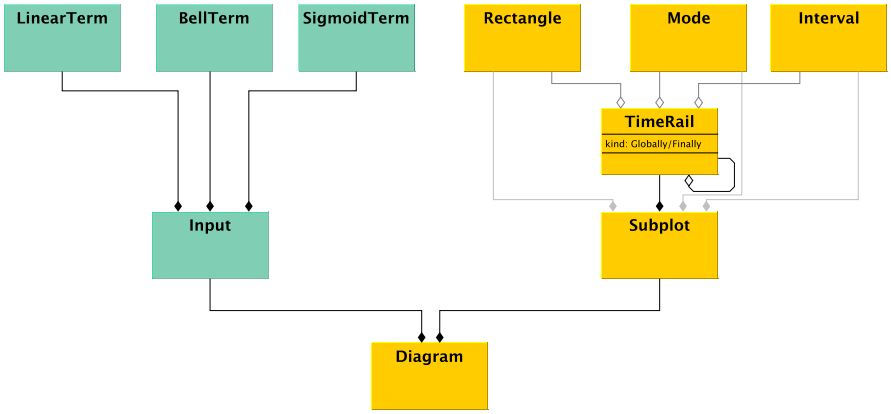


Figure 2.9: Relationship between classes in `TimeRails`. Objects at the top two levels can be directly manipulated using the mouse. Objects at the top two levels can be manipulated programamtically by calling functions on the `diagram` object.

- `allow_shared_times` allows/forbids the linking of the start time of two entities

This flexibility provided by these options is exploited by the example applications described in Section 2.4.

`TimeRails` consists of a number of classes (Figure 2.9). When the user manipulates a graphical element, the values stored in the corresponding object are updated; in order to save the specification, the diagram is serialised recursively (the digram object calls the `.toString()` method on each of its subplots and inputs, which in turn call the `.toString()` method on each rail, rectangle, etc.).

2.4 APPLICATION AREAS

2.4.1 *Search*

"Variable Time Timeboxes" were introduced in [79], and are discussed more fully in a later journal paper [65] and in Harry Hochhesier's PhD thesis [64 🎓]. They were implemented in an application called Timesearcher, which allows the user to construct queries of the following form[6]:

6 This grammar does not appear in any of the publication describing Timesearcher, and was constructed based on reading [64 🎓].

$$\begin{aligned}
\text{Query} &::= \text{Formula} \mid \textbf{not}\ \text{Formula} \\
\text{Formula} &::= \text{Constraint} \mid \text{AndClause} \mid \text{OrClause} \\
\text{AndClause} &::= (\ \text{Constraint} \mid \text{AndClause})\ \textbf{and}\ (\text{Constraint} \mid \text{AndClause}) \qquad (2.8) \\
\text{OrClause} &::= (\ \text{Constraint} \mid \text{OrClause})\ \textbf{or}\ (\text{Constraint} \mid \text{OrClause}) \qquad (2.9) \\
\text{Constraint} &::= (y_i \leq y(t) \leq y_2, \forall t_1 \leq t \leq t_2) \qquad (2.10) \\
&\quad \mid (\exists t_1 \in [t_3, t_4] \text{ s.t. } y_1 \leq y(t) \leq y_2,\ \forall t_1 \leq t \leq t_1 + d) \qquad (2.11) \\
&\quad \mid k_1 \leq \frac{y(t_2) - y(t_1)}{t_2 - t_1} \leq k_2 \qquad (2.12) \\
&\quad \qquad (2.13) \\
&\quad \mid k_1 \leq y'(t) \leq k, \forall t_1 \leq t \leq t_2 \qquad (2.14)
\end{aligned}$$

A query thus consists of a (possibly negated) conjunction (2.8) or disjunction (2.9) of constraints, each of which may be of one of 4 types: TimeBox (2.10), Variable Time TimeBox (2.11), endpoint-only angle constraint (2.12), or all-time angle constraint (2.14). Visually, TimeBoxes are represented as "rectangular query regions drawn directly on a two-dimensional display of temporal data" [79]. Variable Time Timeboxes are TimeBoxes that are free to slide horizontally within an enclosing box (they are similar to how Finally Globally terms are represented in ViSpec). Angular queries are represented by a line with the desired slope, and a vertical line attached to the endpoint indicating the permitted range of angles.

`TimeRails` could be used to provide a more general query interface than TimeSearcher; search results could be displayed separately, or superimposed on top of the `TimeRails` query, with a visual channel such as stroke style (continuous or dashed)/opacity/color used to distinguish between time series that do and do not meet the search criterion. Whilst the inclusion of angle constraints in TimeSearcher may appear to provide additional flexibility, the same constraints can

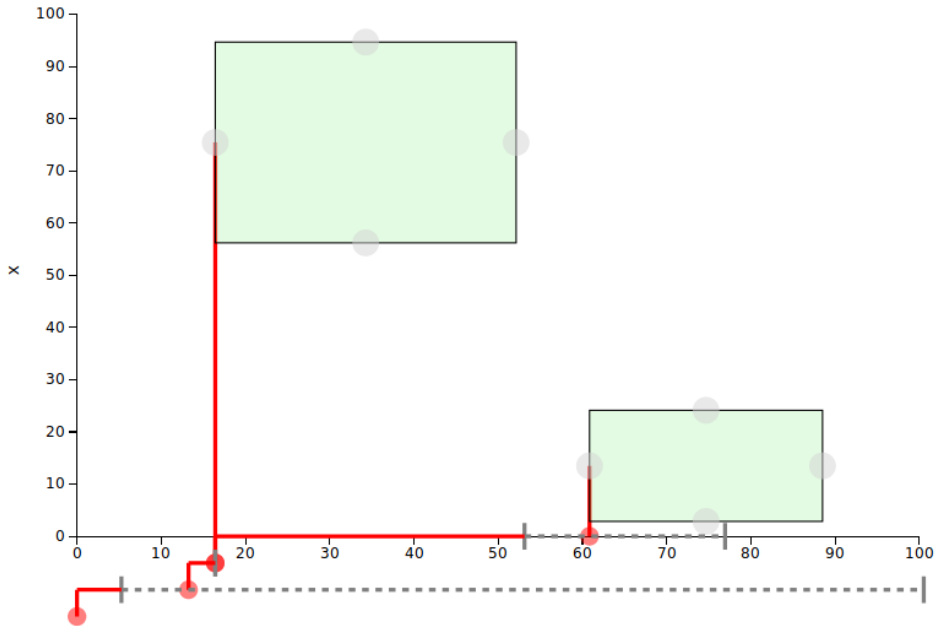


Figure 2.10: A screenshot showing a `TimeRails` diagram that could be used to filter a set of time-series (for example, stock prices). This query could not be expressed with TimeBoxes alone, because it expresses a relation between the time of two boxes whose start times are not fixed.

be expressed using `TimeRails` by augmenting the system to include the derivative of the relevant variable(s). TimeSearcher provides support for the Globally operator (a single TimeBox), and the Finally or Finally Globally operator (a Variable TimeBox), but not for any other combinations of temporal operators.

We implemented a prototype application that allows the user to search a database of time-series by drawing a `TimeRails` diagram. Figure 2.10 shows an example query that expresses a property that can easily be expressed using `TimeRails`, but could not be expressed using TimeBoxes alone. It searches for time-series which fall to a certain range *within a certain time* of having been in an upper ranges; the rails are required to represent the *within a certain time* constraint. Oscillation is another property that could be potentially useful for searching, and be specified by using nested operators that can be represented using `TimeRails` but not Timesearcher (see examples of specifying oscillations in Chapter 4).

2.4.2 *CRN synthesis by SAT modulo ODE*

The `TimeRails` interface can be used to express the desired dynamics for a Chemical Reaction Network, and a network satisfying the constraint can be synthesized using the method of SAT *modulo* ODE (Chapter 3). This approach requires the specification to have a restricted form (e.g., nested temporal operators are not permitted), so we modify `TimeRails` to restrict what can be drawn to what can be handled (Section 3.3.1).

2.4.3 *Bayesian design*

If the specification is interpreted using a quantitative semantics, then it is possible to calculate a numerical score indicating how well a simulation trace meets the specification. This error score can then be used to modify designs in order to meet the specification. This is discussed in Chapter 4.

2.5 SUMMARY & CONTRIBUTION

This chapter has described a diagrammatic notation for formulae in STL, and an interactive editor that can be used to create, edit, and interact with such formulae. In contrast to previous work, the notation proposed in this chapter can represent any Signal Temporal Logic specification in a consistent way. The innovations that provide this generality are representing operators sliding *along* each other rather than *inside* each other, allowing the direct linking of times, and separating out the representation of logical operators from the representation of values and times.

This editor is used to create specifications for Bayesian Design in Chapter 4, and for CRN synthesis in Chapter 3.

3

CRN SYNTHESIS BY SAT MODULO ODE[1]

3.1 MOTIVATION & CONTRIBUTION

Trying to design a CRN 'by hand' to meet a specific aim is typically a frustrating and time-consuming process, requiring many iterative adjustments to the topology and parameters before the desired results are obtained. It is therefore desirable to replace this manual process with an automated design procedure.

This chapter describes a system for automatically synthesizing a Chemical Reaction Network to meet a specification. The synthesized CRN can be considered to be a design for a system that is still 'abstract', in the sense that the actual chemical identity of each species is not determined. In some cases, this may be all that a user wants (for example, their goal may be to find an example system with a particular property as a starting point for further mathematical analysis). In other cases, a user may want the ability to actually construct a physical system that instantiates the CRN; there are existing algorithms for designing a set of DNA sequences that will instantiate a CRN as a set of strand displacement reactions [25, 143].

Section 3.2 provides background information that is relevant specifically to this chapter, including (Boolean) SATisfaction (SAT) and SMT solvers, Section 3.2.6 explains how SMT solvers can be used to synthesize CRNs, Section 3.3 introduces `CRNSynth`, our tool for CRN synthesis, Section 3.4 illustrates how this can be used to solve several example problems, and Section 3.5 is the conclusion.

1 This chapter describes joint work conducted with Max Whitby.
The graphical representations for the CRN sketches and specifications were developed by me; the method for formulating an ODE-SAT problem was developed by Max (and previously published as [23]); the code for formulating the ODE-SAT problem, calling the solver, and processing the results was written jointly; the example problems were formulated jointly.

3.2 INTRODUCTION

This section provides background information about satisfaction problems and algorithms to solve them that is needed to understand how the `CRNSynth` tool described in Section 3.3 is intended to work.

Definitions

An atomic formula or atom contains no logical connectives (conjunction or disjunction) or negation. A literal is an atom or its negation. A box (or hyper-rectangle) is a set of intervals, one for each variable.

3.2.1 *SAT and SMT*

The SAT problem asks whether there is an assignment of values to the variables in a Boolean formula such that the formula evaluates to true (i.e., that the values *satisfy* the formula). An extension to SAT is the problem of SMT, which provides interpretations for particular predicate and function symbols [9, 10, 107, 109].

For example, the theory of *difference logic*[2] allows formulae such as $(x \leq y) \wedge \neg(x \leq y - 3)$ to be interpreted with the usual arithmetic meanings for the symbols $-$, $<$ and $\leq$. Other commonly used theories include various restrictions of arithmetic, arrays (which model array data structures in computer programs), and fixed-width bit-vectors (which are useful when reasoning about digital logic circuits).

One way to solve SMT problems (the *eager* approach) is to translate the formula to be solved into an equivalent Boolean formula by applying some transformation that depends on the particular theory being used. For example, if the domain of an integer-valued variable is bounded, it is possible to replace it with several new Boolean variables, each of which is true if the integer has a particular value, and a new term requiring that exactly one of these variables is true. An alternative approach (the *lazy* approach), is to integrate theory-specific

2 Difference logic allows atoms of the form $a - b = t$ or $a - b \leq t$, where a and b are uninterpreted constants and t is an integer.

decision procedures into a SAT solver. Regardless of which approach is taken, there is a SAT solver at the core of an SMT solver.

SAT problems are often solved by converting to conjunctive normal form[3], and applying one of the many variants of the DPLL algorithm (Table 3.1). This tries to incrementally assign values to variables in a formula by deducing the truth value of a literal if this is possible, guessing a value if no deductions can be made, and backtracking if a guess leads to an inconsistency. Deductions act on *unit clauses*, in which all literals but one are false, requiring the remaining literal to be true, and the corresponding atom to be true (if the literal was the atom) or false (if the literal was its negation). DPLL can be represented as a transition system that can apply rules in order to transition between states, each of which is represented by either $fail$ or a pair $M \parallel F$, in which F is a set of clauses, and M is a set containing a partial assignment of truth values to the atoms that appear in F. Initially, we do not know the truth values of any atoms, and so M is the empty set $\emptyset$. The solver attempts to obtain a sequence of states (a *derivation*) that begins in the state $\emptyset \parallel F$, and ends with either a complete assignment of truth values or *Fail* (indicating that the problem is unsatisfiable). Different SAT solvers using the DPLL framework may make different choices about which transition rule to apply at each step, and may also perform preprocessing steps to re-write the problem in a form that may be more efficiently solved.

The DPLL[4] algorithm for SAT solving can be extended to give the $\mathcal{T}-$DPLL, or DPLL(T), algorithm for SMT solving, which adds the concept of *theory deduction* (to augment *Boolean deduction*) and *theory conflicts* (to augment *Boolean conflicts*). This is done by adding a theory solver which reports whether a given set of literals (i.e., atomic formulae and their negations) is satisfiable under that specific theory.

3 A formula is in conjunctive normal form if it is a conjunction of one or more clauses, each of which is a disjunction of literals. In other words, each clause contains only logical ORs (and negations), and these clauses are combined using logical ANDs.

4 DPPL is an eponym, rather than informative name: it is an abbreviation of Davis–Putnam–Logemann–Loveland, because the algorithm was introduced by Martin Davis, George Logemann and Donald W. Loveland [30] as a refinement of the Davis-Putnam algorithm introduced by Martin Davis and Hilary Putnam [31].

	Rule	Current State	Next State	Necessary Conditions
Basic DPLL	**Start**		$\emptyset \parallel F$	-
	Unit Propagate *	$M \parallel F, C \vee l$	$Ml \parallel F, C \vee l$	$\begin{cases} M \models \neg C \text{ (M falsifies C)} \\ l \text{ is undefined in } M \end{cases}$
	If a clause implies the value of a literal (l), assign that value.			
	Decide (Guess)	$M \parallel F$	$M, l^\bullet \parallel F$	$\begin{cases} l \text{ or } \neg l \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$
	Guess a value for a literal, and record as a 'decision literal' ($l^\bullet$).			
	Fail	$M \parallel F, C$	Fail	$\begin{cases} M \models \neg C \text{ (M falsifies C)} \\ M \text{ contains no decision literals} \end{cases}$
	If there are no decision literals, there is no potential to back-track; so if M falsifies a clause then F is unsatisfiable.			
	Backtrack *	$Ml^\bullet N \parallel F, C$	$M\bar{l} \parallel F, C$	$\begin{cases} Ml^\bullet N \text{ falsifies } C \\ l \text{ last decision literal} \end{cases}$
	Resolve a conflict by reversing the most recent guess ($l^\bullet N$).			
	Backjump	$Ml^\bullet N \parallel F, C$	$Mk \parallel F, C$	$\begin{cases} Ml^\bullet N \text{ falsifies } C \\ \text{for some clause } D \vee k\text{:} \\ \bullet\ F, C \models D \vee k, \\ \bullet\ M \text{ falsifies } D, \\ \bullet\ k \text{ is undefined in } M, \\ \bullet\ k \text{ or } \neg k \text{ occurs in a clause of } F \end{cases}$
	Resolve a conflict by reversing multiple guesses.			
Clause learning	**Learn** *	$M \parallel F$	$M \parallel F, C$	$\begin{cases} \text{all atoms of } C \text{ occur in } F, \\ F \models C \end{cases}$
	Add a new clause (C) to formula (e.g to avoid reaching a conflict clause again).			
	Forget *	$M \parallel F, C$	$M \parallel F$	$F \models C$
	Delete a learned clause (C).			
	Restart	$M \parallel F, C$	$\emptyset \parallel F$	-
	Solved	$M \parallel F$		All literals defined in M

Table 3.1: Abstract view of DPLL algorithm, based on the presentation in [114]. In DPLL *modulo* theory (DPLL-$\mathcal{T}$), the **Propogate** rule is supplemented by $\mathcal{T}-$**Propogate**, and **Learn**/**Forget**/**Backjump** are replaced by their $\mathcal{T}$ equivalents. The $\mathcal{T}-$ versions of the transition rules indicated by an asterisk above are obtained by replacing $\models$ with $\models_{\mathcal{T}}$ (indicating that the literals are jointly consistent with the theory) in the original definition.
In this notation, the state $M \parallel F$ consists of a set of clauses F, and a set M containing a partial assignment of truth values to the atoms that appear in F. F, C is a shorthand for $F \cup C$, and $l^\bullet$ indicates a literal whose value was chosen by a Decide rule.

3.2.2 *SAT modulo ODE*

SAT *modulo* ODE concerns the satisfiability of logical formulae that include Ordinary Differential Equations. Its development was initially motivated by hybrid dynamical systems, which exhibit discontinuous jumps between different modes; within each mode, they exhibit continuous dynamics (which may differ between modes). For example, a model of the temperature in a house may include mode transitions corresponding to a heater turning on and off when particular temperature thresholds are crossed. In typical applications, hybrid systems arise from combining a continuous dynamical system with a discrete controller, but they can also arise due to dynamical systems that themselves contain discontinuities (for example, a simple model of a ball instantaneously changing direction when it bounces on the floor).

The iSAT-ODE solver extends the interval SAT solver iSAT [44] to handle ODEs, by combining it with an enclosure-safe ODE solver. The following subsections will explain what this means.

3.2.3 *Interval SAT*

Interval SAT, as implemented in tools such as iSAT [44], is an extension to DPLL that is quite different to DPLL-$\mathcal{T}$. As the name suggests, it keeps track of intervals that bracket the value of each non-Boolean variable, and interprets arithmetic operators using the standard extensions to interval arithmetic[5].

It attempts to iteratively narrow these intervals by making *deductions* that prune off definite non-solutions (using either unit propa-

5 For two variables $\mathbf{a}, \mathbf{b} \in \mathbb{IR}$:

$$\mathbf{a} + \mathbf{b} = [\underline{\mathbf{a}} + \underline{\mathbf{a}}, \overline{\mathbf{a}} + \overline{\mathbf{b}}]$$
$$\mathbf{a} - \mathbf{b} = [\underline{\mathbf{a}} - \overline{\mathbf{a}}, \overline{\mathbf{a}} - \underline{\mathbf{b}}]$$
$$\mathbf{a} \times \mathbf{b} = [\min\left\{\underline{\mathbf{ab}}, \underline{\mathbf{a}}\overline{\mathbf{b}}, \overline{\mathbf{a}}\underline{\mathbf{b}}, \overline{\mathbf{ab}}\right\}, \max\left\{\underline{\mathbf{ab}}, \underline{\mathbf{a}}\overline{\mathbf{b}}, \overline{\mathbf{a}}\underline{\mathbf{b}}, \overline{\mathbf{ab}}\right\}]$$
$$\mathbf{a}/\mathbf{b} = [\underline{\mathbf{a}}, \overline{\mathbf{a}}] \times [1/\overline{\mathbf{b}}, 1/\underline{\mathbf{b}}]$$

If $\underline{x}$ and $\overline{x}$ are represented by floating-point numbers, it is necessary to perform directed rounding to ensure the entire interval resulting from an arithmetic operation is bracketed ($\underline{x}$ must be rounded downwards and $\overline{x}$ must be rounded up).

gation or interval constraint propagation), and making *decisions* by splitting an interval, guessing that one half will contain a solution, and trying to propagate the resulting bound (backtracking and trying to propagate the other part of the interval if necessary).

It can be considered to extend DPLL by modifying several of the transition rules:

(UNIT) INTERVAL CONSTRAINT PROPAGATION : if a constraint becomes unit it is immediately used to prune intervals by excluding any definite non-solutions, and the new intervals are then propagated to other constraints in which the affected variables appear. For example, propagation from the literal atom $x < 5$ may reduce the upper-bound of x; if another constraint required that $y < 2x$ then the upper-bound of y may also be reduced. There may be several iterations of pruning and propagation, stopping when the intervals are shrinking by an amount less than the user-defined *absolute bound progress*.

INTERVAL DECISION : if it is not possible to make further deductions by constraint propagation, the solver must make a guess by splitting the interval bracketing a variable into two halves, and selecting one of the two halves. If a Boolean variables are treated as integers in the interval $[0, 1]$ then making a decision about its value is a special case of applying the interval decision rule.

INTERVAL CONFLICT LEARNING : this adds an additional clause to record a region of the search space (e.g., $(x > 5) \wedge (x < 2)$) that has been found to cause a conflict, and thus confirmed to contain no solutions. This will then be used by unit-propagation to prevent this box from being revisited, hopefully saving time.

UNCERTAIN : the solver terminates if all variables are bracketed by intervals with a width below the user-defined *minimum splitting width*. As pruning excluded only definite non-solutions, the candidate solution box may or may not contain points that satisfy the formulae.

TERMINATION : if all clauses are satisfied, then the entire box is a solution

As with DPLL and DPLL-$\mathcal{T}$, there is considerable scope to design clever heuristics for choosing which transition rules to apply at each step, and what to apply them to (for example, when deciding which interval to split, the solver could select Booleans, the widest intervals, the intervals most frequent involved in conflicts, or those matching some other criterion).

3.2.4 *Enclosure-safe ODE solvers*

The tool Validated Numerical ODE solver developed using Literate Programming (VNODE-LP) was developed to solve the following problem [112, p.3]:

> We consider the IVP
>
> $$y'(t) = f(t, y), y(t_0) = y_0, y \in \mathbb{R}^n, t \in \mathbb{R}$$
>
> We denote the set of closed (finite) intervals on $\mathbb{R}$ by
>
> $$\mathbb{IR} = \{\mathbf{a} = [\underline{\mathbf{a}}, \overline{\mathbf{a}}] \mid \underline{\mathbf{a}} \leq x \leq \overline{\mathbf{a}}, \underline{\mathbf{a}}, \overline{\mathbf{a}} \in \mathbb{R}\}$$
>
> Given a point $t_{end} \neq t_0$ ($t_{end} \in \mathbb{R}$) and $\mathbf{y}_0 \in \mathbb{IR}^n$, the goal of VNODE-LP is to compute $\mathbf{y}_{end} \in \mathbb{IR}^n$ at t_{end} that contains the solution to the IVP at t_{end} for all $y_0 \in \mathbf{y}_0$. If VNODE-LP cannot reach t_{end}, bounds on the solution at some $t^\star$ between t_0 and t_{end} are reported.

Taylor's theorem[6] states that, for a continuous single-valued function $f(x)$ of x having continuous derivatives with respect to x, for all a and h there exists some $a \leq \zeta \leq a + h$ such that

6 Specifically, the version including the Lagrangian form for the remainder $R_n(h)$.

$$f(a+h) = f(a) + hf'(a) + \frac{h^2}{2!}f''(a) + \ldots + \frac{h^{(n-1)}}{(n-1)!}f^{(n-1)} + \overbrace{\frac{h^n}{n!}f^{(n)}(\zeta)}^{R_n(h)}$$

This suggests that an integration scheme could not only *estimate* the solution $y(t+dh)$ after a time-step of size dh from an estimate of $y(t)$ and knowledge of the derivative function $y'(\bullet)$, but also identify an *interval guaranteed to enclose* the solution at each time-step by propagating an interval around the initial condition $y(t) = y_0$ using the standard rules for interval arithmetic. Producing a robust scheme that works well in practice is more complicated than this, and further discussion can be found in [37 🎓, 112]

3.2.5 *SAT modulo ODE*

The exposition of this algorithm here is necessarily brief: a more complete explanation of the method used by iSAT-ODE is provided in [37 🎓, Chapter 3].

The problem specification provided to iSAT-ODE has several components: a definition section that defines variables, an initialization section that defines initial values for variables (and any logical formulae that must hold at the starting time), guard conditions that determine whether a particular mode transition can occur, formulae that relate the values of variables before and after a mode transition, flow constraints that specify ODEs that define the dynamics of particular variables within each mode, and a target or goal condition that must be met at the end time. Typically a number of Boolean variables are used to record which mode the system is in, and these are used to formulate guard conditions and to control when each ODE applies.

An initial pre-processing step filters out ODE and flow invariant constraints and registers them with the ODE solver, replacing them with new Boolean trigger variables. If a trigger variable has a value of 1, then this indicates that the corresponding constraint is active and can be used for pruning. This enables communication between

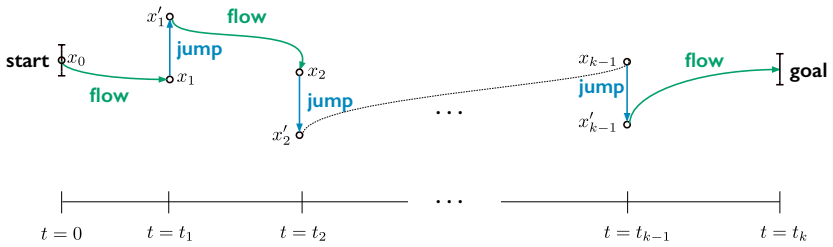


Figure 3.1: Logical encoding of a SAT *modulo* ODE problem. The model is unrolled k-times: *flow* transitions during each of the k modes are constrained to be trajectories of the corresponding ODE; transitions at mode switches are constrained by the corresponding *jump* conditions. The problem is to either find a set of values for $t_1, t_2, \ldots, t_k, x_0, x_1, \ldots, x_k, x'_1, x'_2, \ldots, x'_{k-1}$ that satisfy the flow and jump constraints, start in some specified box, and end with x_k in the goal or target region, or to show that no such combination is possible.

the interval-SAT solver and the ODE solver: the trigger variables allow the interval-SAT solver to reason about *which* ODE and flow invariants apply constraints apply, then communicate this to the ODE solver and receive back an interval deduction. The ODE solver is thus added to the interval SAT solver as a separate module, analogously to the way that a theory solver is added to a DPLL based SAT solver.

The SAT *modulo* ODE solver tries to find a candidate solution with the minimum possible number of mode transitions k, by starting with $k = 1$ and iteratively incrementing k if the problem is found to be unsatisfiable, terminating if a user-specified maximum unrolling depth is reached. At each iteration, the model is unfolded k times: each state variable in the model is duplicated, creating a new variable that records its value at the end of each mode (in addition of a variable recording its initial value). Guard conditions and flow constraints are similarly duplicated for each mode (see Figure 3.1).

Given an expanded logical formula, the solver first tries to apply Interval Constraint Propagation to the non-ODE constraints. The ODE solver then attempts to perform deductions by applying interval constraint propagation to the ODE constraints. It considers trajectories that begin in some *prebox*, and end in some *postbox* after *delta_time* (represented by an interval) has elapsed. Forward propagation prunes the postbox, by removing parts that cannot be reached

by any trajectory that starts in the prebox; backwards propagation prunes the prebox, by removing parts from which any trajectory will be outside the postbox for all times in the *delta_time* interval. These pruning operations rely on an enclosure-safe ODE solver to construct the postbox reachable from a prebox, and (by reversing the sign of time in the ODEs) *vice versa*.

3.2.6 *CRN synthesis by SMT*

There have been two published approaches for synthesizing a CRN to meet a specification using SMT, based on two different underlying theories. In this subsection we briefly introduce both.

3.2.7 *CRN design by SMT modulo linear integer arithmetic*

Dalchau *et al.* [27] uses the Continuous Time Markov Chain interpretation of CRN kinetics, and considers constraints in the form of two state predicates, the first of which must be satisfied by the initial state of the path, and the second as some later time (which is considered to be the path's final state, as the state is unconstrained thereafter). They allow a state *state predicate* $\phi : X \to \mathbb{B}$ to have the syntax [27, p.4]:

$$
\begin{aligned}
\phi &::= E_b \\
E_b &::= true | false | E_c | \neg E_b | E_b \triangleright E_b \text{ where } \triangleright \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\} \\
E_c &::= E_a \triangleright E_a \text{ where } \triangleright \in \{<, \leq, =, >, \geq\} \\
E_a &::= s \in \Lambda | c \in \mathbb{Z} | E_a \triangleright E_a \text{ where } \triangleright \in \{+, -, \times\}
\end{aligned}
$$

They require the user to specify the number of reactions ($|\mathcal{R}| = M$) and species ($|\Lambda| = N$). These values determine the shape of two stoichiometry matrices $\mathbf{r} \in \mathbb{N}_0^{M \times N}$ and $\mathbf{p} \in \mathbb{N}_0^{M \times N}$, which record the number of molecules of each species participating in each reaction as a reactant or product, respectively.

Constraints on $\mathbf{r}$ and $\mathbf{p}$ are added to encode the requirements that species labeled as *inputs* are consumed by at least one reaction, species labeled as *outputs* are produced by at least one reaction, no two reactions have the same net stoichiometry change for all species, and every reaction results in a net production or consumption of at least one species. The final two of these conditions ensures that the number of reactions is exactly M, rather than being less than or equal to M.

The system's state $\mathbf{x}_t$ is encoded by a vector whose elements are the molecule counts for each species. The matrices $\mathbf{r}$ and $\mathbf{p}$ define the allowable transitions between states: if each element of $\mathbf{x}_t$ is greater than $\mathbf{r}_i$ then the i'th reaction can occur, so a transition from state $\mathbf{x}_t$ to state $\mathbf{x}_t - \mathbf{r}_i + \mathbf{p}_i$ is possible. This fits neatly into the framework provided by the theory of linear integer arithmetic.

The user also chooses a parameter K that represents the maximal trajectory length to be considered. This establishes a Bounded Model Checking problem, in which the SMT solver must try to find $\mathbf{r}$ and $\mathbf{p}$ together with a consistent sequence of states $\boldsymbol{x}_i$ that satisfies the path specification.

However, even if a path is found that is consistent with $\mathbf{r}$ and $\mathbf{p}$, the probability that this system would follow this path may be very low. Dalchau *et al.* therefore perform a second step, in which they try to identify the values of rate constants that maximise the probability that a simulated trajectory of the synthesized system would meet the specification.

3.2.8 *CRN design by SMT modulo ODE*

Cardelli *et al.* [23] adopt an approach that is conceptually very different: they use specifications of a different form, use a different modeling approach that represents CRNs using ODEs (either the Rate Reaction Equation or Linear Noise Approximation – see Section 1.3) rather than Markov chains, and formulate a satisfaction problem *modulo* the theory of ordinary differential equations, rather than the theory of linear integer arithmetic.

We adopt and extend this approach: the details are explained further in Sections 3.3.2 and 3.4.1.

3.3 `crnsynth` TOOL

Cardelli *et al.* [23] provided three example problems to illustrate the method that they introduced. However, at this time they had not written a software tool to implement their method: instead, they manually constructed the input files for the SMT solvers iSAT-ODE and dReach [85], extracted the synthesized parameter values from the output of the solvers, and substituted these into the original parametric CRN sketch. Each of these steps has the potential for error, and constructing the input files is particularly tedious.

`CRNSynth` is a software tool that automatically implements these steps. A Python package provides the core functionality of converting an CRN sketch and a specification into an SMT-ODE problem expressed in a format that can be solved by dReach or iSAT, calling one of these solvers, and extracting parameters from its output. There is a graphical interface layered on top of this, which is implemented as a website using Python and the Flask microframework on the backend, and integrating `TimeRails` on the frontend. The core Python package was written jointly by me and Max Whitby; the graphical web interface was developed by me.

3.3.1 *User Interface*

`crn-designer` is a web application, which the user opens in their web browser. Once the user has logged in and clicked a link to create a new project, they are presented with a page that lets them visually draw a CRN sketch and `TimeRails` diagram (Figure 3.2).

CRN Sketch

A CRN sketch is represented visually as a bipartite directed-graph in which nodes represent both reactions (with a circular edge) and

Route	Description
/about/ (GET)	About page
/logout/ (GET)	Login
/register/ (POST,GET)	New user registration
/projects/ (GET)	View list of projects
/projects/<int:project_id>/ (GET)	View single project
/projects/add (POST,GET)	Create project
/projects/<int:project_id>/edit (POST,GET)	Edit details of a project (title, description, isPublic)
/projects/<int:project_id>/delete (POST,GET)	Delete project
/projects/<int:project_id>/copy (POST,GET)	Copy an existing project
/projects/<int:project_id>/dReach (GET)	Download `.drh` file encoding SMT problem for dReach
/projects/<int:project_id>/iSAT (GET)	Download `.hys` file encoding SMT problem for iSAT
/projects/<int:project_id>/solve (POST,GET)	Call dReach or iSAT to solve problem on the server
/projects/<int:project_id>/process (POST,GET)	Extract parameters from uploaded dReach/iSAT output and simulate
/projects/<int:project_id>/**save** (POST)	Save modified specification
/projects/<int:project_id>/**saveCRN** (POST)	Save modified CRN sketch
/projects/<int:project_id>**saveCode** (POST)	Save modified CRN code
/projects/static/<path:filename>(GET)	Fetch a file that is static (i.e., not dynamically generated in response to the request)

Table 3.2: Routes provided by the `CRN-designer` tool. Actions that edit the contents of the database require an HTTP POST, rather than GET, request. Requests to most of these routes are made by the user clicking on links, but requests to several (indicated in **bold**) are made by the front-end of the editor in order to save the state of the project after editing.

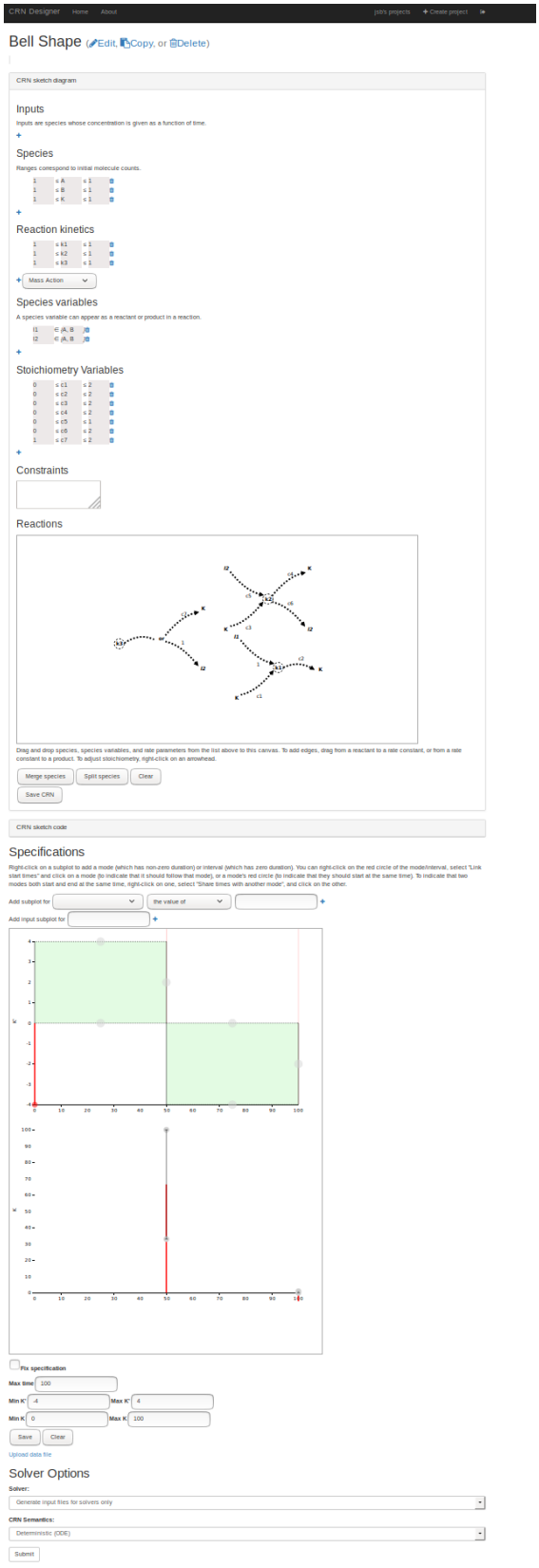


Figure 3.2: Screenshot of `CRNSynth`, showing the CRN sketch and `TimeRails` specification.

species (with no edge). Edges link reactants to reactions, and reactions to products, labelled by the corresponding stoichiometric coefficient (which may be either integers or variables).

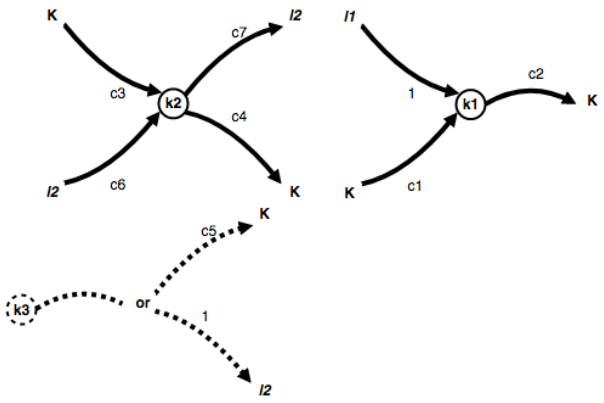
A reaction may be marked as optional: the circular border of an optional reaction, and any incident edges, are drawn with dashed lines. If a species' stoichiometric coefficient is a variable, then the corresponding edge is labelled with the variable's name rather than an integer value.

A sketch can also express joint constraints over a (species, stoichiometry) pair: this is represented by a dummy node labelled 'or' with a separate link from (to) each alternative reactant (product) labelled with the corresponding stoichiometry.

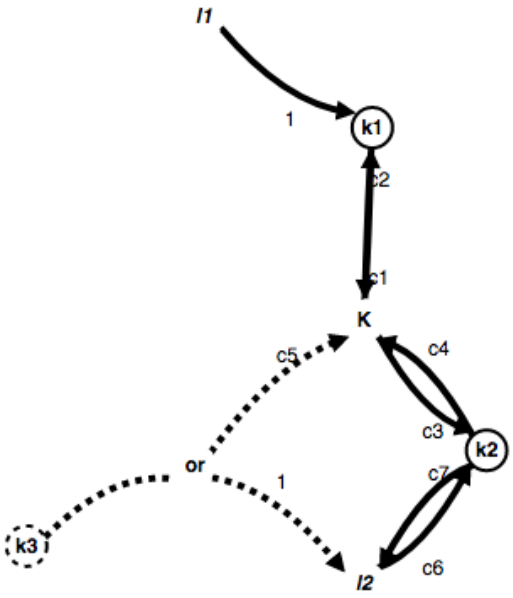
The user can automatically switch between two alternative representations: in one, each species is represented by exactly one node; in the other, species are duplicated as necessary to allow each reaction to be represented by an isolated subgraph (Figure 3.3). The transformation between these two networks is achieved by Algorithms 2 and 3.

Inputs

It is often desirable to design systems that exhibit a particular response to changes in the value of an input (physically represented by the concentration of some chemical whose concentration is not determined by the system itself, but by something external to it). Practically, inputs may correspond to some aspect of the environment that a circuit is intended to detect and respond to. Ideally, the user would be able to specify that *for all* inputs satisfying a particular property, the output should also satisfy some property. However, such quantifiers are difficult to work with, so we instead allow the user to specify a *single, particular* input as a function of time, and require that if this input is provided then the system's output should satisfy some property. For example, the user may want a system that achieves a particular response whenever the molecule count of input species A increases above 10; in order to obtain a synthesis problem that we are able to solve, we require them to draw a specific input time series, and then



(a)



(b)

Figure 3.3: The two views of a CRN in `CRNSynth`. Both are node-link diagrams (with several node types), but in one each species is represented by exactly one node; in the other, species are duplicated as necessary to allow each reaction to be represented by an isolated subgraph.

Algorithm 2 Merging nodes corresponding to the same species in a CRN sketch.

```
procedure mergeDuplicatedSpecies(nodes, links)
    speciesNames ← []
    firstSpeciesWithName ← []

    for var i=0; i<nodes.length; i++ do
        if nodes[i] represents choice or reaction then
            continue
        l ← nodes[i].label

        if l not in speciesNames then
            append l to speciesNames
            firstSpeciesWithName[l] ← nodes[i]
        else
            for var j = 0; j < links.length; j++ do
                if links[j].source == nodes[i] then
                    links[j].source ← firstSpeciesWithName[l]
                if links[j].target == nodes[i] then
                    links[j].target ← firstSpeciesWithName[l]
            remove i'th node from nodes
            i ← i − 1
    Redraw graph using nodes and links
```

attempt to synthesise a system that shows the desired response to this.

We allow a user to designate species as 'input species', whose concentration follows a pre-determined trajectory over time, rather than being determined by the kinetics of reactions. Such a species can appear as a reactant, in which case it influences the rate of the corresponding reaction but is effectively not consumed. This concentration of an input species as a function of time can be a sum of linear, sigmoidal (Hill) and bell-shaped (Gaussian) terms (Figure 3.4). These have closed-form expressions that can be differentiated, allowing the concentration of the input species to be expressed as a non-autonomous ODE, satisfying the requirement of iSAT-ODE that all variables that appear in an ODE be defined by an ODE. Note that any function can be approximated as the sum of delta-functions (which Gaussians approach in the limit as they become narrower) or step functions (which sigmoids approach as they become steeper).

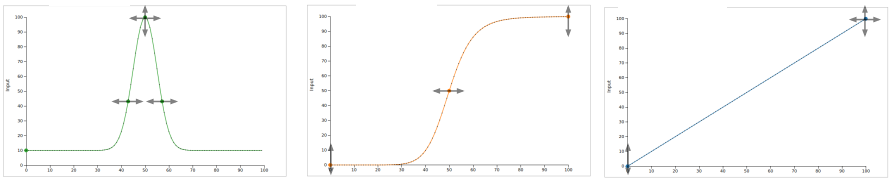
Algorithm 3 Creating duplicate nodes representing the same species in different reactions in a CRN sketch.

```
procedure splitDuplicatedSpecies(nodes, links)
    speciesTypes ← ["species", "speciesVariable", "input"]
    for i = 0; i < nodes.length; i++ do
        reaction ← nodes[i]
        skip node unless it is a reaction
        participants ← []
        inAnotherReaction ← []
        for j = 0; j < links.length; j++ do
            source ← links[j].source
            target ← links[j].target

            if target == reaction and source not in participants then
                append source to participants
            if source == reaction and target not in participants then
                append target to participants

            if source.type in speciesTypes then
                if source not in inAnotherReaction then
                    if target! == reaction then
                        append source to inAnotherReaction

            if target.type in speciesTypes then
                if target not in inAnotherReaction then
                    if source! == reaction then
                        append target to inAnotherReaction

        for j = 0; j < participants.length; j++ do
            s ← participants[j]
            if s in inAnotherReaction then
                n ← {id : ++lastNodeId, type : s.type, label : s.label}
                append n to nodes

                for k = 0; k < links.length; k++ do
                    if links[k].source == reaction then
                        if links[k].target == s then
                            links[k].target ← newNode
                    if links[k].target == reaction then
                        if links[k].source == s then
                            links[k].source ← newNode
    Redraw graph using nodes and links
```

Shape	$I(t)$	$\dot{I}(t)$
Linear/Affine	$at + b$	a
Sigmoidal (Hill)	$a_0 + a' \frac{1}{(T/t)^n+1}$	$a' \frac{n(T/t)^n}{t((T/t)^n+1)^2}$
Bell-shaped (Gaussian)	$a_0 + a' \exp\left(-(-(t'/\sigma)^2\right)$	$\frac{a't'}{\sigma^2} \exp\left(-(t'/\sigma)^2\right)$

$$a' = (a_1 - a_0),\ t' = (t - T)$$

(a)



(b)

Figure 3.4: CRN Designer allows the use of input functions that are linear combinations of constant, linear, sigmoidal, and bell-shaped terms. Screenshot of `CRNSynth`, showing the control points that can be used to adjust each kind of input term. The two control points on the Gaussian are linked to ensure they remain symmetric about the peak: moving one also moves the other. The grey arrows indicate the directions in which each control point can be moved: they are not displayed, and were manually added to the screenshot, though the cursor does adopt these shapes when mousing-over these points.

The user can add a term by right-clicking and selecting the type of terms to add from a contextual menu. They can adjust the coefficients by opening a model dialog containing a form in which they can directly set precise values, or by dragging control points on the curve (Figure 3.4).

Specification

We use a diagrammatic representation that is a restriction of the `TimeRails` system (Chapter 2) to the fragment of logic that `CRNSynth` supports.

A specification is represented by a series of subplots with a common time-axis. Each subplot represents a different term: the expectation or variance of the concentration of a particular species, or a (first or second-) time-derivative of an expectation or variance.

Within a subplot, a user can draw a *mode rectangle*, indicating that the value of the corresponding term should lie in the correspond-

ing range. Importantly, neither the precise start and times nor the duration of the time range for which the condition holds is directly specified, but it is possible to link rectangles in order to express requirements about their ordering (a user can represent *meets* or *is equal to* relations between them, in the sense of Allen's interval algebra). The start time of a mode can be linked to another mode, indicating that the constraint represented by the second should apply from the time at which the first constraint ceases to hold; this is indicated by vertical alignment of the start of the second rectangle with the end of the first, and a vertical linking line. Two rectangles in different subplots can also be linked, so that the corresponding conditions start and end at the same time; this is indicated by vertical lines joining both the start and end edges of the rectangles. Leaving a gap between two mode rectangles is equivalent to drawing an intermediate mode during which no conditions apply.

It is also possible to add line segments (equivalent to zero-width rectangles), indicating that a condition need only hold momentarily, at the transition between two rectangles.

In the SAT-ODE problem these rectangles correspond to *modes* linked by discrete state-transitions. Each mode has the same dynamics, but applies different constraints to the state.

3.3.2 *Formulating the SMT problem*

We implement a `CRNSketch` class, whose constructor accepts a list of `reaction` objects, a list of optional reactions, and a list of input species. Each reaction has a list of reactant terms, a list of product terms, and a `RateConstant` object.

The term objects represent a chemical species and corresponding stoichiometric coefficient. However, to allow the synthesis of an CRN topology, not just the value of rate constants for a pre-determined topology, these may be parametric. Specifically, we allow terms to be a tree with any of the following forms:

Term ::=TermChoice
| (StoichiometryChoice, SpeciesChoice)
| (StoichiometryChoice, Species)
| (Stoichiometry, SpeciesChoice)
| (Stoichiometry, Species)

TermChoice ::=Term **or** Term
| Term **or** TermChoice

StoichiometryChoice ::=Stoichiometry **or** Stoichiometry
| Stoichiometry **or** StoichiometryChoice

SpeciesChoice ::=Species **or** Species
| Species **or** SpeciesChoice

where the pipe symbol indicates different choices that may be made when writing a sketch, and **or** indicates that something should be interpreted as representing a choice that may be taken at the time that a specific system is synthesized

The `CRNSynthesis` library implements `Term`, `Species`, `StoichiometryChoice`, and `Stoichiometry` objects, so a user can directly call these constructors to create the objects necessary to represent their sketch. If the graphical interface is used to draw a sketch, then a function iterates over the object representing the tree, and calls constructors as necessary (Algorithm 4). These generate a parametric set of ODEs that are the Chemical Langevin Equation for the corresponding CRN, and describe the rate of change of the expected value and variances of the concentration of each species (see Section 1.3). The specification may constrain variances: if it does not, generation of the equations for

the variances are skipped, giving just the deterministic Rate Reaction Equations, and resulting in a lower-dimensional system for the SMT solver to work with.

We introduce new structural decision variables that capture choices about the structure of the CRN. For each of the choices, we introduce a variable for each option, which takes the value of 1 if that option is chosen (and 0 otherwise); constraints are added to the generated SMT problem to ensure consistency.

Regardless of how this tree was generated, it can be transversed to obtain expressions that include these decision variables, and indicate:

- the net change in stoichiometry for each species in each reaction
- the number of molecules of each species participating as reactants in each reaction

This enables us to write down the reaction rate equations for a CRN sketch as the product of a matrix of stoichiometry changes and a vector of propensities: entries in both of these may be algebraic expressions that include these structural decision variables.

We introduce Boolean variables to indicate which mode the system is in, add constraints to allow only sequential progression between modes, and apply the user-specified invariant constraints (which apply during each mode) and guard conditions (which apply during transitions between modes).

3.4 EXAMPLES

3.4.1 *Bell-shape*

This problem originally appeared as an example in [23], in which it was manually converted to `.hys` format by Max Whitby and then solved by iSAT. It tries to synthesize a chemical reaction network such that the concentration of one of the species (K) is approximately bell-shaped: its concentration must initially be low, reach a maximum of at least 30, and then decrease.

Algorithm 4 Constructing an ODE describing the dynamics of a CRN sketch.

```
procedure CALCULATE_FLOW(isLNA, derivatives)
    propensities ← [reaction.get_propensity() for reaction in all_reactions])
    stoichiometry_change ← PARAMETRICNETREACTIONCHANGE()
    dSpeciesdt ← stoichiometry_change^T * propensities

    if isLNA then
        J ← dSpeciesdt.jacobian(real_species)
        C ← generateCovarianceMatrix(real_species)
        G ← generateG(stoichiometry_change, propensities)
        dCovdt ← J * C + C * J^T + G
        for i in range(C.cols * C.rows) do
            a[C[i]] ← dCovdt[i]

    derivative_expressions ← compute_derivatives(derivatives, a)
    extend a with derivatives of input species
    return a

procedure PARAMETRICNETREACTIONCHANGE
    change ← []
    for reaction in all_reactions do
        netChange ← 0
        for reactant in reaction.reactants do
            netChange ← ADD_STOICHIOMETRY_CHANGE(netChange, reactant, '−')
        for product in reaction.products do
            netChange ← ADD_STOICHIOMETRY_CHANGE(netChange, product, '+')
        change.append(netChange)
    return change

procedure ADD_STOICHIOMETRY_CHANGE(stoich_change, fragment, sign)
    for i, sp in enumerate(real_species) do
        if sp in fragment.specRep() then
            new_term ← sign + fragment.specRep().coeff(sp)
            stoich_change[i] = stoich_change[i] + " +" +new_term])
    return stoichiometry_change
```

```
procedure generateCovarianceMatrix(species_names)
    mat ← eye(len(species_names))
    for i, m in enumerate(species_names) do
        for j, n in enumerate(species_names) do
            if m == n then
                mat[i, j] = 'var' + m
            else
                mat[i, j] = 'cov' + n + m
    for x in range(len(species_names)) do
        for y in range(len(species_names)) do
            mat[x, y] ← mat[y, x]
    return mat

procedure generateG(stochVector, propensityVector)
    i ← 0
    G ← 0
    for i in range(len(propensityVector)) do:
        G+ = (stochVector[i])^T * stochVector[i] * propensityVector[i]
    return G
```

```
Class Reaction
        procedure get_propensity
            propensity ← 1
            for r in self.reactants do
                propensity ← propensity * r.constructPropensity()
            if self.is_optional then
                propensity ← propensity * self.variable_name
            return propensity
        ...
EndClass

Class ArbitraryRateReaction
        procedure get_propensity
            return self.reactionrate
        ...
EndClass
```

```
Class TermChoice
    procedure constructPropensity
        terms ← []
        for i, t in enumerate(self.possible_terms) do
            terms.append("%s%s * (%s)" %
(self.base_variable_name, i, t.constructPropensity()))
        return " + ".join(terms)
    procedure specRep
        substrings ← []
        for i, term in enumerate(self.possible_terms) do
            substrings.append("%s%s * (%s)" %
(self.base_variable_name, i, term.specRep()))
        return " + ".join(substrings)
    ...
EndClass

Class Term
    procedure constructPropensity
        if isinstance(self.species, LambdaChoice) then
            term_to_exponentiate ← self.species.constructChoice()
        else
            term_to_exponentiate ← self.species.name
        coeff ← self.coefficient
        if isinstance(coeff, Choice) then
            terms ← []
            for value in range(coeff.min, coeff.max+1) do
                terms.append("(%s ** %s)*%s_%s" %
(term_to_exponentiate, value, coeff.name, value))
            return " + ".join(terms)
        elsereturn term_to_exponentiate + "**" + coeff
    procedure specRep
        coeff ← self.coefficient
        if isinstance(coeff, Choice) then
            coefficient_expression ← 0
            for value in range(coeff.min, coeff.max+1) do
                coefficient_expression+ = value ∗ (coeff.name +
"_" + value)
            coeff ← coefficient_expression
        if isinstance(self.species, LambdaChoice) then
            return coefficient * self.species.constructChoice()
        else
            return coefficient * self.species.name
    ...
EndClass
```

The topology of the CRN is partly constrained by this CRN sketch (shown graphically in Figure 3.5):

$$
\begin{aligned}
c_1, \ldots, c_4 &\in \{0, 1, 2\} \\
c_5 &\in \{0, 1\} \\
c_6 &\in \{0, 1, 2\} \\
c_7 &\in \{1, 2\}
\end{aligned}
$$

$$
\lambda_1, \lambda_2 \in \{A, B\}
$$

$$
\begin{aligned}
\tau_1 &: \lambda_1 + c_1 K \xrightarrow{k_1} c_2 K \\
\tau_2 &: c_5 \lambda_2 + c_3 K \xrightarrow{k_2} c_6 \lambda_2 + c_4 K \\
\tau_3 &: \varnothing \xrightarrow{k_3} \{\lambda_2, c_7 K\}
\end{aligned}
$$

Encoding

To represent this, we introduce indicator variables that take the value of zero or one:

$$
\begin{aligned}
\lambda_{Xn} &= \mathcal{I}(\lambda_n = X) \\
c_{ij} &= \mathcal{I}(c_i = j) \\
\gamma_1 &= \mathcal{I}(r_3 \text{ produces } \lambda_1) \\
o_i &= \mathcal{I}(\text{ optional reaction } i \text{ occurs})
\end{aligned}
$$

When formulating the SAT-ODE problem, constraints are added to prevent these from being assigned inconsistent values (such as $c_{11} = 1$ and $c_{12} = 1$).

We can then write the dynamics as:

$$\dot{x} = SP$$

$$S = \begin{bmatrix} -\lambda_{B1} & -\lambda_{A1} & -c_{11} + c_{21} + 2c_{22} \\ \lambda_{B2}(-c_{51} + c_{61} + 2c_{62}) & \lambda_{A2}(-c_{51} + c_{61} + 2c_{62}) & -c_{31} - 2c_{32} + c_{41} \\ \lambda_{B2}\gamma_1 & \lambda_{A2}\gamma_1 & (1-\gamma_1)(c_{71} + 2c_{72}) \end{bmatrix}$$

$$P = \begin{bmatrix} k_1 \left(A\lambda_{A1} + B\lambda_{B1}\right)\left(Kc_{11} + c_{10}\right) \\ k_2 \left(c_{50} + c_{51}\left(A\lambda_{A2} + B\lambda_{B2}\right)\right)\left(K^2 c_{32} + Kc_{31}\right) \\ k_3 o_1 \end{bmatrix}$$

Result

Given the CRN sketch and specification, `crn-designer` synthesises a CRN; when simulated, this gives the results shown in Figure 3.6, which can be seen to satisfy the specification.

3.4.2 *Phosphorelay*

This problem originally appeared as an example in [23], in which it was manually converted to `.hys` format by Max Whitby and then solved by iSAT.

This example considers a model of a phosphorelay system, in which an input species B is produced at a constant rate, and causes the auto-phosporylation of $L1$ into $L1p$, which transfers this phosphate group to $L2$, which transfers it to $L3$, which de-phosphorylates itself as a first-order process. The specification requires the concentration of $L3p$ to be a sigmoidal function of time. This is encoded by two modes: the first derivative of $[L3p]$ is ≥ 0 in both modes, but the second derivative is ≥ 0 in the first mode and ≤ 0 in the second mode; $[L3p]$ is required to be > 100 at the end of the second mode. The corresponding CRN sketch and `TimeRails` diagram are shown in Figure 3.7. Simulation results for the synthesized system are shown in Figure 3.8.

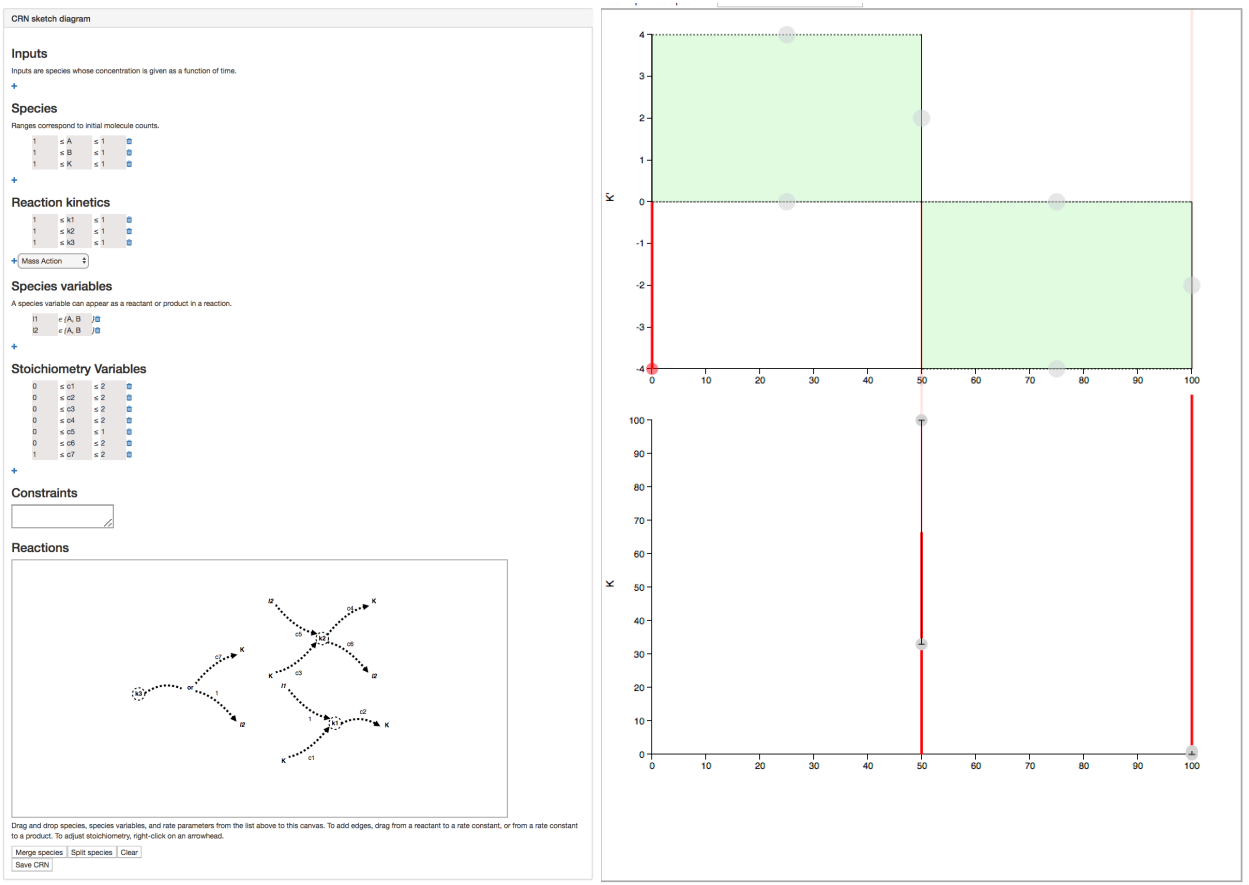


Figure 3.5: Bell-shape example. The CRN sketch (left) and `TimeRails` specification (right) provided as an input to `CRNsynth`. Each element of this figure is a screenshot from `CRNSynth`, but they have been cropped and combined into a single figure to save space.

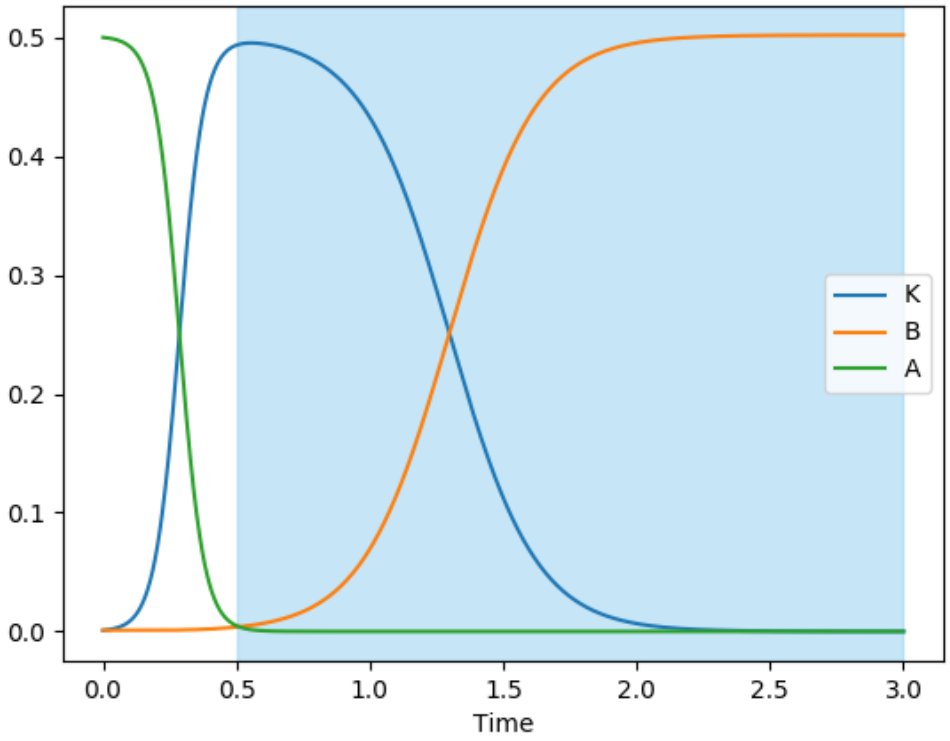


Figure 3.6: Plot of simulation results for the synthesized bell-shape system. As required by the specification, K has an approximately bell-shaped profile.

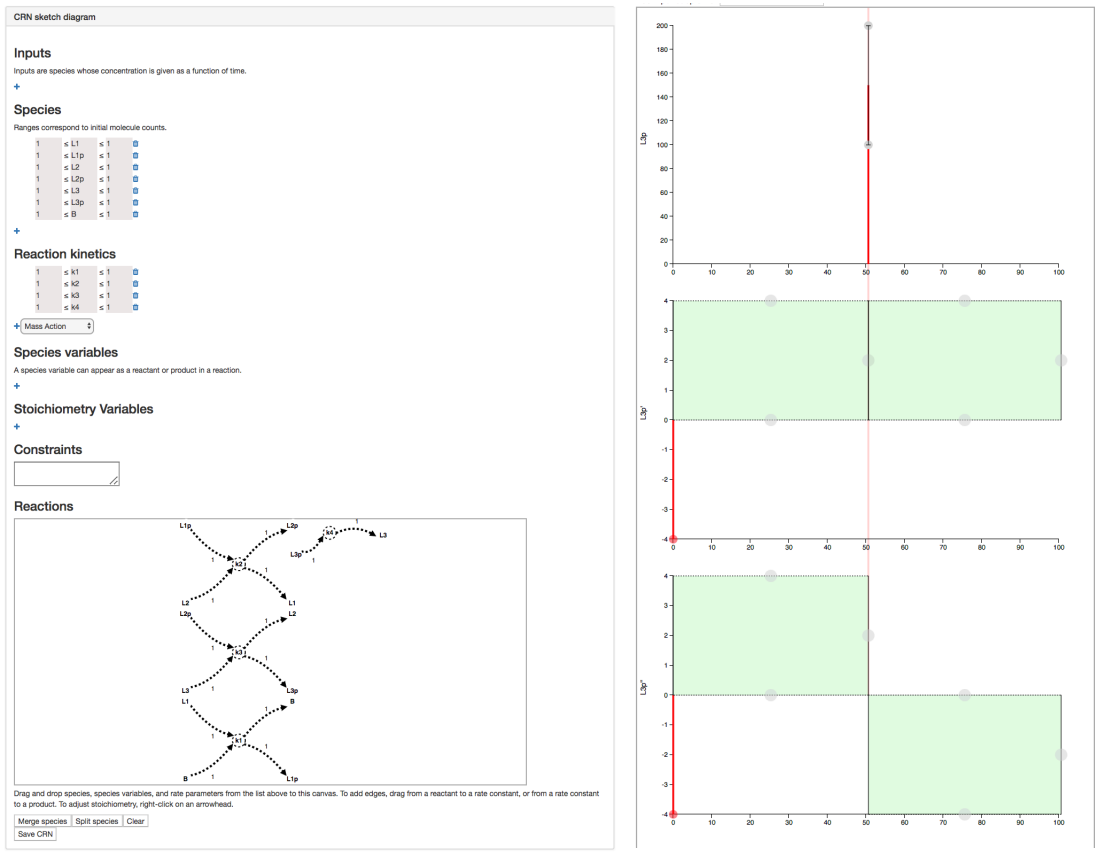


Figure 3.7: The CRN sketch (left) and `TimeRails` specification (right) provided as an input to `CRNsynth` for the phosphorelay example. Both elements of this figure are screenshots from `CRNSynth`, but they have been cropped and combined into a single figure to save space.

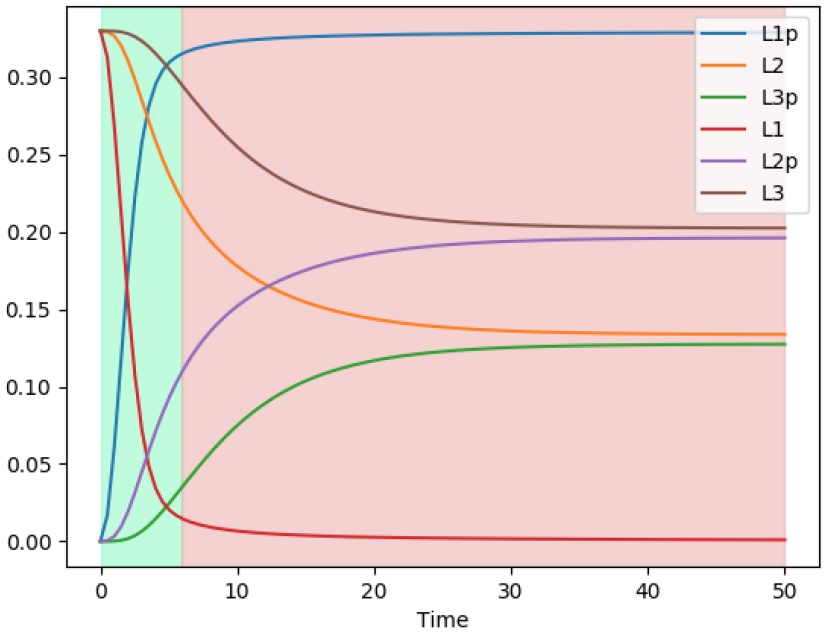


Figure 3.8: Plot of simulation results for the synthesized phosphorelay system with sigmoidal response.

3.4.3 *Toggle-switch*

This example tries to synthesize a toggle-switch; this is a new example, which has not been previously been formulated as an SMT-ODE problem.

In this example the CRN structure is fixed, and we are simply trying to synthesize parameters. This switch consists of two proteins, $cIts$ and $lacI$, each of which repress the transcription of the other; an input lifts the repression of $cIts$ by $lacI$. This is modelled as:

$$\text{input} \xrightarrow{g} \text{lacI} + \text{input} \qquad g = \frac{\alpha_2}{(1+k_2\,[\text{lacI}]\,k_1\,[\text{input}]+k_2)^2}$$
$$\text{lacI} \xrightarrow{hr} \text{lacI} + \text{cIts} \qquad hr = \frac{1}{(1+\alpha_1)^2}$$
$$\text{lacI} \xrightarrow{dr1} \emptyset$$
$$\text{cIts} \xrightarrow{dr2} \emptyset$$

$$k_1, k_2, dr1, dr2 \in [0,2]$$
$$\frac{dInput}{dt} = -356 \times \exp\left(-178(-0.5+t)^3\right)$$

We construct a specification in terms of a pulse of input: we require that the concentration of $cIts$ initially be low, but increase after the

pulse and remain above a threshold for a specified time after the pulse has ended. The simulations of the synthesized system shown in Figure 3.9 show that the output (the concentration of lacI) rises when a pulse of input is supplied, and remains high thereafter; if there is no input pulse, then the output remains low.

3.5 SUMMARY & CONTRIBUTION

This chapter has described a software tool (`CRNSynth`) for the automated synthesis of CRNs that meet user-provided specifications. The features of this tool are illustrated through several example problems.

The contribution of this chapter are:

- creating a software tool to automatically formulate the CRN synthesis problem as an SMT-ODE problem that can be solved by existing solvers, using the method originally described in [23]
- extending this approach to handle inputs, enabling the synthesis of CRNs with particular responses to a given input function
- demonstrating that this approach works on several example problems

Manually constructing input files for iSAT or dReach was extremely time consuming and error-prone, so we created a tool that made this process much faster whilst eliminating the potential for human errors, and proposed a way to incorporate time-varying inputs. We had hoped that the automation provided by the tool would allow us to construct more complex problems involving a larger number of reactions. However, whilst we can easily *construct* such satisfaction problems, the experimentation that the tool allowed us to perform convinced us that (at least with the current state of the art in ODE-SMT solvers), the actual *solving* of such problems is too slow for the method to be usefully applied to realistically sized design problems at the current time. Specifically, we had planned to synthesise circuits with various output responses to an input pulse, using as a CRN

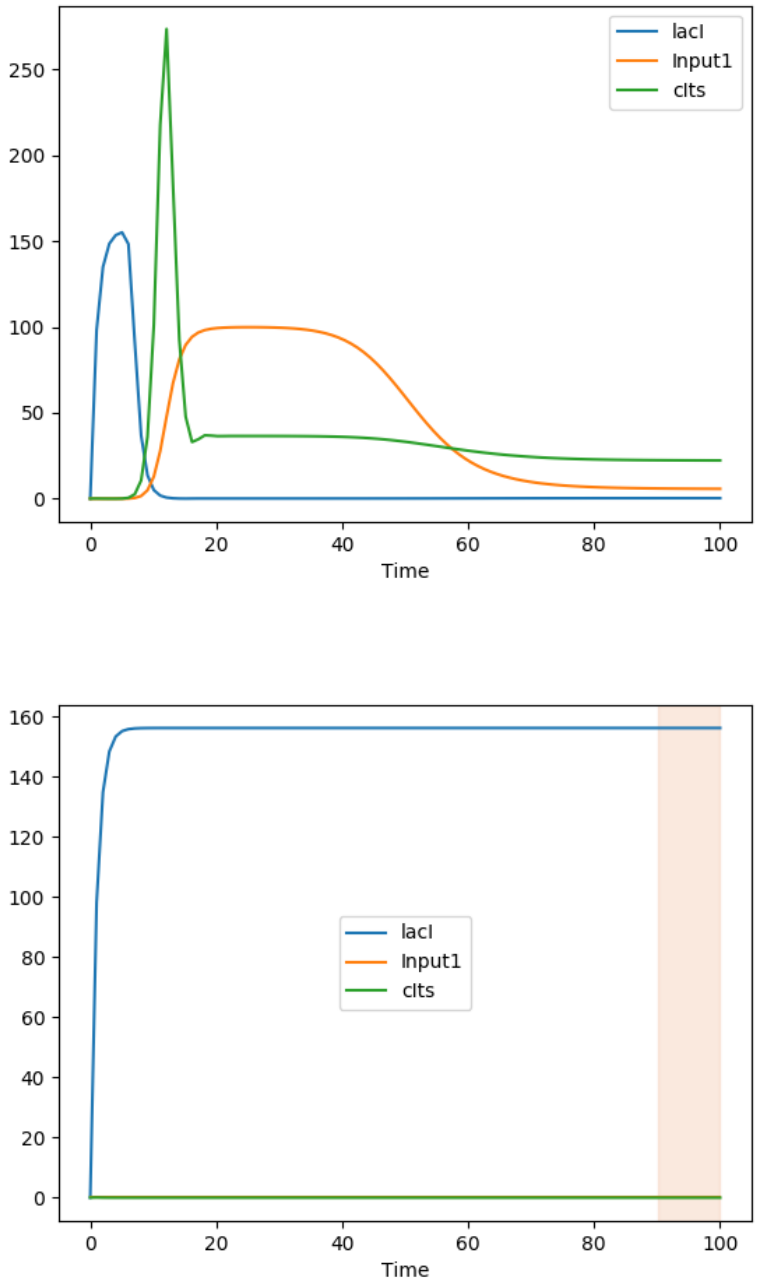


Figure 3.9: Simulation of CRN synthesized by `CRNSynth` for the toggle-switch example. In response to a pulse of input, $cIts$ rises above a threshold and remains above this threshold until $t = 100$, as required (top). If no input pulse is supplied, then $cIts$ remains low (bottom).

sketch a generic model of a three-protein signaling system [144][7], but found it too computationally intensive.

However, the solving of SAT modulo ODE problems is an active area of research, and solvers are expected to improve over time due to advances in a number of areas, including the implementation of methods for obtaining tighter enclosures for the solutions of ODEs, better heuristics for choosing which action to take at each step during the DPLL-$\mathcal{T}$ procedure, and additional pre-processing. The potential for improved pre-processing is clear from the observation that equivalent problems can have very different solution times. The solution of optimization and satisfaction problems is in general very difficult, and much of the progress that has recently been made has been achieved by developing specialised methods that work well on particular problem types that have been found to frequently arise in particular applications; the prospect for improvements in solving the problems formulated by `crnsynth` will depend on how similar they are to the problems motivating the development of SAT modulo ODE solvers.

The benefit of having written `crn-designer` is that it is now available for the evaluation of improved solvers as they are released. When solvers have advanced sufficiently, it will be ready to solve more complex CRN design problems

`CRNSynth` provides an example of how the `TimeRails` approach to representing specifications that was introduced in Chapter 2 can be integrated into an automated design tool in synthetic biology. An alternative design approach that also uses `TimeRails` is described in Chapter 4.

7 In this model, each of the three proteins could exist in either an active and inactive form. The active form of each could potentially catalyse the activation and/or inactivation of either of the other two species. Proteins also underwent basal activation and inactivation.

4

BAYESIAN DESIGN

4.1 MOTIVATION & CONTRIBUTION

Suppose that we are interested in a particular dynamical behaviour, such as oscillation. Given a specific model, how can we find sets of parameter values that will cause it to exhibit this behaviour? Or, given a set of models, how can we determine which joint choices of model and parameter values will achieve this behaviour?

These questions arise in several settings. When designing a synthetic circuit, a designer may wish to make a comparison between multiple candidate designs that rely on different mechanisms, or assess whether a modification to a system (such as the addition of a particular feedback loop) can improve the robustness[1] with which it meets the specification. It is possible to automatically construct models corresponding to a particular genetic construct [131], but this leaves the challenge of assigning values to the parameters. If the system has actually been constructed and characterised, parameter values may be estimated from experimental measurements; if no such data are available, parameters may be arbitrarily assigned values from some plausible range, but not all parameter assignments will exhibit the desired behaviour. An alternative approach is to select parameters from a plausible range such that the specification is satisfied; this can give an idea of how a functional circuit might work before actually trying to construct it. A related problem is to map from an abstract design to a concrete design that can actually be constructed, by choosing a well-characterized part from a library to assign to each abstract part in a design; for the final design to function as intended,

1 Robustness roughly corresponds to how much parameters can be varied from their nominal values before the desired behaviour is lost.

this assignment process must take into account the parameters corresponding to each part.

There is currently a lack of general tools for investigating these questions. This chapter describes a visual analytics system that allows a user to investigate which circuit designs, and choices of parameter values, are able to meet a specification. This is an integrated tool that allows a user to diagrammatically draw a specification, either upload a SBML file or diagrammatically draw a set of genetic networks, specify a prior distribution over the parameters, and visualize the posterior distribution of parameters that cause the model(s) to satisfy the specification.

Section 4.2 provides some theoretical background on topics specific to this chapter: quantitative interpretations of satisfaction for Signal Temporal Logic specifications (Section 4.2.1), and sampling methods based on ABC (Section 4.2.2). Section 4.3 describes the design and implementation of `Bayesian Designer`, the tool that is the subject of this chapter. Section 4.4 provides several examples of how `Bayesian Designer` can be applied. Section 4.5 compares the approach described in this chapter to the SMT-ODE approach taken in Chapter 3. Finally, 4.6 concludes the chapter. Appendix A provides some background information on probability.

4.2 INTRODUCTION

Previous studies have developed problem-specific computer programs that perform a brute-force search over parameters and network topologies to investigate which pairs exhibit particular properties (including bistability, oscillations, stripe-formation/'band-pass filtering', and perfect adaptation) [139] (Table 2.1). However, as this approach requires programming skills, it is not accessible to all potential users.

Other studies have employed more efficient sampling methods, such as ABC-Sequential Monte Carlo (SMC), as implemented by tools such as ABC-SysBio [95]. These are able to sample from the posterior distribution over parameters, conditioned on the fact that a system instantiated with those parameters performs in a way that is in some sense

close to what is desired (see Section 4.2.2 for a more complete discussion). However, this still requires the user to encode the specification as an objective function, implemented as a function in a programming language, that assigns a numerical score to a simulation trace.

Temporal logics (Section 2.2.1.2) provide a more general framework for descriptions of particular properties of systems over time, but tools for applying them to synthetic biology have been limited. Tools for designing circuits by decomposing such specifications and matching to a library of genetic parts are in development [166]. We present a tool that allows a user to graphically draw a specification in temporal logic, automatically constructs an objective function that computes the quantitative degree of satisfaction for this specification by a time-series, and uses this for parameter estimation and model/design comparison.

4.2.1 *Quantitative satisfaction of specifications*

The satisfaction of a specification can be interpreted as a Boolean that simply reports *whether or not* the specification was satisfied, but this discards quantitative information about the value of signals and time at which an event occurs. Information is lost when real-valued signals are Booleanized by predicates: for example, if only the Boolean value of a predicate such as $x > 12$ is recorded, the the precise value of x is lost, and there is no indication of how large a change in x would be needed to change the truth-value of $x > 12$.

There have therefore been several attempts to define measures of *how close* a signal is to satisfying a specification; these are typically signed, so that the sign indicates *whether or not* the specification was satisfied. Such measures are important, because they allow the application of optimization procedures that attempt to modify a signal to achieve satisfaction: these can be applied either to choose the signal that a model-predictive controller should supply to a system online, or to adjust the design of a system offline. They can also be used, as in `Bayesian Designer`, to determine whether a signal is within some distance threshold of satisfying a specification. For this purpose, the

important criteria for a measure of satisfaction is that it should not be too expensive to calculate, and that it should be usable in optimization to synthesize a system that meets the specification; based on these criteria, Bayesian Designer uses the space robustness estimate.

Space robustness

During his PhD research, Fainekos (supervised by George Pappas) introduced the concept of *robustness degree*, which is (roughly), a bound on the size of perturbation that the signal can tolerate without changing whether or not it satisfies a specification [40–42]. It is defined in terms of a metric d on the space of *signals*.

We define the unsigned distance from a point $x \in X$ to a set $S \subseteq X$ as:

$$\text{dist}_d(x, S) = \inf \{d(x, \hat{x}) : \hat{x} \in S\}$$

and the *signed distance* as

$$\text{Dist}_d(x, S) = \begin{cases} -\text{dist}_d(x, S) \text{ if } x \notin S \\ +\text{dist}_d(x, X \setminus S) \text{ if } x \in S \end{cases}$$

Denoting the set of signals that satisfy a specification ϕ as $\mathcal{L}(\phi)$, and the set of signals that do not satisfy ϕ as $\mathcal{L}(\neg\phi)$, the robustness degree of a signal s with respect to a specification ϕ is defined as $\text{Dist}_d(x, \mathcal{L}(\phi))$. This robustness degree is positive for signals that satisfy the specification, and negative for signals that violate it. For signals that satisfy the specification, it is the radius of the largest ball around the signal such that any signal contained within the ball satisfies the specification (i.e., $x \in \mathcal{L}(\phi)$ and $d(x, \hat{x}) \leq \text{Dist}_d(x, \mathcal{L}(\phi)) \implies \hat{x} \in \mathcal{L}(\phi)$).

Whilst it has the advantage of having a clear meaning, the robustness degree is difficult to calculate exactly. Fortunately, Fainekos proved that an easily calculated *robustness estimate* provides an underestimation of the robustness degree [42, Theorem 13][40, Theorem

3.1.1]. The robustness estimate at time t for a signal s with respect to a specification ϕ, $r(s, \phi, t)$, is evaluated recursively as follows[2]:

$$r(s, \top, t) = +\infty$$

$$r(s, g_\mu(s(t)) \geq c_\mu, t) = g_\mu(s(t)) - c_\mu$$
$$r(s, g_\mu(s(t) < c_\mu, t) = c_\mu - g_\mu(s(t))$$

$$\begin{aligned} r(s, \neg\phi, t) &= -r(s, \phi, t) \\ r(s, \phi_1 \wedge \phi_2, t) &= \min(r(s, \phi_1, t), r(s, \phi_2, t)) \\ r(s, \phi_1 \vee \phi_2, t) &= \max(r(s, \phi_1, t), r(s, \phi_2, t)) \\ r(s, \phi_1 U_{[a,b)} \phi_2, t) &= \sup_{t' \in [t+a, t+b]} (\min(\inf_{t'' \in [t, t')} r(s, \phi_1, t''), r(s, \phi_2, t'))) \end{aligned}$$

where $g_\mu : \mathbb{R}^n \to \mathbb{R}$ is any arbitrary function from the value of the signal at a particular time to a real number.

For brevity, $r(s, \phi, 0)$ can be written as $r(s, \phi)$.

Time robustness

A related question is how much a signal can be shifted in *time* rather than in *value* without changing whether it satisfies a specification. This question is answered by the *Time Robustness* introduced by Alexandre Donze (supervised by Oded Maler) in [34]. For an atomic proposition p, the left (θ^-) and right (θ^+) time robustness give the distance to the closest time-point (in each direction) at which the proposition changes value, and are defined as:

2 Different authors variously represent this function as $r(\bullet)$, $\rho(\bullet)$, or $[\![\bullet]\!]$. Some also refer to a signal trace as w rather than s.

$$\chi(\phi, s, t) = \begin{cases} +1 \text{ if signal } s \text{ satisfies specification } \phi \text{ at time } t \\ -1 \text{ if signal } s \text{ violates specification } \phi \text{ at time } t \end{cases}$$

$$\theta^{-}(s, p, t) = \chi(p, s, t) \cdot \max\{d \geq 0 \text{ s.t. } \forall t' \in [t-d, t], \chi(p, s, t') = \chi(p, s, t)\}$$
$$\theta^{+}(s, p, t) = \chi(p, s, t) \cdot \max\{d \geq 0 \text{ s.t. } \forall t' \in [t, t+d], \chi(p, s, t') = \chi(p, s, t)\}$$

Formulae ϕ including the operators $\neg, \vee, \wedge, U$ can be evaluated using the same recursive approach as for the space robustness estimate:

$$\begin{aligned} \theta^{\pm}(s, \neg\phi, t) &= -\theta^{\pm}(s, \phi, t) \\ \theta^{\pm}(s, \phi_1 \wedge \phi_2, t) &= \min(\theta^{\pm}(s, \phi_1, t), \theta^{\pm}(s, \phi_2, t)) \\ \theta^{\pm}(s, \phi_1 \vee \phi_2, t) &= \max(\theta^{\pm}(s, \phi_1, t), \theta^{\pm}(s, \phi_2, t)) \\ \theta^{\pm}(s, \phi_1 U_{[a,b)} \phi_2, t) &= \sup_{t' \in [t+a, t+b]} (\min(\inf_{t'' \in [t, t')} \theta^{\pm}(s, \phi_1, t''), \theta^{\pm}(s, \phi_2, t'))) \end{aligned}$$

The overall interpretation of the time robustness is that shifting a signal s to the right by less than $\theta^{-}(s, \phi, t)$ (or to the left by less than $\theta^{+}(s, \phi, t)$) will not change whether it satisfies the specification ϕ.

Satisfaction degree

Rather than asking how much a *signal* must be change to affect whether it satisfies a specification, we can instead ask how much the *specification* must change. This is the approach taken by Aurélien Rizk (supervised by François Fages) [125]. Given a trace and specification, they treat constants in the specification formula as variables, and form a constraint satisfaction problem to find the set of values for these such that the specification is satisfied by the trace (the 'validity domain'). They then calculate the distance between the validity domain and the values actually assigned to the variables in the specification.

Calculating this robustness measure is a more computationally intensive procedure, as it requires calculating the domain of validity. It is also somewhat less intuitive to interpret.

4.2.2 *Bayesian design*

In designing a system, we want to maximise the probability that it does what we want it do (as measured by satisfying a specification). We are thus interested in the distribution of candidate models and parameters θ, conditional on the output of the system satisfying the specification[3]:

$$p(\theta|\text{specification satisfied}, \mathcal{H}) = \frac{p(\text{specification satisfied}|\theta, \mathcal{H})p(\theta|\mathcal{H})}{p(\text{specification satisfied}|\mathcal{H})}$$
$$\propto p(\text{specification satisfied}|\theta, \mathcal{H})p(\theta|\mathcal{H})$$

This is analogous to the traditional problem of model comparison and parameter estimation, which tries to find the distribution of candidate models and parameters, conditional on the output of the system being consistent with experimental observations. Designing a system that is likely to behave in a particular desired way is essentially equivalent to fitting a model to observations of the way in which a system has behaved; this observation has been previously referred to as 'Bayesian Design' [7, 8, 160]. The candidate design topologies play the roles of models in a model selection procedure, and can be explored at the same time as parameters.

Having defined this distribution of interest, the question is how to sample from it. It is possible to sample from a simple distribution by drawing independent uniformly distributed samples and transforming appropriately (e.g., using the inverse of the Cumulative Distribution Function for a distribution where this exists in closed form, or using the Box-Muller transform to sample from a Gaussian distribution).

3 $\mathcal{H}$ represents the assumptions on which the probabilities are based; see Appendix A.

However, for sufficiently complicated distribution this is not possible, and so it is necessary to use Monte Carlo methods, such as Markov Chain Monte-Carlo (MCMC) [20, 100 📖, 111]. MCMC works by constructing a Markov Chain whose equilibrium distribution is the target distribution from which we are trying to sample, and then performing a random walk on this chain. However, models corresponding to biomolecular systems are typically too complicated to allow calculation of the likelihood function $p(\text{specification satisfied}|\theta, \mathcal{H})$, which restricts us to using likelihood-free methods (reviewed in [101, 142]). In these methods, *evaluation* of the likelihood function is replaced by performing *simulations*: we draw a choice of models and parameters from some proposal distribution, use these to perform a simulation, and accept them as a sample only if the simulated results are, in some appropriate sense, 'close enough' to the data. This results in sampling from the approximate posterior $p(\theta|d(x^*, x_0) \leq \epsilon)$, where ϵ is a parameter that determines the closeness of the approximation, and d is a distance function. The relationship between some Monte-Carlo algorithms and their likelihood-free variants is shown in Table 4.1.

ABC-SMC [151] is the ABC variant of Sequential Importance Sampling [106]. It is an iterative procedure, and uses a value of ϵ that decreases between iterations according to some schedule. It can be difficult to choose an appropriate ϵ schedule for a particular problem *a priori*, and one alternative is an automatic schedule in which ϵ is set to a very high value for the first iteration, and for subsequent iterations is assigned the value of the n'th percentile of the distances for all accepted particles (the optimal choice of n is still problem specific, but regardless of the value chosen the ϵ schedule will still be somewhat adaptive). This is the default adopted by `Bayesian Designer`, and applied to all of the examples in this chapter.

Note that whilst working in the framework of Bayesian statics naturally allows the consideration of stochastic models of biochemical systems, it can also be applied to deterministic models. In such a case, it still makes sense to talk about a posterior distribution, as a conditioning process has been applied to the parameters' prior distribution, and it is still meaningful to marginalise the joint posterior

Approach	Monte-Carlo Method	ABC Variant

Rejection Method	**for** $i = 1 \dots N$ **do**
	  repeat
	  Sample θ^* from $\pi(\theta)$
	  Sample u from $U(0,1)$
	  until $u < p(x_0\|\theta^*)$
	  $\theta^i = \theta^*$
		for $i = 1 \dots N$ **do**
		  repeat
		  Sample θ^* from $\pi(\theta)$
		  Sample x^* from $p(x\|\theta^*)$, by simulation
		  until $d(x_0, x^*) \leq \epsilon$
		  $\theta^i = \theta^*$

Metropolis-Hastings	Initialise $\theta^{(1)}$
	for $i = 1 \dots N$ **do**
	  Sample θ^* from the proposal distribution $Q(\theta^*\|\theta^{(i)})$
	  Sample u from $U(0,1)$
	  if $u < \min\left(1, \frac{P(x_0\|\theta^*)\pi(\theta^*)Q(\theta^*\|\theta^{(i)})}{P(x_0\|\theta^{(i)})\pi(\theta^{(i)})Q(\theta^{(i)}\|\theta^*)}\right)$ **then**
	  $\theta^{(i+1)} = \theta^*$
	  else
	  $\theta^{(i+1)} = \theta^{(i)}$
		Initialise $\theta^{(1)}$
		for $i = 1 \dots N$ **do**
		  repeat
		  Sample θ^* from the proposal distribution $Q(\theta^*\|\theta^{(i)})$
		  Sample x^* from $p(x\|\theta^*)$, by simulation
		  until $d(x_0, x^*) < \epsilon$
		  if $u < \min\left(1, \frac{\pi(\theta^*)Q(\theta^*\|\theta^{(i)})}{\pi(\theta^{(i)})Q(\theta^{(i)}\|\theta^*)}\right)$ **then**
		  $\theta^{(i+1)} = \theta^*$
		  else
		  $\theta^{(i+1)} = \theta^{(i)}$

Sequential Importance Sampling (SIS)	**for** $T = 0 \dots T$ **do**
	  for $i = 0 \dots N$ **do**
	  repeat
	  if $t = 0$ **then**
	  sample θ^{**} independently from $\pi(\theta)$
	  else
	  sample θ^{**} from previous population θ^i_{t-1} with weights w_{t-1}
	  obtain θ^{**} by perturbing with kernel K_t
	  until $\pi(\theta^{**}) > 0$
	  Set $\theta^{(i)}_t = \theta^{**}$
	  if $t = 0$ **then**
	  $w^{(i)}_t = 1$
	  else
	  $w^{(i)}_t = \frac{P(x_0\|\theta^{(i)}_t)\pi(\theta^{(i)}_t)}{\Sigma_{j=1}^N w^{(j)}_{t-1} K_t(\theta^{(i)}_t\|\theta^{(j)}_{t-1})}$
		for $T = 0 \dots T$ **do**
		  for $i = 0 \dots N$ **do**
		  repeat
		  repeat
		  if $t = 0$ **then**
		  sample θ^{**} independently from $\pi(\theta)$
		  else
		  sample θ^{**} from previous population θ^i_{t-1} with weights w_{t-1}
		  obtain θ^{**} by perturbing with kernel K_t
		  until $\pi(\theta^{**}) > 0$
		  Sample x^* from $p(x\|\theta^*)$, by simulation
		  until $d(x_0, x^*) \leq \epsilon$
		  Set $\theta^{(i)}_t = \theta^{**}$
		  if $t = 0$ **then**
		  $w^{(i)}_t = 1$
		  else
		  $w^{(i)}_t = \frac{\pi(\theta^{(i)}_t)}{\Sigma_{j=1}^N w^{(j)}_{t-1} K_t(\theta^{(i)}_t\|\theta^{(j)}_{t-1})}$

Table 4.1: Some Monte-Carlo sampling schemes and their Likelihood-Free ABC variants. If we can evaluate the likelihood $p(x_0|\theta)$ for given values of x and θ, then we can use the methods on the left to draw N samples from the exact posterior $p(\theta|x_0)$. If we cannot do this, then we rely on drawing a sample from $p(x|\theta^*)$ by performing a simulation, and accept θ^* only if this is sufficiently close to x_0. The methods on the right use this to drawn N samples from the approximate posterior $p(\theta|d(x^*, x_0) \leq \epsilon)$, where ϵ is a parameter that determines the closeness of the approximation, and d is a distance function.

(for example, to find the probability of a circuit meeting the specification as the value taken by a particular parameter varies, assuming the values of the other parameters are picked randomly).

4.2.3 *GPU accelerated simulations*

The Graphics Processing Unit was initially developed to quickly render graphics by performing calculations in parallel. The GeForce 3 was the first Graphical Processing Unit (GPU) capable of running short programs called shaders in parallel; each would process either an individual pixel in the final image or a single vertex in a 3D model being rendered. This led to attempts to run shaders that were intended to perform a computation other than producing an image.

In 2007, Nvidia released Complete Unified Device Architecture (CUDA), an interface that allowed programmers to write general purpose programs that would run on the GPU using an extension to the C programming language, without having to think in terms of graphics primitives. This has become a hugely popular way to run a range of computationally intensive tasks, and provides Nvidia with a major competitive advantage over its rival AMD (whose GPUs are not compatible with CUDA, but are compatible with the similar OpenCL standard). Nvidia now even produces 'GPU' cards with no attached video connectors (e.g., the Tesla series).

Using a GPU allows a very high degree of parallelism for a relatively low cost (though GPU prices are currently being distorted due to high demand for use in mining cryptocurrency and performing machine learning). An inexpensive desktop GPU may have 640 cores; the Tesla V100 card has 5120 CUDA cores.

`cuLsoda` is an ODE solver created by translating the `LSODA`[4] solver from Fortran into CUDA-using C. LSODA begins using a non-stiff solver (Adam's method), and automatically switches between using this and a stiff solver (Backwards Difference Formula) [63, 121]; this switching frees the user form the responsibility of deciding whether to use a stiff or non-stiff solver for their problem, and also allows the

4 The **L**ivermore **S**olver for **OD**Es with **A**utomatic method selection.

use of more than one solver type for a single problem (e.g., initially using a more efficient non-stiff solver, and then switching to a stiff solver after an initial transient has passed).

`PyCUDA` [84] provides an Python interface to the CUDA API that makes it easy to compile a string of C code into a CUDA kernel that can be loaded onto a GPU, transfer arrays of data to the GPU device, call the kernel to process this data, and copy the results back into the main memory of the host computer, whilst automatically handing memory deallocation and cleanup. The ability to create a kernel from a string of code makes it possible to define a kernel using Python code that automatically generates C code.

`cuda-sim` [164] is a tool that uses the GPU to perform many simulations in parallel; solving the same set of ODEs with different values for the parameters and initial conditions is an embarrassingly parallel problem for which the GPU is ideally suited. One inconvenience, however, is that if GPU performing the simulations is also used to drive a graphical desktop, then the compute kernels will be killed by a watchdog timer after a few seconds, which is typically too short a time for the simulation to complete.

`cuda-sim` can parse an SBML file, and use this to generate CUDA code for functions that will calculate the derivative of the system with respect to time and have an interface compatible with `cuLsoda`; `PyCUDA` is used to call `cuLsoda` to simulate the system. As an alternative to these deterministic simulations, `cuda-sim` is also able to perform simulations using the Gillespie method, or numerically integrate the Chemical Langevin equation using the Euler–Maruyama method (see Section 1.3 for discussion of these methods).

`abc-sysbio` calls `cuda-sim` to perform simulations, and uses these to carry out parameter inference and model comparison using ABC-SMC [96].

4.3 BAYESIAN DESIGNER

Bayesian Designer is a tool for visualising sets of parameters sampled using ABC-SMC. A major novel feature is the ability to sample

parameters *conditional on satisfying a specification* expressed in Signal Temporal Logic as drawn using `TimeRails`. This is achieved by using the space robustness measure $r(x^*, \phi)$ in place of the distance function $d(x_0, x^*)$. Thus instead of accepting a sampled parameter vector if the result x^* of a simulation using it is sufficiently close to the results x_0 of an experiment, we accept it if the simulation result is sufficiently close to satisfying a specification ϕ. However, Bayesian Designer can also be used to visualise the results of fitting a model to experimental measurements, rather than a specification.

4.3.1 *Graphical interface*

The interface for setting up a problem in Bayesian Designer is shown in Figure 4.1, and the interface for viewing the results is shown in Figure 4.2.

4.3.1.1 *Representation of circuit topology*

A user can either upload one or more SBML models corresponding to models of circuits that they wish to parametrise, or draw a circuit topology using the graphical CRN editing widget that was created for CRN Designer (Chapter 3).

4.3.1.2 *Representation of specification*

Signal Temporal Logic specifications are represented diagrammatically using the `TimeRails` approach 2.

This represents a constraint on the values that a variable may take using a rectangle; for the specification to be satisfied by the system, the variable's value must remain within each rectangle at all times between its start and end times. The Globally and Finally operators are represented by rails along which rectangles or other rails can slide. The graphical components (rectangles and rails) compose in the same way as the corresponding symbols do when the specification is represented as a formula.

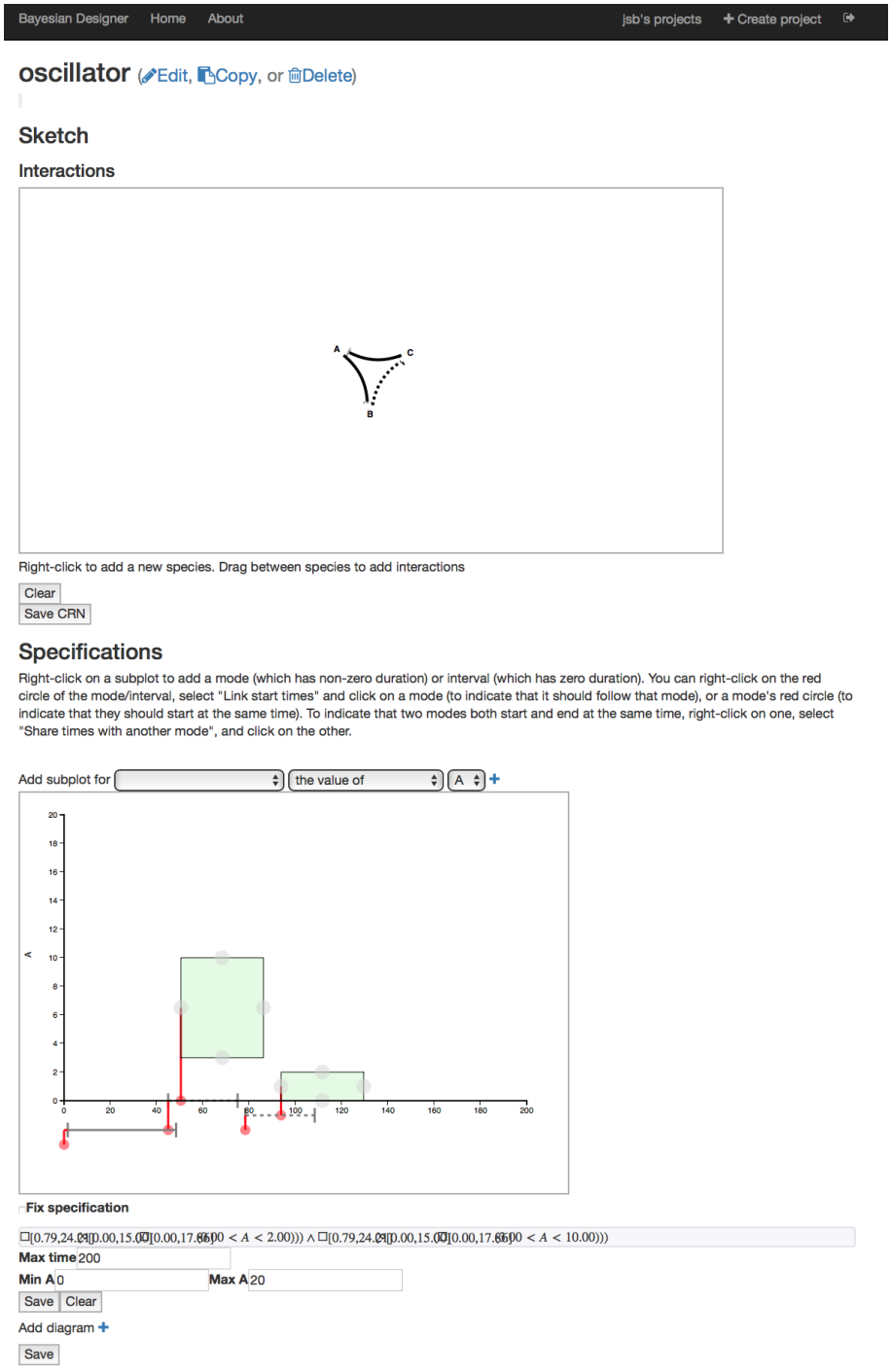


Figure 4.1: The graphical interface for setting up a problem in Bayesian Designer.

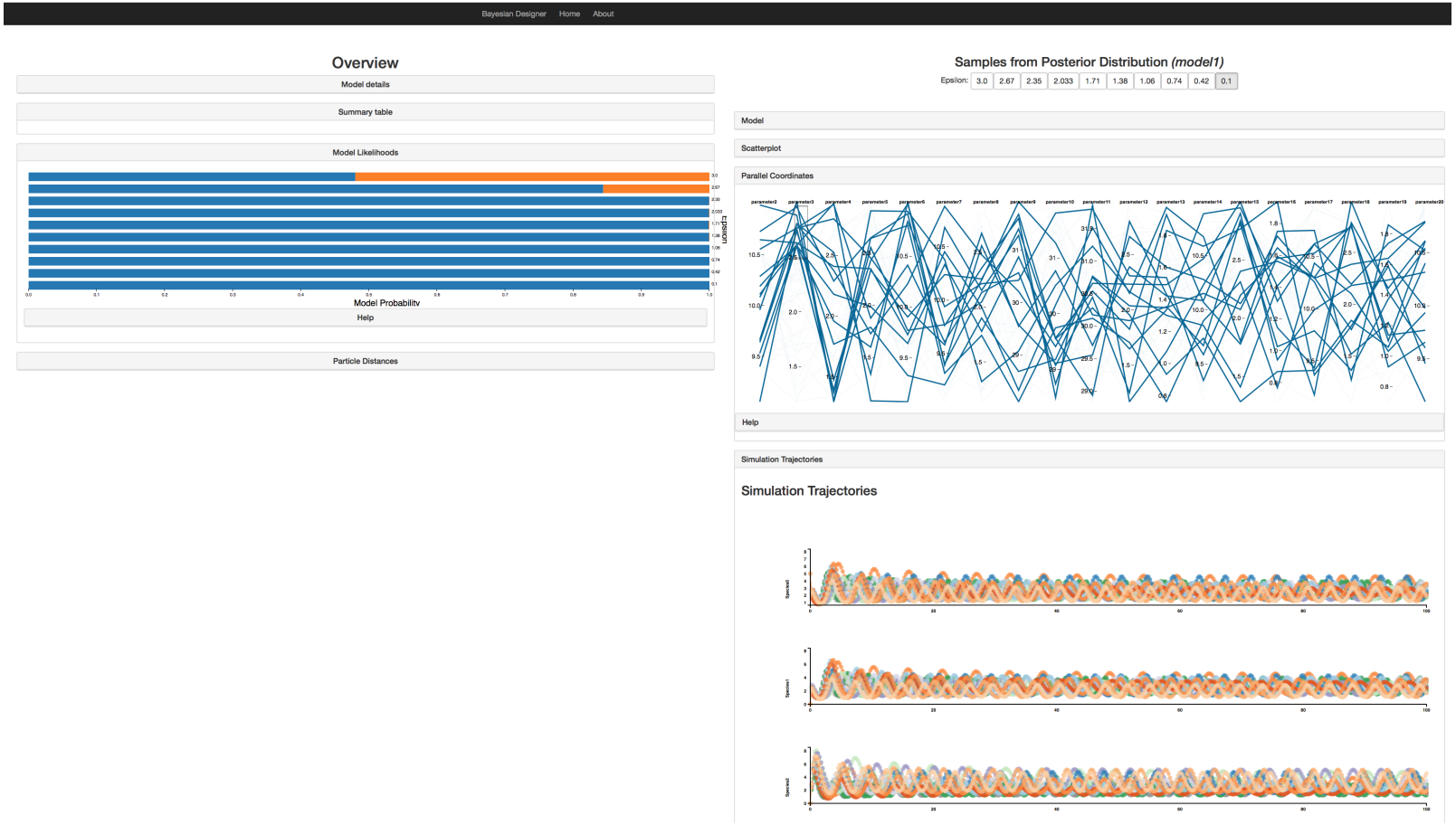


Figure 4.2: The graphical interface that presents the results of inference in Bayesian Designer.

The Finally operator indicates uncertainty in the time at which a proposition is true; this is represented diagrammatically by a dashed rail, and for a trajectory to satisfy the specification it must be possible to slide along the rail into a position where the trajectory remains inside the rectangle. The Globally operator represents a set of constraints; it indicates that a proposition is true for every time in a range. This is represented diagrammatically by a solid rail, and for a trajectory to satisfy the specification the trajectory must remain inside the rectangle for every position along the rail.

4.3.2 *Representation of results*

Monte Carlo sampling methods provide not just a single parameter set, but a set of samples that approximate the full posterior distribution, providing additional information such as what ranges of parameters can meet the specification, and what correlations exist between parameters. We want to find not only a single parameter set consistent with a specification or experimental measurements, but rather to understand the characteristics of all such parameters sets, as well as reassuring ourselves that the method has worked. Whereas for `TimeRails` (Chapter 2) we introduce a new visual abstraction to represent a set of objects, here we combine familiar idioms of statistical graphics into a series of interactive linked views.

Cohesion between the separate views is achieved by linking highlighting between plots, and applying colors consistently to each of the separate plots.

An earlier form of our work on visualizing ABC-SMC results was described previously [137]. In our updated viewer, we represent the results in two panes (Figure 4.2): each pane contains a number of panels, which can be collapsed to temporarily hide its contents, allowing attention to be focused on the plots of interest at the current time. These panels include the following visualizations:

1. A table of acceptance probabilities and simulation times.

2. A line graph showing the error calculated for each particle.

3. A stacked bar chart showing the model likelihood estimated after each generation.

4. A diagram of the model's structure, or diagram comparing the models, produced by `sbml-diff` (Chapter 5).

5. A parallel coordinates plot and a matrix of scatterplots showing the parameter values corresponding to each particle.

6. A line graph showing the time-series of each simulation.

7. An estimate of the posterior distribution of each parameter (Section 4.3.2.2).

The questions that a user may ask include:

- **How rapidly has the error/distance for the population of particles decreased between generations? Was the epsilon schedule used good?** This is addressed by plot 2, which shows the error for each particle in each generation, and how close this was to the acceptance threshold. This will show whether most particles had an error far less than the threshold (in which case a more rapidly decreasing ϵ schedule could have been used), or if particles were only just meeting the threshold (in which case a more gradually decreasing ϵ schedule might work better). Similar information is also provided by plot 1, in a more summary form.

- **What is the final estimate of the posterior distribution? Specifically, what is the posterior distribution for each specific parameter, and are there any apparent correlations between parameter? Which parameters is the system most sensitive to/which are most highly constrained? How much did the posterior change between generations, and does it seem to have converged?** These questions can be answered using the parallel coordinates and ScatterPLOt Matrix (SPLOM) (plot 5). The SPLOM is effective in showing any pairwise correlations between parameters. However, it is difficult to visually connect the points representing the same particle in different subplots: in contrast,

the parallel coordinates plot makes it easy to see the values of all the parameters associated with a single particle. The posterior distribution of each individual parameter is shown directly by the density estimates in plot 7; this has the advantage of weighting each sample appropriately to show the estimated posterior, rather than the distribution of samples.

- **Which model is most consistent with the specification/observations? Has the method converged on an answer to this question? How do the posterior distributions differ between parameters?** This is addressed by plot 3; as this shows a bar for each generation, it is possible to see directly whether the estimated model likelihood for each model has converged.

- **What do the actual simulated trajectories corresponding to each set of parameters look like? Are they reasonable, or do they reveal an obvious error in the simulations (e.g., concentrations becoming negative or conservation relations being violated)? How does their shape vary qualitatively as parameter change?** This is addressed by plot 6, which can be filtered by brushing either the parallel coordinates plot or SPLOM.

4.3.2.1 *Interactivity*

A dedicated interactive viewer for inference results is useful both because the results are multidimensional and because they have a complex structure: producing diagnostic or exploratory plots is therefore a slightly tricky process requiring programming skill. There are a number of questions to address, each requiring a different plot to answer; whilst a range of static plots could be generated automatically, static plots have limitations that can be addressed by interactivity. We make use of a number of forms of interactivity: linking and brushing, interactive adjustment of parameters (e.g., kernel bandwidths, scales for axes), discrete filtering (e.g., selection of a specific model and generation number/ϵ threshold), and the re-ordering of axes in parallel coordinates plots.

4.3.2.2 *Density estimate of posterior distribution*

In addition to plotting the parameter values corresponding to the population of particles at a particular generation, we can plot the estimate of parameter values that would be obtained from them. This depends not only on the parameter values, but also the corresponding weights (as defined in the bottom-right entry of Table 4.1): the posterior distribution can be approximated as

$$\tilde{\pi}_N(x) = \frac{1}{N}\sum_{i=1}^{N} w(X^{(i)})\delta_{X^{(i)}}(x)$$

where $X^{(i)}$ is the vector of parameters corresponding to the i'th particle, $w(X^i)$ is the weight of this particle, and $\delta_{X^{(i)}}$ is the Dirac delta-function centered on $X^{(i)}$ [150].

This probability distribution function could be represented as a histogram, but it seems most natural to instead apply a kernel density estimation representation: a histogram imposes arbitrary rectangles that have no direct meaning (unlike the kernel functions, which each correspond to a particle or sample), and requires a choice of bin width. The Dirac delta function can be approximated by a Gaussian: it is the limit of the sequence of zero-centered Gaussians

$$\delta_a(x) = \lim_{a\to 0} \frac{1}{|a|\sqrt{\pi}} e^{-(x/a)^2}$$

It can also be considered as the limiting case of a rectangular or triangular distribution, or as $\lim_{\Omega\to\infty}\left(\frac{\sin \Omega t}{\pi t}\right)$ [124 📖, p.443], but the Gaussian approximation has the advantage of being both smooth and non-oscillatory.

In order to calculate an estimate of the density that can be plotted, a specific value of a must be chosen. There are several general approaches to handling arbitrary parameters in visualization:

- picking *a single value* by applying some heuristic or optimizing some quality metric
- picking *a small number of discrete values*, and representing each using small multiples

- simultaneously representing the results of *all possible values* (e.g., a mode tree [104], or barcode-tree [2])
- picking a single value, but allowing the user to *interactively adjust* it

We adopt the final one of the approaches, and allow the user to adjust a: the larger the value that is selected, the smoother the resulting graph appears.

4.3.3 *Implementation*

The frontend of `Bayesian Designer` is implemented using JavaScript and the D3 library [17]. The `d3.parcoords.js` component[5] is used to draw the parallel coordinates plot. The backend is written in Python using the Flask framework.

The backend receives a serialised representation of a TimeRails diagram, and uses this to generate a Python function that evaluates the robustness estimate for a signal with respect to that specification. It uses the details of the prior distributions chosen by the user to generate a configuration file for `abc-sysbio`, then calls `abc-sysbio`, specifying the Python code that it has generated should be used as a custom distance function. It then serves the output files generated by `abc-sysbio` to the frontend, where they are processed and visualised.

4.4 EXAMPLE APPLICATIONS

This section shows how `Bayesian Designer` can be applied to a series of increasingly complicated problems. Sections 4.4.1 investigates a single system with no inputs. Section 4.4.2 compares two candidate systems with no inputs. Section 4.4.3 considers the problem of trying to find parameters such that a system's output satisfies a specification when a particular input profile is provided. Several of the behaviours investigated have been previously characterized using *ad hoc* specifications listed in Table 2.1.

5 https://syntagmatic.github.io/parallel-coordinates/

All experiments were performed on a Dell OptiPlex 9020 desktop with 16 GB of RAM, a 3.6GHz Intel Core i7-4790 processor, and a 2 GB GeForce GTX 750 Ti graphics card, running Linux Ubuntu 16.0.4 LTS.

4.4.1 *Bell-shape generator*

Problem (based on [23, Example 1]): *Given the CRN*

$$A + K \xrightarrow{k_1} 2K$$
$$B + K \xrightarrow{k_2} 2B$$

identify sets of initial conditions and parameters such that the evolution of K has a bell-shaped profile (i.e., during an initial interval the concentration of K increases up to a maximum, and then decreases).

We can represent this problem using a specification that is the conjunction of three terms: a *Globally* term requiring that K be below a threshold for some initial period, a *Finally Globally* term requiring that K should remain in some range for a period of time beginning at some time in a range, and a *Globally* term requiring that K be below a threshold for some final period.

To make the specification precise, we require that $K < 0.2$ for the initial 10 and final 23 seconds of the simulation, and $0.6 \leq K \leq 1.5$ for a period of 20 seconds beginning at some time-point in the initial 53 seconds. This can be represented diagrammatically in `TimeRails` as shown in Figure 4.3.

The time taken by simulations at each generation is shown in Table 4.2. At each generation, simulations continue to be performed until 100 particles (parameter sets) have been identified whose distance from meeting the specification is less than ϵ. As ϵ decreases between successive generations, this condition becomes more stringent and so an increased number of simulations is required. However, because many simulations are run in parallel on the GPU, the running time does not increase continuously but in discrete jumps of $\approx$ 180 s (=3

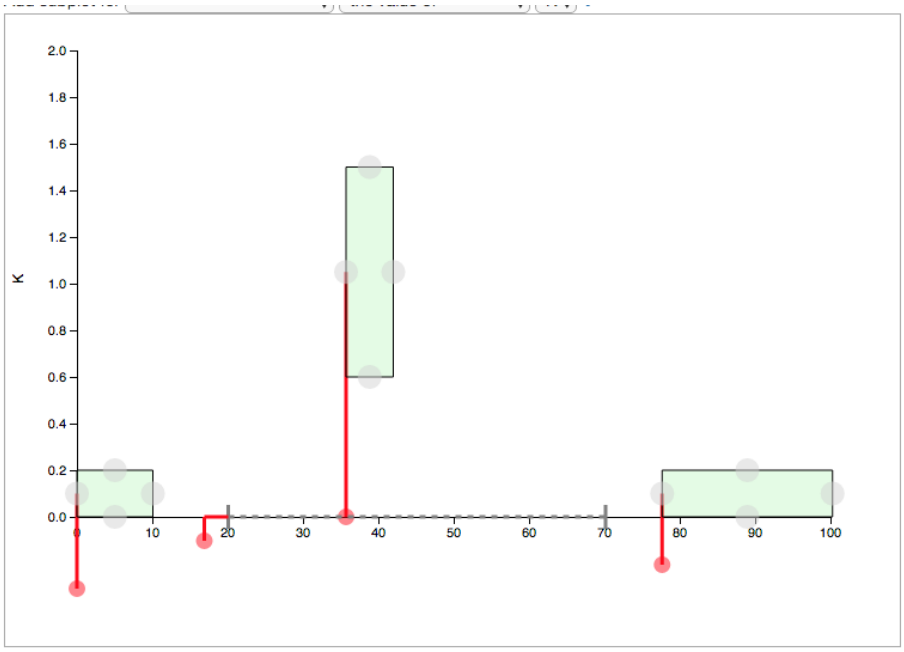


Figure 4.3: The specification for Bell-shape example, as represented using `TimeRails` in `Bayesian Designer`.

minutes): in this example, each generation requires 1, 2, or 3 batches of simulations.

The total run time is approximately 36 minutes.

ϵ	Simulations	Acceptance Rate	Simulation Time/s
10000000000.0	100	1.0	179.87
0.5944	1,052	0.0951	180.56
0.3891	1,808	0.0553	181.59
0.2273	3,934	0.0254	181.05
0.157	13,841	0.00722	181.59
0.0931	30,395	0.00329	363.28
0.0405	62,241	0.00161	546.13
0.01	49,608	0.00202	362.7

Table 4.2: Running time and acceptance rates for the Bell-shape Bayesian Designer example.

At the first generation, ϵ is set to a high value so that all particles are accepted. The parallel coordinates plot for this generation shows that k_1 and k_2 are being uniformly sampled (Figure 4.4). The corresponding trajectories for K show a range of shapes: some are bell-shaped as desired (though most of these are not consistent with the numerical values included in the specification), but others show an asymptotic increase towards an equilibrium value (Figure 4.4).

In the final generation, the trajectories are consistent with the spec-

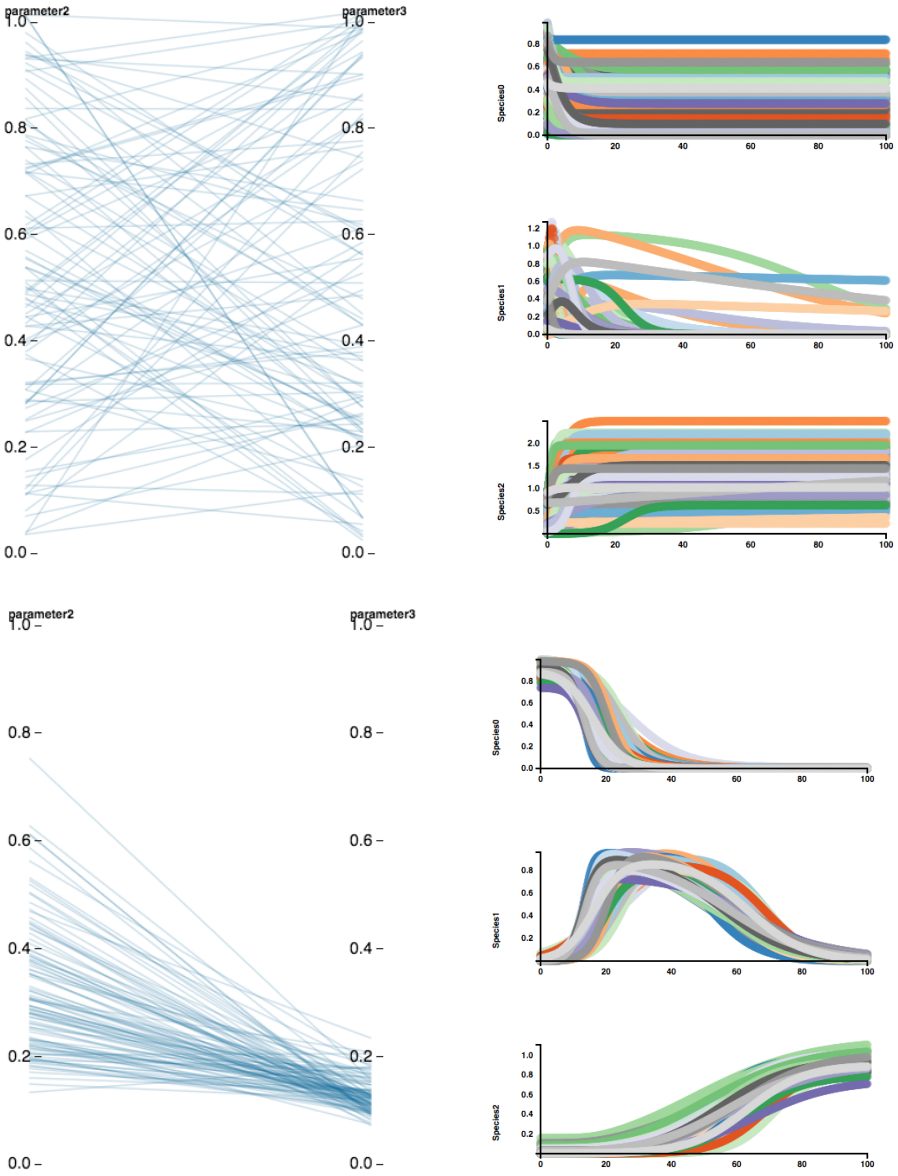


Figure 4.4: Parallel coordinates plot showing sampled parameter values (left), and corresponding simulated trajectories of concentrations (right), for the **first** and **last** generations of the Bell-shape example.

This problem is related to that studied by [160], but we use specifications of a very different form and (in the interest of computational time) consider only two circuits rather than the full set of 9,963 circuit topologies.

The model being considered is the repressilator shown in Figure 5.3, and the specification used to represent oscillations is shown in Figure 4.5. Translation and the degradation of mRNA and protein molecules

are modelled as first-order reactions; transcription is modelled as a reaction with Hill repression kinetics.

The specification requires that for all times in a certain range, the output will both become high (and remain high for a period) and become low (and remain low for a period) within a certain time. This requires the output to keep switching between the high and low ranges, and hence oscillate.

The results are shown in Figure 4.6. As in the previous example, the parameters sampled in the first iteration are uniformly distributed and the systems instantiated with them are not close to having the desired dynamics. However, by the final iteration the values of some parameters appear to be quite constrained and all of the sampled parameters result in meeting the specification. In particular, it is very noticeable that the parameters labeled as 2, 6, 7[6] (the protein degradation rates), and 13, 16, 19 (the mRNA degradation rates) are required to be small, whereas those labelled 14, 17, 20 (the maximum translation rates) are required to be large.

4.4.3 *Toggle switch*

Problem: *Parametrise the 'Mutual activation' switch from [154, Box 1][7]:*

$$
\begin{aligned}
G(u, v, J, K) &= \frac{2uK}{(v - u + vJ + uK) + \sqrt{(v - u + vJ + uK)^2 - 4(v - u)uK}} \\
\frac{dR}{dt} &= k_0 E_P(R) + k_1 S - k_2 R \\
&= k_0 G(k_3 R, k_4, J_3, J_4) + k_1 S - k_2 R
\end{aligned}
$$

The specification for this switching behaviour is represented in Figure 4.7. This expresses the input as a pulse formed from the sum of two sigmoidal functions: a sigmoid that increases from 0 to 1.25 with a midpoint at $t = 30$, and a sigmoid that decreases from 0 to -1.25 with a midpoint at $t = 55$. The specification expresses what output

6 Note that the plotted parameters are labelled starting at 2, as the first parameter is the cell volume, which is constant and so not plotted.

7 Note that [154] erroneously prints the final term of $\frac{dR}{dt}$ as $-k_2 XR$ rather than $-k_2 R$.

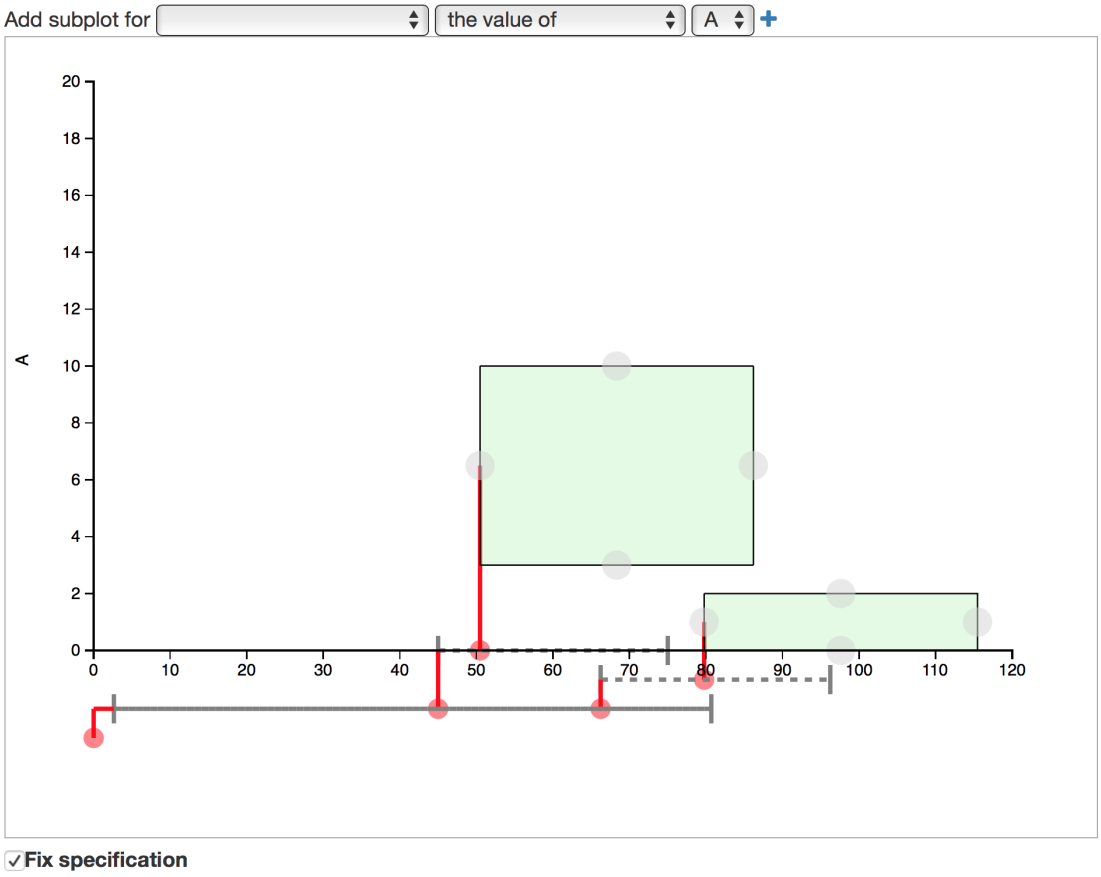


Figure 4.5: The specification for the oscillator example, as represented using `TimeRails` in `Bayesian Designer`.

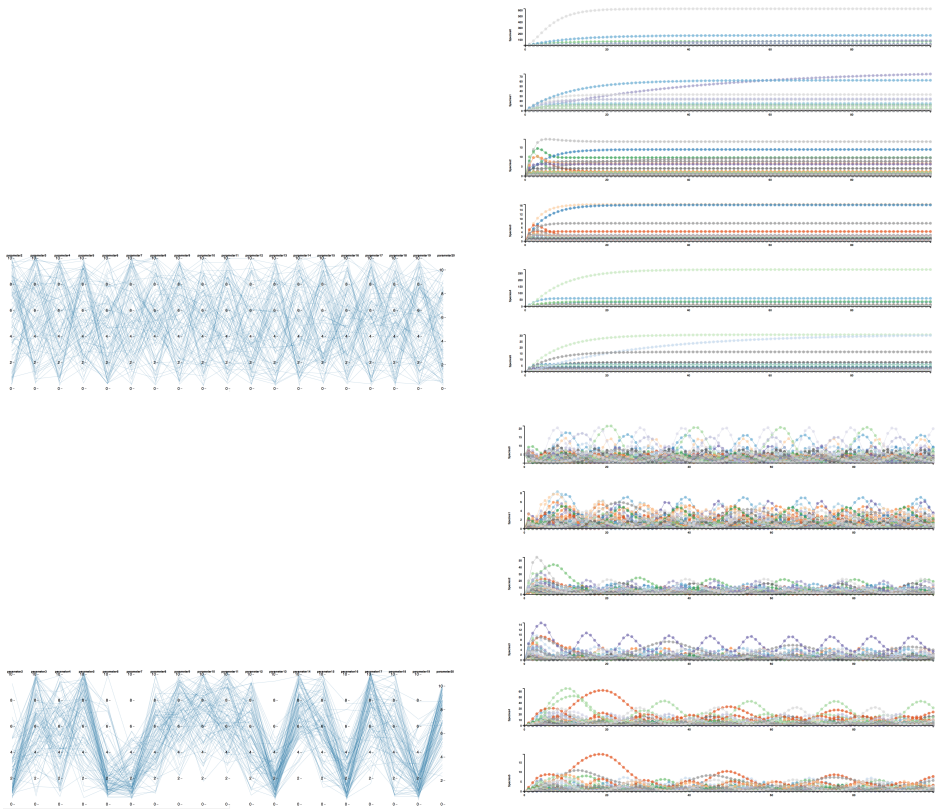


Figure 4.6: Parallel coordinates plot showing sampled parameter values (left), and corresponding simulated trajectories of concentrations (right), for the **first** and **last** generations of the oscillator example.

trajectories are acceptable, given this input, as a conjunction of three *Globally* terms: the output is required to be initially low (the range of permitted positions for the 'off' equilibrium), is permitted to take any value in a wide range (allowing a transient peak that is higher that the 'on' equilibrium), and then to be in an range corresponding to the permitted positions for the 'on' equilibrium.

We can incrementally 'tighten up' the specification and observe the effect that this has on the parameters (Figure 4.8). We start with a relatively loose specification, in which the 'on' equilibrium value is permitted to take any value in the range $0.2 \leq R \leq 0.5$. Narrowing this to $0.3 \leq R \leq 0.4$ has a clear effect: most visibly p_5 (k_3) is required to be high and p_6 (k_4) low, but p_2, p_3, p_7 (k_0, k_1, J_3) all decrease and p_4 (k_2) increases. Reducing the amount of over-shoot that is permitted whilst the input is supplied causes a further decrease in p_3 and p_7 (k_1 and J_3).

Insight into the effect of each parameter on the system's dynamics can also be gained by observed by brushing particular regions of individual axes in the parallel coordinates plot. However, the information provided using the two interaction types is different: filtering can only reveal patterns that are present in the (limited number of) sampled particles, whereas re-simulating after adjusting a specification draws additional samples.

4.5 COMPARISON WITH `crn-designer`

Two systems for synthesizing a CRN based on specifications have been described in this thesis: one based on SMT-ODE (Chapter 3) and one based on ABC-SMC (this chapter). Both try to find parameters that result in a specification being met, but there are important differences between them.

Compared to randomly sampling space, both methods are an improvement, because they try to exploit information from previous ODE evaluations about which parameter values to use in the next simulations.

Generality

The structure of SMT-ODE solvers restricts them to use for bounded model-checking problems with specifications that can be expressed as a finite number of modes. The approach adopted by Bayesian Designer is more general, and allows the use of a wider family of specifications.

Information provided

Both `CRN Designer` and `Bayesian Designer` can be used to synthesize a single system that meets a specification. However, they each also have additional capabilities. As `CRN Designer` works by pruning away only regions of parameter space that cannot contain any so-

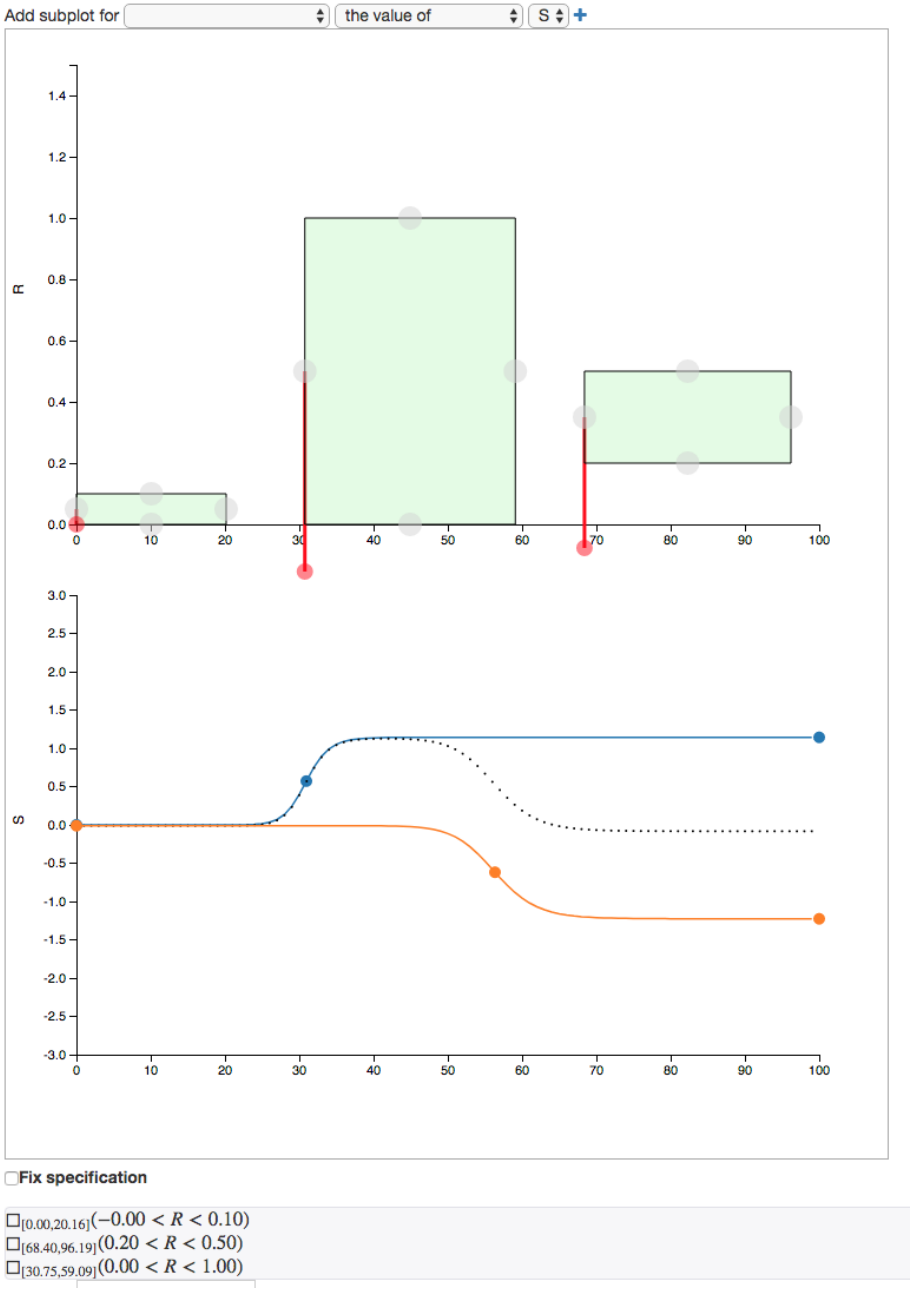


Figure 4.7: The specification for switch example, as represented using `TimeRails` in `Bayesian Designer`.

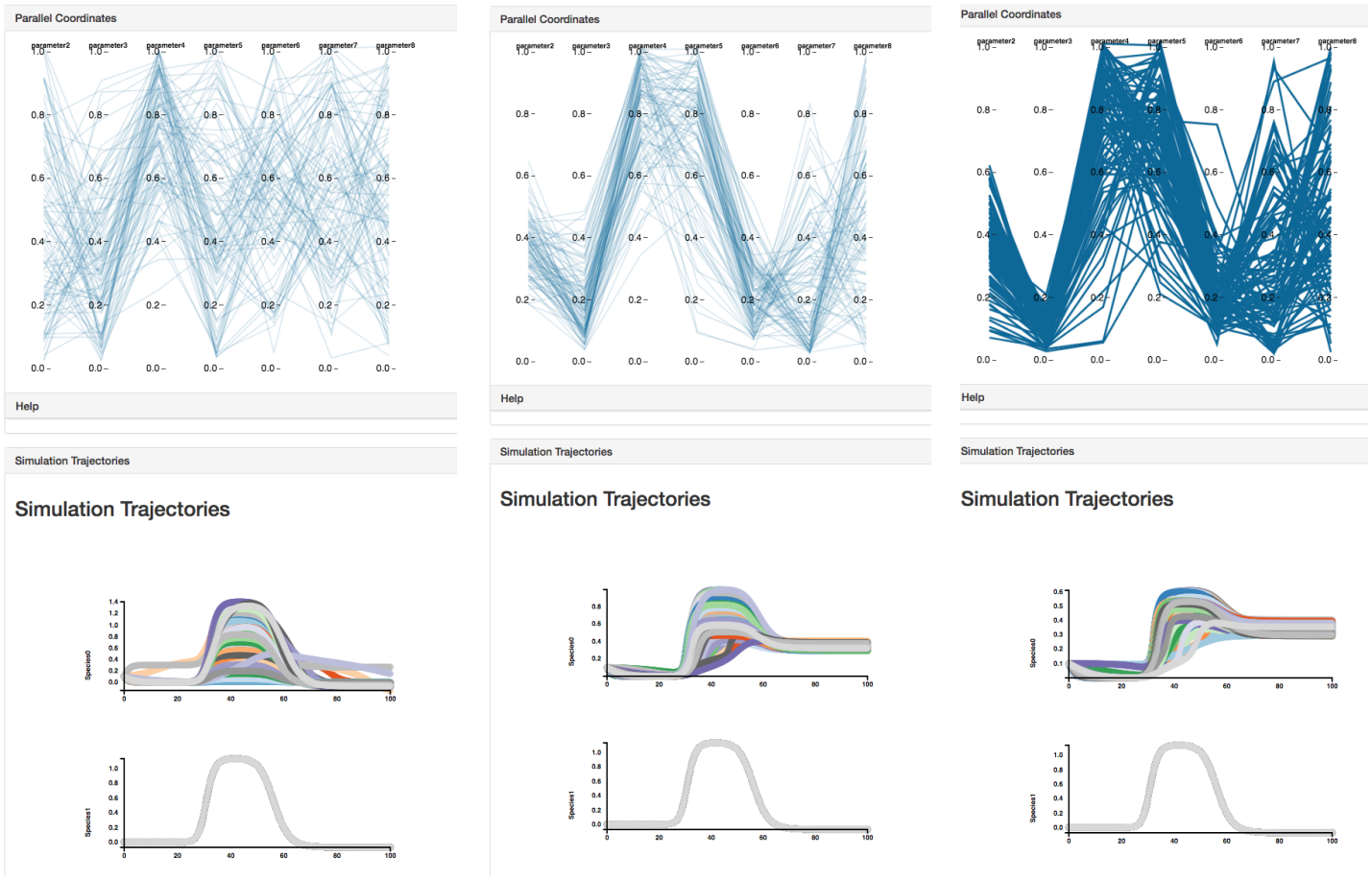


Figure 4.8: Sampled parameters (top) and simulated trajectories (bottom) for three increasingly strict specifications for the switch. **Left**: the 'on' equilibrium value is permitted to take any value in the range $0.2 \leq R \leq 0.5$. **Middle**: this range is reduced to $0.3 \leq R \leq 0.4$. This has a clear effect on the posterior distribution of the parameters: most visibly p_5 (k_3) is required to be high and p_6 (k_4) low, but p_2, p_3, p_7 (k_0, k_1, J_3) all decrease and p_4 (k_2) increases. **Right**: reducing the amount of over-shoot that is permitted whilst the input is supplied causes a further decrease in p_3 and p_7 (k_1 and J_3).

lutions, it can be used to prove the non-existence of solutions in a particular region. As `Bayesian Designer` works by sampling sets of parameters that result in the specification being satisfied, it can be used to gain insight about the region of parameter space that gives rise to a particular behavior: it can reveal how robust or sensitive specification satisfaction is to the value of each parameter, and can reveal correlation structure between different parameters.

Parallelism

Existing SMT-ODE solvers such as iSAT and dReach use only a single thread. This is understandable, as DPLL (Table 3.1) is essentially a sophisticated depth-first search. In principle some parallelism may be achievable by splitting the initial parameter search space into orthants, and using each of these to separately initialise an SMT-ODE solver. However, this would be imperfect as no information would be exchanged between the different solvers, but a deduction made by one solver might also be useful to prune an interval in a region that was assigned to another solver.

In contrast, `Bayesian Designer` is already highly parallelised across the many cores of a GPU (or even across multiple GPUs). It can thus perform far more ODE evaluations in the same time as `CRN Designer`.

Interactive monitoring

Regardless of the degree of parallelism employed, non-trivial synthesis problems will take (at least) several minutes to solve. It is therefore desirable for the user to be able to interactively monitor progress, so that they can identify whether they have mis-specified their problem without waiting for synthesis to complete, decide whether the solver is making progress and should be left to continue (or has become stuck and should be terminated), and be less impatient or anxious about the progress of the solver.

It is easier to interactively monitor the progress of `Bayesian Designer`: after each round of simulations, it is possible to visualise progress by plotting the trajectory of each simulation, as well as the distribution of distances from meeting the specification. It is not possible to exactly predict what will happen after the next iteration, but the user can try to get a rough idea by extrapolating based on the amount of progress that was made by the previous iteration.

If the SMT solver was called using a sufficiently high verbosity setting then the output could in principle be extracted and plotted as it ran. However, a major challenge is that it may at any point become necessary to backtrack, which would require decisions/guesses to be forgotten, enlarging the candidate solution box. This means that the volume of the candidate solution box (or the width of the interval for any particular variable) does not necessarily decrease monotonically, making it hard to judge how much progress towards finding a solution has actually been made.

Instantiation of selected parameters

However, going from a desired set of numerical values for the parameters in a model to a specific DNA sequence that will achieve these is currently a major unsolved challenge for a genetic circuit (the problem is somewhat easier for designing a set of DNA strand-displacement reactions). In the future, models based on some combination of detailed biophysical modeling and machine learning may allow the accurate predictive of rates of transcription, translation and degradation from sequences, enabling the design of parts with particular parameters. However, at present, design is limited to the choice between members of a small set of available parts.

This makes design a problem of selecting *discrete* values corresponding to part choices, rather than of directly choosing *continuous* values corresponding to actual parameter values. In the Bayesian design framework, this can be captured by representing the choice of each part with a random value from a categorical distribution.

However, it is also necessary to ensue that the parts assigned are consistent with a particular topology. These relationships are difficult to incorporate into the Bayesian design approach, but are comparatively easy to encode as a logical formula (e.g., $(promoter2 == pTet1 \wedge promoter2 == pTet2) \implies cds1 = TetR$), in which form an SMT solver is able to reason about them. SMT-ODE solver might be useful for such design problems, which have a much more logical/-combinatorial structure than the examples considered in Chapter 3.

4.6 SUMMARY & CONTRIBUTION

The contributions described in this chapter are:

- creating a visual analytics tools to visualise the results of parameter estimation performed using ABC-SMC, enabling the investigation of how parameter values affect the behaviour of a system
- extending the existing packages `cuda-sim` and `abc-sysbio`: specifically, these modified the way in which rate rules and compartments with volumes other than 1 are handled, and implemented support for `kineticLaws` including time delays (simulated using the constant-step Runge-Kutta method [60 📖, Chapter II.17] running on the GPU)
- demonstrating that the ABC-SMC method can be combined with the robustness degree of a specification expressed in Signal Temporal Logic to estimate the posterior distribution of parameters, given that a specification is met
- creating a software tool that allows a user to visually draw a specification using TimeRails, and then visually explore the sets of parameters for which this specification is satisfied

Part II

REPRESENTING DESIGNS

5

MODEL COMPARISON

5.1 INTRODUCTION

This chapter describes `sbml-diff`, a tool that is able to read a model of a biochemical reaction network in SBML format and produce a range of diagrams that represent it at different levels of detail. Each diagram type can be used to either visualise a single model, or to visually compare two or more models. `sbml-diff` is freely licensed under the 3-clause BSD license. It can be used online through a form interface[1], downloaded[2] and imported into other software as a Python package or used as a free-standing command-line application.

In order to predict how synthetic biological circuits will function, or determine how parameters should be tuned, it is necessary to use mathematical models. Ideally, these models would be expressed in a standardised data format that provides interoperability between software packages, so that models can be constructed and simulated using different software, the results of simulating the same model using different simulation packages can be compared, and researchers can use their preferred tools to reproduce or extend previous work that used different tools. One such standardised format is SBML [71], a free and open interchange format for computer models of biological processes based on eXtensible Markup Language (XML). CellML [26] is an alternative modeling language that is also based on XML, but it was initially developed to represent models of cardiac cell dynamics and is still predominantly used for models in physiology, rather than in systems or synthetic biology.

1 http://sysos.eng.ox.ac.uk/tebio/sbml-diff
2 https://github.com/jamesscottbrown/sbml-diff

SBML

An SBML model consists of `species` that participate in `reactions`. A `reaction` has associated lists of `reactants` (which are consumed in the reaction), `products` (which are produced by the reaction), and `modifier` species (which are neither consumed nor produced by the reaction, but modify its rate). The rate of a `reaction` is given by its `kineticLaw`, which is a formula expressed in Content Mathematical Markup Language (MathML) that may depend on `parameters` and/or the concentration of `reactant` or `modifier` species.

Additionally, an `event` may reset the values of parameters and concentrations when a particular `trigger` expression is satisfied. A `rule` may specify that the value (in the case of an `AssignmentRule`) or rate of change (in the case of a `RateRule`) of a parameter or concentration is given by a particular MathML expression, or that a particular function of parameters and/or concentrations must always equal zero (in the case of an `AlgebraicRule`).

Why a visual comparison?

The ability to compare SBML models is important, both to compare models of different systems, and to compare different versions of a model corresponding to the same system. Many other engineering fields, particularly software engineering, rely heavily on systems of *version control* to keep track of designs produced at each stage of an iterative design cycle. This is often accompanied by the use of file differencing (`diff`) tools to directly compare different versions and identify what changes were made. However, directly comparing two models in SBML format as text is unsatisfactory: `diff` can be used, but it is difficult to spot the salient features in their output (compare Listing D.1 and Figure 5.3). Also, many textual changes are not significant (e.g., changes in white-space or the ordering of elements), and if the `id` of a species is changed, this change will appear in many places (e.g., the list of `reactants`, list of `products`, and `kineticLaw` for reactions involving that species).

A visual comparison is able to put a change into context, showing not only what has changed, but also how this relates to the other, unchanged, components. Whilst it is possible to compare a set of models by simply juxtaposing independently created visualisations of each, this requires the user to 'spot the differences' between diagrams in which equivalent nodes may be in different spatial positions, a difficult task. `sbml-diff` allows visual comparisons using a single diagram and can quickly highlight changes made between *versions of a single model*, make it easy to identify when a particular change was made, or check that no unintended changes were made at the same time as an intentional edit. It can also be used to check a model has not been accidentally altered by the process of importing and then re-exporting from another tool, or converting back-and-forth between file formats.

Importantly, `sbml-diff` can also be used to compare *models corresponding to different synthetic circuit designs*, and visually show the difference between them. The most developed standard for biological designs is Synthetic Biology Open Language (SBOL) [132], which represents designs as being composed of components with sequences and sequence annotations. Tools exist that use SBOL designs to automatically generate corresponding SBML models [131], which could be compared using `sbml-diff`.

`sbml-diff` can also produce a diagram of a *single model*. Such a diagram is particularly useful in understanding the structure of a model produced by another researcher. Visualisation can also reveal structural features that are likely to correspond to errors. For example, any parameters that are set by a rule but not used elsewhere in the model become obvious; these may indicate that the wrong parameter is used or set somewhere in the model, or that the model has been changed and the parameter is no longer needed.

EXISTING TOOLS

There are some existing tools which are capable of visually representing a single SBML model as a bipartite reaction graph. These include

large GUI packages with many other features (e.g., `CellDesigner` [46] and `iBioSim` [110]), for which the visualisation component is difficult to script or incorporate into other tools, and a few freestanding tools that can generate DOT output (e.g., The Systems Biology Format Converter's `SBML2DOT` Java module [129], or the Python library `VisualiseSBML` [52]). Several of these ignore rules and events, or otherwise do not visualise all elements of an SBML model.

`Cy3sbml` [87] provides the ability to import an SBML file into Cytoscape [140], creating a network in which each component is represented by a node. However, whilst a user could manipulate this network in Cytoscape to produce a diagram similar to those produced by `sbml-diff`, `cy3sbml` cannot produce such diagrams automatically.

`BiVeS`[3] [135, 136 🎓] is a tool for comparing SBML models, intended to track changes in a model over time. It produces outputs in a variety of formats but its visualisation abilities are currently relatively limited: work is underway to produce a companion tool to produce diagrams in the Systems Biology Graphical Notation (SBGN) Process Description language [105]. However this would be complementary to `sbml-diff`, rather than competing with it, as SBGN represents the underlying biochemistry, rather than the structure of the corresponding mathematical model; the same biochemistry can be represented in multiple ways in an SBML model, and in practice there may be difference between these equivalent formulations (for example, if a tool is being used that does not support rules).

DIAGRAM TYPES

By default, elements in two models are treated as the same entity if they have the same `id` attribute. Optionally, two elements with different `id` attributes can be treated as the same entity if they have the same set of MIRIAM [91] annotations of type `is`. In this case, the `id` of one is changed to be the same as the other, and all `reactions`/`rules`/`events` are updated accordingly.

3 The name `BiVeS`, together with that of the `BudHat` package from the same group, is a pun on the animated sitcom *Beavis and Butt-Head*.

Colour is used to indicate whether each node and edge is common to all models (grey), some but not all models (black), or a single model (the colour specific to that model); the default colours were chosen to be distinguishable by colour-blind users. A dashed node edge indicates that a component is shared between models, but with differences in its attributes: a rectangle with a dashed border indicates a reaction for which not all models have the same `kineticLaw`; an ellipse with a dashed double border indicates a species for which not all models have the same value for the `boundaryCondition` attribute[4].

5.1.1 *Default view*

In the default view, there is a direct mapping between the structure of an SBML model and the visual elements in the diagram. This view is intended to directly represent the structure of the mathematical model represented in SBML, rather than the structure of the reaction network that is being modeled, which may be modeled in several different ways. For example, a reaction may be modeled in SBML using a reaction with the attribute `fast` set to `true`, or by an `assignmentRule`. We therefore map components directly onto symbols, using the mapping shown in Figure 5.1. A rule may set values for multiple concentrations or parameters; the graphical representation for such rules is shown in Figure 5.2.

The key SBML components are compartments (drawn as dashed rectangles), species (ellipses, drawn with a double border if `boundaryCondition` is `true`), parameters (labels), rules (parallelograms), reactions (rectangles), and events (diamonds).

To reduce clutter and increase clarity, we depict only those parameters whose values are set by `rules` or `events`. However, the effect of every parameter can be seen if reaction nodes are labeled with their rate law. Function definitions are not shown, as we substitute them into any expression which uses them. Units and `initialAssignment`

4 If this attribute is `true`, it means that the concentration of the species is not changed by reactions (though, if `constant` is `false`, it may still be affected by `rules` or `events`).

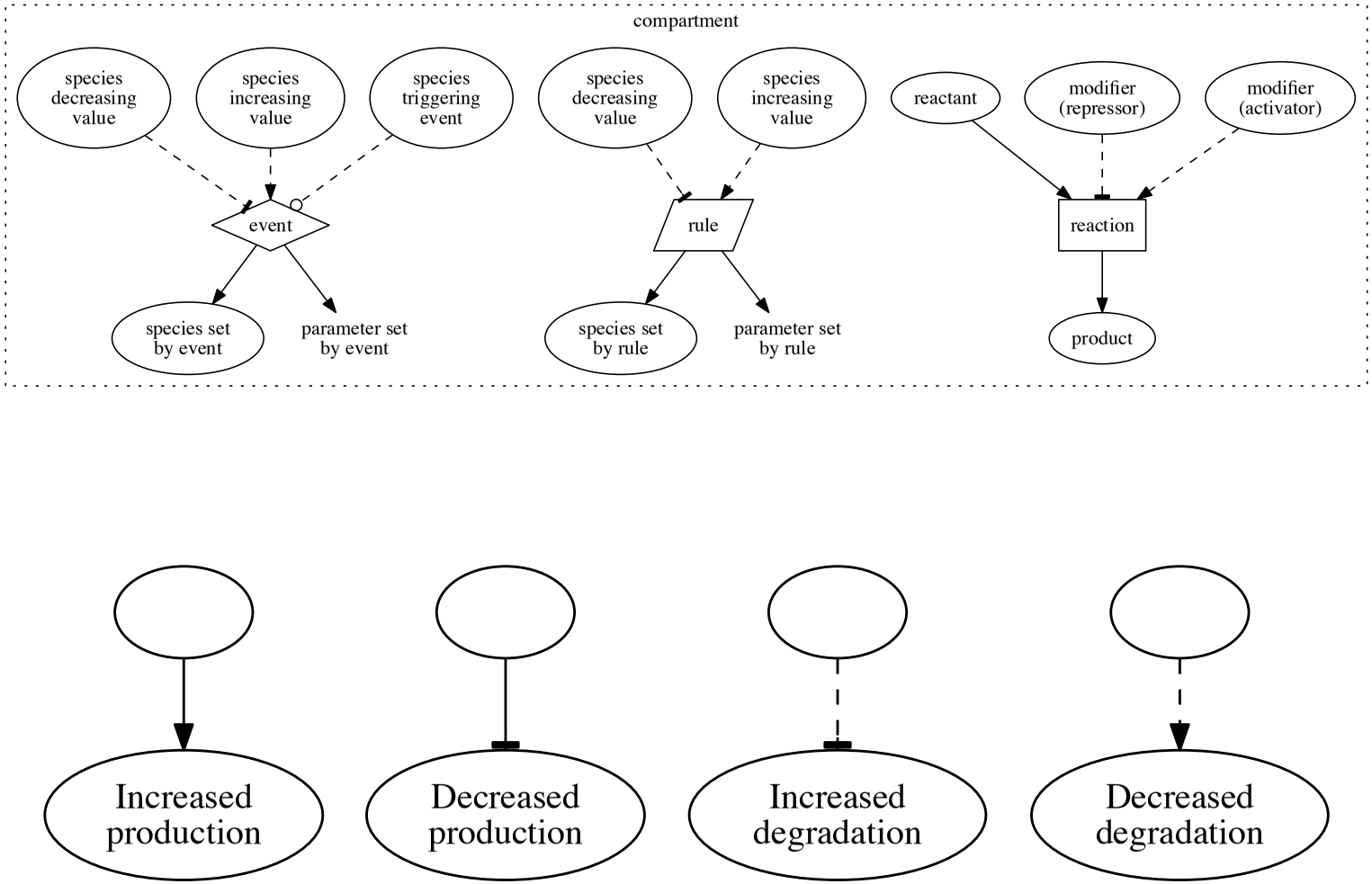


Figure 5.1: Meaning of each graphical element in the default (top) and abstract (bottom) views. The meaning of the lines differs between the two diagram types, but this should not cause confusion as in the default diagram they join species and non-species, and in the abstract diagram they join species to species.

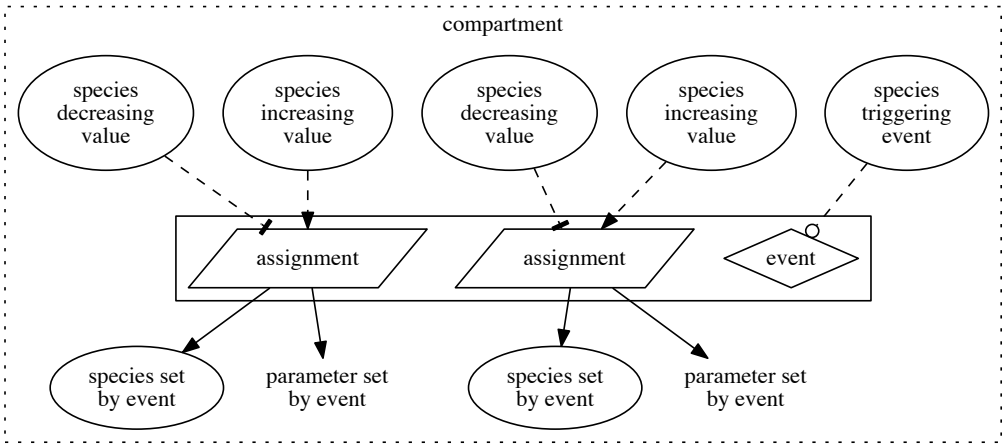


Figure 5.2: Representation of an event containing two assignment elements.

components are not shown, as these concern the values of parameters rather than the structure of the model.

Components are joined by arrows: if a line is solid, it indicates that a species' concentration (or parameter's value) *is affected by* an event, rule or reaction; if a line is dashed, it shows that a species *affects* an event, rule or reaction in some way. Normal arrowheads indicate activation, and T-shaped arrowheads indicate repression (arrowhead direction is determined numerically—see Section 5.3.2).

If an event includes multiple assignment elements, it is represented as a rectangle containing a separate parallelogram node representing each assignment, and a single diamond node representing the trigger statement.

An option allows the user to adjust the level of detail, by choosing between labelling reaction nodes with the corresponding reaction `name`, `kineticLaw`, both, or neither. If a reaction has the attribute `fast` set to `true`, this is indicated by an 'F'; if a reaction has the attribute `reversible` set to `false` this is indicated by 'IR' (**IR**reversible): when models are compared, these two markers are individually coloured using the same rule as other visual elements.

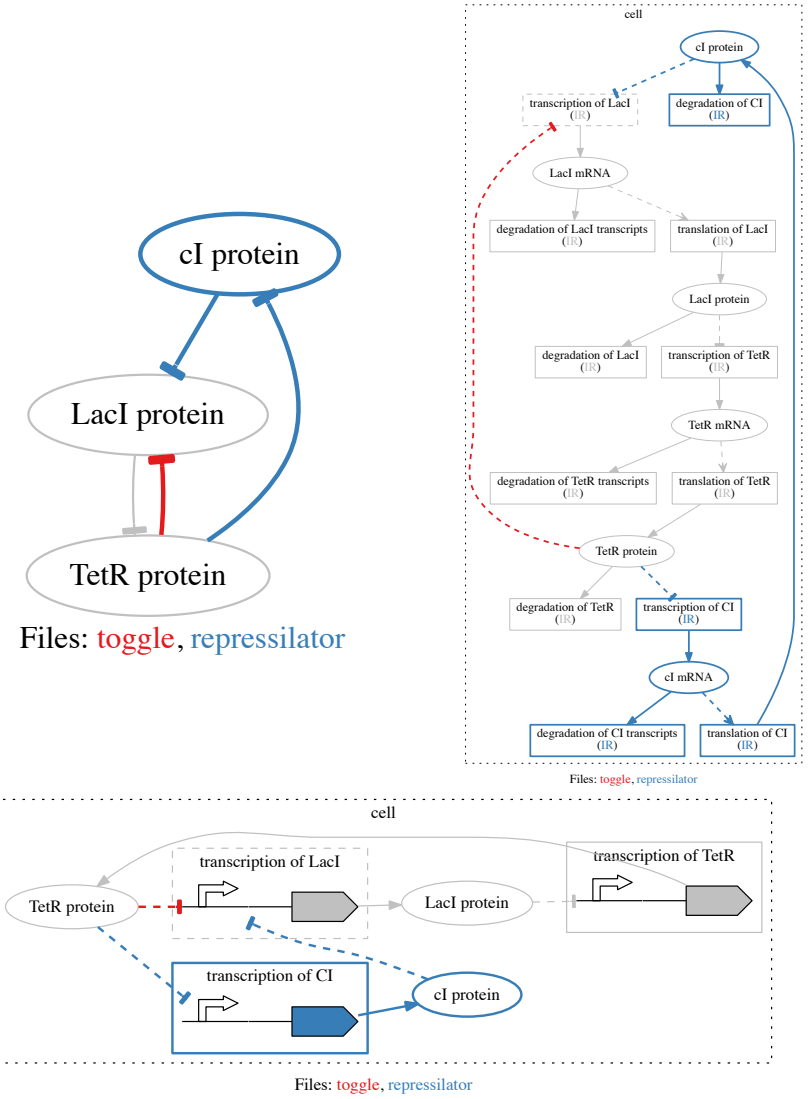


Figure 5.3: Three views of the comparison between two models, representing the repressilator [39] and the toggle-switch [51]: the abstract view (top left, Section 5.1.2), the default view (top right, Section 5.1.1), and the cartoon view (bottom, Section 5.1.3).

5.1.2 *Abstract view*

The abstract view (Figure 5.1 bottom, Figure 5.3 top left) does not represent reactions using nodes, and instead draws edges directly between species to indicate interactions between them. For each reaction, an edge is added from each species appearing in the `kineticLaw` to each reactant and product. Edges corresponding to a species increasing the rate of its own degradation are hidden, as these would typically be present for all species and create clutter.

These interactions are categorised into four types, distinguished visually by two styles of arrowhead (arrow or T-shaped, indicating the *sign* of the interaction) and two styles of line (solid or dashed, indicating whether the interaction affects the rate of production or degradation).

`sbml-diff` provides the option to elide a list of species (e.g., intermediate mRNAs)—these species are not drawn, and if they increase the production of a second species, then the heads of arrows incident to them are moved to that species (i.e., if $A \to B \to C$ and B is elided, then B is not drawn and instead an arrow is drawn directly from A to C).

5.1.3 *Cartoon view*

The cartoon view (Figure 5.3 bottom) resembles the default view, but replaces some rectangular reaction nodes with symbols indicating the nature of the reaction. Specifically, any reaction annotated with a Systems Biology Ontology [92] term of SBO:0000183 (transcription) is replaced by a compound symbol containing a single promoter symbol and a coding sequence symbol for each product. Reactions of type translation are automatically elided, unless the intermediate mRNA participates in a reaction that is not annotated as SBO:0000184 (translation) or SBO:0000179 (degradation).

5.2 EXAMPLE: MULTIPLE REPRESENTATIONS OF A REPRESSILATOR

To provide a comparison between different output diagram types applied to a single model, we present several representations of the same model of the repressilator, BioModels entry BIOMD0000000012.

The full view (Figure 5.4), makes it clear that the parameters `beta`, `alpha`, and `alpha0` have values that are assigned by rules, but are not otherwise used. These would be difficult to identify by reading the text of the SBML model, and impossible to identify from the graphical output of a visualization tool that ignores rules.

We can investigate this further by using `sbml-diff` to compare the rate-laws (using the `-kinetics` argument):

Reaction	Latest Model	Original Model
Reaction1	kd_mRNA * X	k1 * X
Reaction10	a0_tr + (a_tr * (KM ^ n)) / (KM ^ n + PZ ^ n)	alpha0 + (alpha + (PZ ^ n) * alpha1) / (K ^ n + PZ ^ n)
Reaction11	a0_tr + (a_tr * (KM ^ n)) / (KM ^ n + PX ^ n)	alpha0 + (alpha + (PX ^ n) * alpha1) / (K ^ n + PX ^ n)
Reaction12	a0_tr + (a_tr * (KM ^ n)) / (KM ^ n + PY ^ n)	alpha0 + (alpha + (PY ^ n) * alpha1) / (K ^ n + PY ^ n)
Reaction2	kd_mRNA * Y	k1 * Y
Reaction3	kd_mRNA * Z	k1 * Z
Reaction4	k_tl * X	X * beta
Reaction5	k_tl * Y	Y * beta
Reaction6	k_tl * Z	Z * beta
Reaction7	kd_prot * PX	PX * beta
Reaction8	kd_prot * PY	PY * beta
Reaction9	kd_prot * PZ	PZ * beta

This reveals the probable reason why the model includes unnecessary rules. The original file used `alpha` instead of `0`, `alpha0` instead of `a0_tr`, `alpha1` instead of `a_tr`, and `beta` instead of `k_tl` and `kd_prot`. The model was then changed so that these parameters (`alpha0, alpha, beta`) were set by rules rather than being directly assigned values in the `listOfParameters`, and the reaction rate laws were modified to use new parameters (`a0_tr, a_tr, k_tl`), but the old parameters and the rules that set their values were not removed.

Hiding the rules (Figure 5.5) shows the structure of the chemical reactions more clearly; and the cartoon view (Figure 5.6) further clarifies the structure of the genetic network.

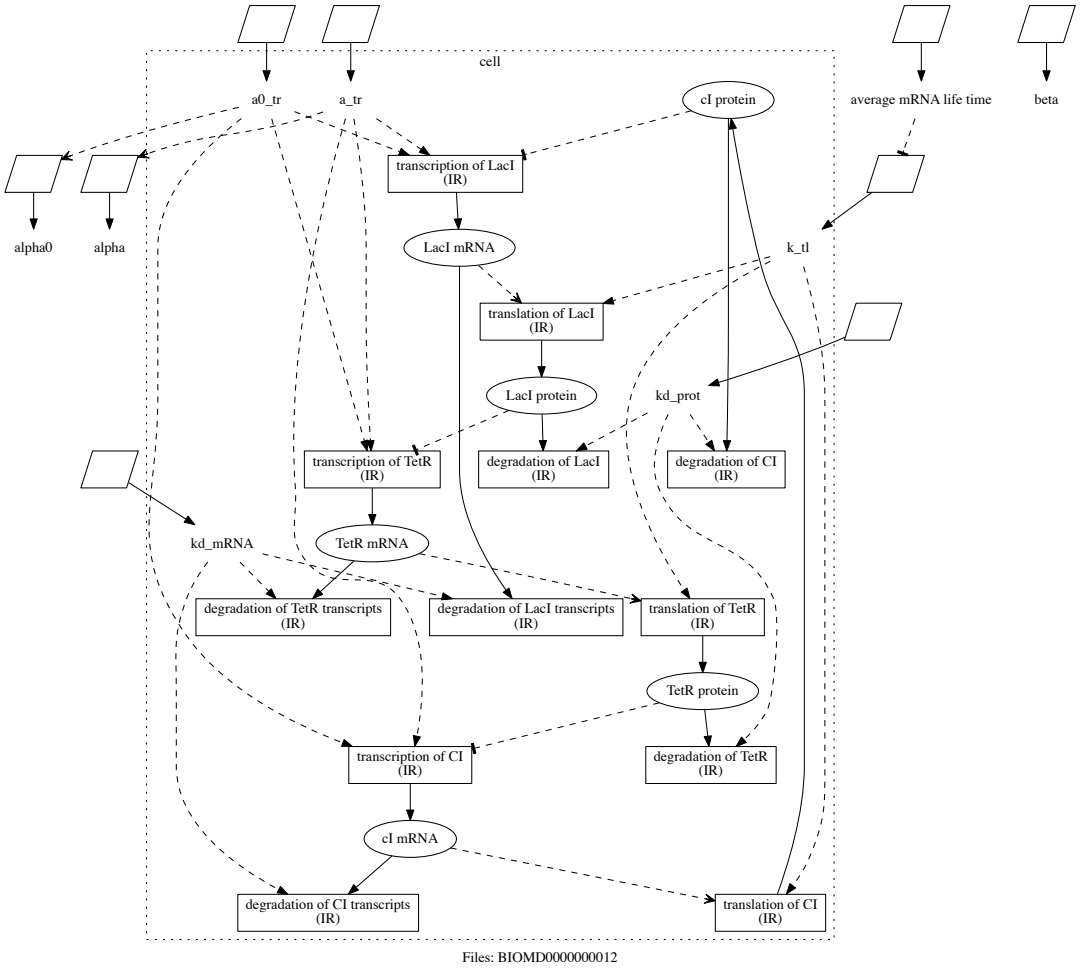
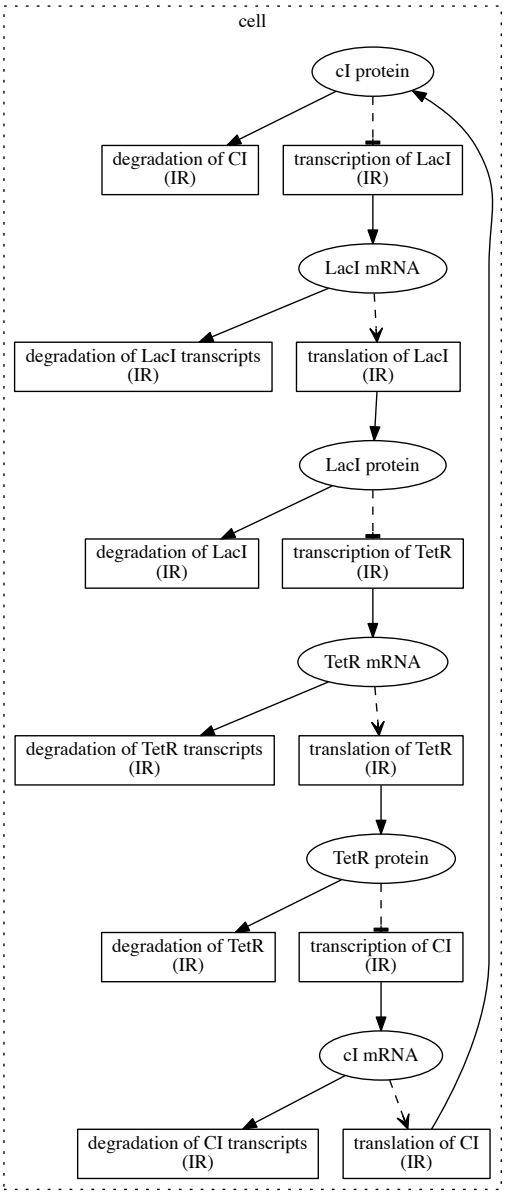


Figure 5.4: It can be seen that the parameters `beta`, `alpha`, and `alpha0` have values assigned by rules, but are not otherwise used. These would be difficult to identify by reading the text of the SBML model, and impossible to identify from the graphical output of a visualization tool that ignores rules.



Files: BIOMD0000000012

Figure 5.5: Hiding the rules reveals the structure of the chemical reaction network more clearly.

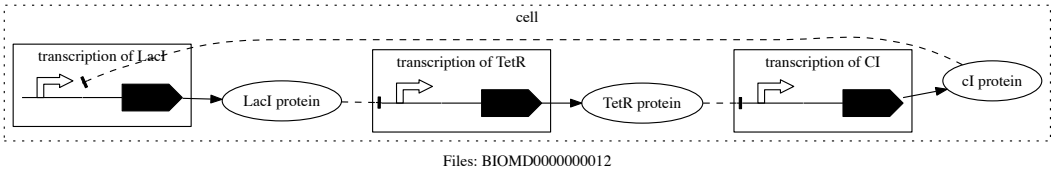


Figure 5.6: The cartoon view shows the structure of the genetic network more clearly.

5.2.1 *Application to larger models*

`sbml-diff` was intended to be applied to relatively small systems, such as synthetic genetic circuits. It can be applied to systems of any size, but for complex systems the resulting diagrams are likely to become unintelligible due to overlapping edges. One potential solution to this problem would be to hide much of the network, and only draw a small neighbourhood around nodes or edges that differ between the systems being compared.

5.3 IMPLEMENTATION

5.3.1 *Overall structure*

`sbml-diff` can be used as a Python package, as a freestanding command-line tool, or through a form on our website.

The core functionality is implemented as a Python package which receives SBML models as strings, and produces output in the DOT language specifying how the requested diagram should be drawn. The DOT file can be converted into an image by Graphviz [49], or by other compatible software.

The command-line program is a Python script that provides a wrapper around this package.

The web interface accessible at `http://sysos.eng.ox.ac.uk/tebio/sbml-diff` is also implemented in Python, using the Flask microframework. When files are uploaded through the form, they are saved to

a directory on the web-server. The `sbml-diff` package is used to perform the comparison, and the resulting DOT output is saved to disk. Graphviz is then used to produce images in both PNG and PDF formats.

`sbml-diff` first iterates over each model, recording every object that occurs in any model in a single meta-model corresponding to the union of each individual model. Each object (`species`, `reaction`, `rule`, or `event`) and edge (involvement of a `species` or `parameter` with another object) in this meta-model is associated with a list of models in which it is present; each attribute (e.g., a `reaction`'s `kineticLaw`, or the stoichiometric coefficients associated with an edge) is recorded as a dictionary mapping values to a list of the models in which the corresponding attribute takes that value. As each model is processed, any expressions that include user-defined functions are expanded out using the function definition. `sbml-diff` then iterates over this combined meta-model, generating Graphviz `DOT` code for every object, and applying styling to each based on the corresponding list of models.

5.3.2 *Determination of arrow direction*

In an SBML model, a `reaction` can have a `listOfModifiers` element, listing species that appears in the kinetic rate formula of a reaction, but do not appear in the list of reactant or products. SBML itself does not specify a way to record whether this interaction is activatory or inhibitory, but it is possible for the modeller to explicitly indicate the role of the modifier species by annotating it using the appropriate term from the Systems Biology Ontology: either SBO:0000459 (stimulator), SBO:0000020 (inhibitor), or one of their more-specific child-terms. However, such annotations are often lacking from models.

For example, of the 640 curated models in the 2017-06-26 release of the BioModels database [93], 221 (35%) contain no reactions with modifier species, and 391 (61%) contain modifier species but no an-

notations describing the role of for any of these, and 28 (4%) contain one or more annotated modifier species[5] (Figure 5.7).

Additionally, there are 'shorthand' languages that facilitate writing SBML models by hand (such as Antimony and SBML-shorthand) which do not support annotation with SBO terms.

In the absence of explicit information, the simplest (and most conservative) approach is to simply draw arrows with generic arrowheads, indicating that the effect of the interaction is unknown. However, such ambiguity would considerably reduce the usefulness of the diagrams, and so we attempt to determine which arrowhead to draw by examining the reaction's `kineticLaw`. The heuristic used may occasionally give misleading results[6], but we believe that the benefit of being able to correctly draw arrowheads in most cases outweighs the risk of sometimes getting it wrong.

A näive approach would be to recursively transverse the tree of arithmetic operations, keeping track of whether the expression at each stage is a monotonically increasing, monotonically decreasing, constant, or non-monotonic function of the modifier species, assuming that all parameter values and concentrations are non-negative. Such a procedure works for some functions, such as the Hill repression function: as x and n are non-negative, x^n is a monotonically increasing function of x, so $(K_m + x^n)$ is a monotonically increasing function of x, and $1/(K_m + x^n)$ is a monotonically decreasing function of x. However, such an approach fails for the commonly encountered Hill activation function $x^n/(K_m + x^n)$: both the numerator and denominator of this fraction are monotonically increasing functions of x, so this is an indeterminate case. To determine whether this is an increasing or decreasing function, we would need to differentiate symbolically, and test whether the resulting expression $(nK_m x^{n-1}/(K_m +$

5 Of the 612 curated models in the 2016-05-10 release of the BioModels database [93], 7 use SBO:0000020 (inhibitor), 2 use SBO:0000206 (competitive inhibitor), 2 use SBO:0000207 (non-competitive inhibitor), 5 use SBO:0000459 (stimulator), 20 use SBO:0000461 (essential activator), and 3 use SBO:0000462 (non-essential activator).

6 The arrowhead may be misleading if the arrow direction depends on the value of parameters and the SBML file does not set the `value` attribute on each parameter, or if the `kineticLaw` is not monotonic with respect to the species concentrations. In the latter case, the arrowhead is correct for *some* values of the concentration vector, but not for *all*.

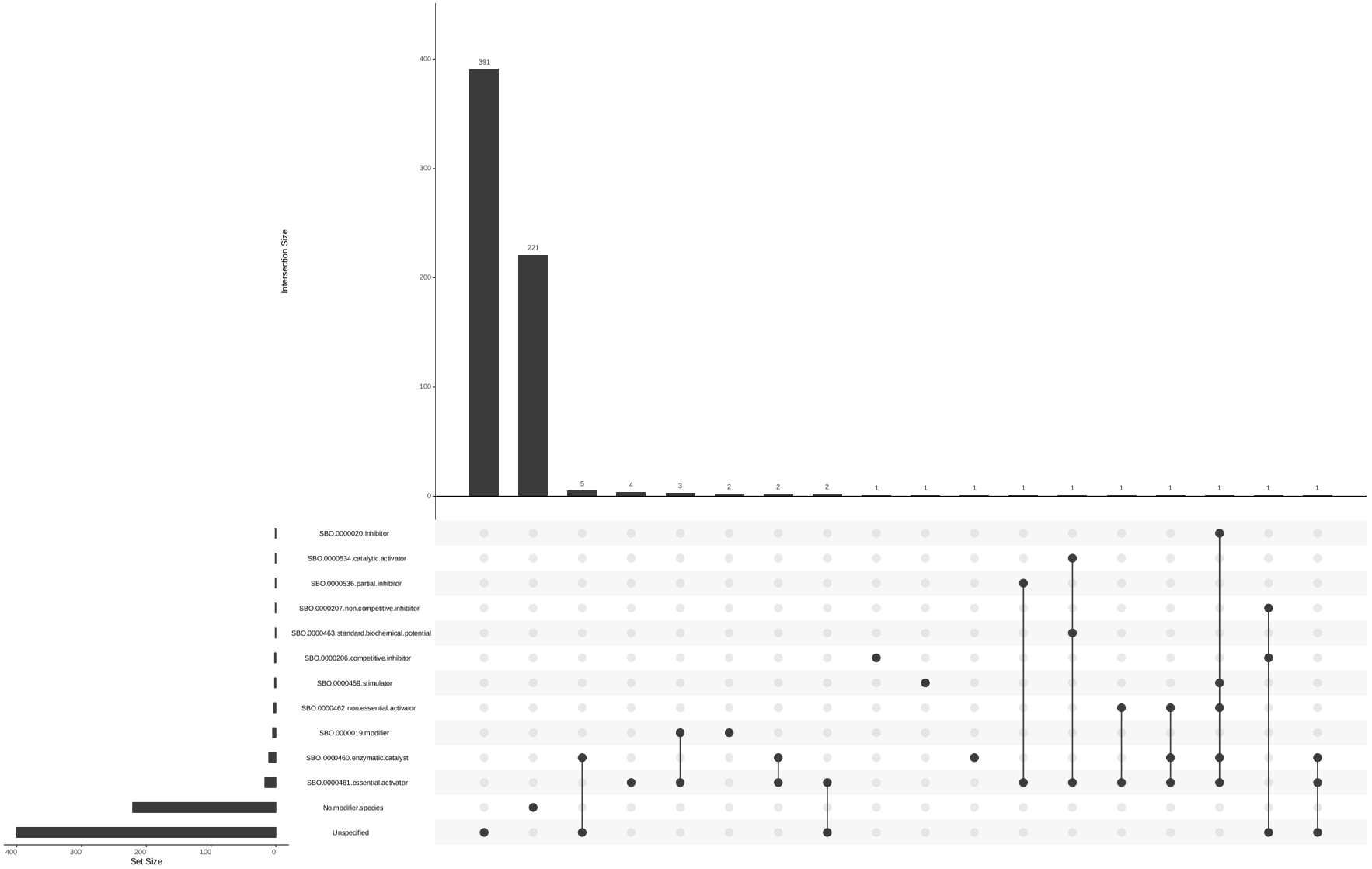


Figure 5.7: An UpSet plot [94] showing the proportion of SBML models in the curated BioModels [93] collection (release 31, 2017-06-26) using each combination of SBO terms to annotate the role of modifier species. Horizontal bars indicate the total number of modifier species annotated with each term; vertical bars indicate the number of modifier species annotated with the particular combination of terms indicated by the black circles beneath it. 'Unspecified' indicates that the type of interaction for a particular modifier is not indicated; 'No modifier species' indicates that no reactions in a model involve modifier species.

$x^n)^2$) has a constant sign for all non-negative parameter values and concentrations[7]. Whilst the first task can be easily achieved in Python using the SymPy library, the second is more challenging and SymPy's `Assumptions` module is not currently sufficiently powerful for this task.

As it is difficult to write a program that can verify whether an arbitrary input function is monotonic we do not attempt to do this, and instead simply assume that the `kineticLaw` is a monotonic function of each modifier species. This assumption is true for most of the commonly chosen rate functions (e.g. Mass-Action, Hill activation, Hill repression, Michaelis-Menten). There are times when such an assumption may be mistaken[8], but such cases are relatively unusual.

Under such an assumption, it is sufficient to evaluate the `kineticLaw` at two values of the modifier species concentration, and the choice of these values will not affect the determined direction.

It is also necessary to choose values for the parameters (and other concentrations). However, the choice of these values could affect the determined direction: for example, if the Hill coefficient is negative, then the Hill activation function becomes monotonically decreasing, rather than monotonically increasing. Fortunately, this choice is not important in many of the most common cases: provided that all parameters and concentrations are positive, Hill activation, Hill repression and Michaelis-Menten `kineticLaws` are monotonically increasing, decreasing and increasing (respectively) functions, regardless of the precise numerical values.

If the model specifies the value of a parameter (or initial concentration of a species), we use that numerical value. Otherwise, we arbitrarily use a value of one. The `kineticLaw` is then compared at two arbitrarily chosen values of the modifier's species concentration (0.1 and 1); if a numerical error (e.g. a division-by-zero error) is encountered during one of the evaluations of the `kineticLaw`, the interaction is treated as indeterminate and represented by a diamond arrowhead.

7 Or, if parameter values are known, that it has a constant sign for all concentrations.

8 For example, the 'species concentration' might actually represent cell density, and a component of growth medium may increase growth rate when provided at low concentration, but decrease the growth rate when at high concentration due to toxicity or osmotic effects.

For a monotonic `kineticLaw`, this method is equivalent to testing the sign of an element of a finite difference approximation to the Jacobian of the system at a single point.

5.4 SUMMARY & CONTRIBUTION

This chapter has described `sbml-diff`, a tool for visually comparing models of biochemical reaction systems in SBML format. It can be used as a free-standing command-line application, or accessed through a web interface. It can also be used as a Python package, allowing it to be incorporated into larger software packages, such as tools for editing and curating collections of models, or incorporated into automated tests to ensure that other tools do not contain bugs that cause unintended changes to SBML files.

`sbml-diff`'s main innovation is the ability to directly visualize the differences between two or more SBML files (which may be different revisions of the same model, or distinct models). However, its ability to produce diagrams that show a range of levels of details, and the fact that it is able to represent `rules` and `events`, not just `reactions`, mean that it is also useful for visualizing the structure of a single model.

`sbml-diff` is re-used as a component in the system for Bayesian Design described in Chapter 4.

Part III

REPRESENTING EXPERIMENTAL PROTOCOLS

6

EXPERIMENTAL PROTOCOLS

6.1 INTRODUCTION

This chapter describes a graphical system (`List of Liquids`) for writing experimental protocols that can be executed by robotic systems such as those shown in Figure 6.1. We introduce a new graphical notation for protocols, and show that this is able to concisely represent many protocols that are in current use.

Section 6.1 introduces liquid-handling robots, describes existing systems for programming them, and explains why a new system is needed. Section 6.2 suggests an alternative diagrammatic representation for protocols designed to meet this need and Section 6.3 describes `List of Liquids`, a prototype protocol editor that uses it. Section 6.3.1 describes the interface, and Section 6.3.2 explains its implementation. Section 6.5 describes potential future work to extend the system. In addition to the examples contained in this chapter, many further examples of protocols expressed in the `List of Liquids` system are presented in Appendix C.

Why Automate?

One reason for automation is the desire for higher experimental throughput. This can be partly attributed to falling costs for many experimental techniques, such as DNA sequencing and synthesis (Figure 6.2): as experiments become cheaper, it becomes economically feasible to perform more of them than a human experimenter is capable of. The ambition of synthetic biology, and amount of experimental work that goes into a single paper, has expanded hugely since the early papers describing the toggle-switch [50] and repressilator [39], each of which characterized a single construct: the 2016 paper introducing the Cello

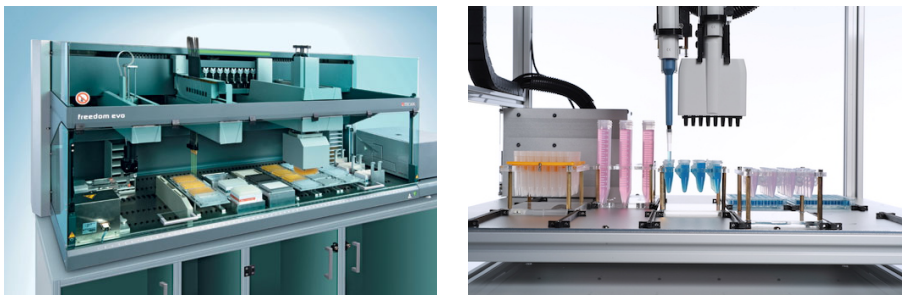


Figure 6.1: Two commercially available liquid-handling robots. **Left**: A Freedom Evo robot manufactured by Tecan. **Right**: An OT-Pro robot manufactured by OpenTrons.

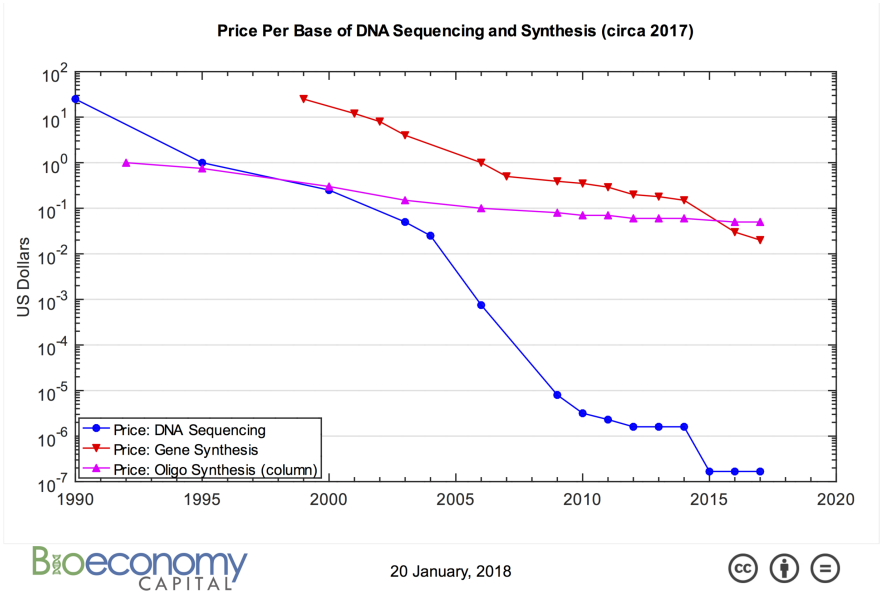


Figure 6.2: Costs for DNA synthesis and sequencing have fallen hugely, making higher throughput experiments financially feasible. Figure from Rob Carlson's blog (http://www.synthesis.cc/synthesis/2016/03/on_dna_and_transistors).

system for automated genetic circuit design [113] evaluated 60 designs, and the 2018 metabolic engineering paper about the 3-month DARPA biofoundry 'pressure test' describes work that "constructed 1.2 Mb DNA, [and] built 215 strains spanning 5 species" [24]. The construction of orthogonal gene expression systems requires screens that typically start with a library of 10^9 cells, select $10^3 - 10^4$ of these for further examination, and end up with one final hit: it is clearly infeasible for a graduate student to conduct such a screen by hand.

Improved reproducibility is another motivation for automating experiments. Compared to a human, a robot can potentially perform tasks such as pipetting with higher reliability and reduced variability. There is also a risk that humans will make mistakes, such as accidentally omitting or repeating an operation, whereas a robotic system does not suffer from tiredness or distraction.

Another reason to formally express protocols is to allow outsourcing to an external organisation or a division of labour between different groups of people in the same organisation (e.g., researchers and technicians). For example, the Aquarium system guides human technicians (rather than robots) through the execution of remotely submitted protocols [162 🎓]:

> By November 2016, the yeast cloning process in Aquarium enabled researchers at the University of Washington to build over 7,000 novel yeast strains without the researchers ever performing the cloning protocols themselves. This enabled researchers to view the yeast cloning as an abstract process so they can focus on the high level experimental design of their projects.

Outsourcing experiment execution could not only free up researchers' time, but also remove the need for some research groups to have their own lab, reducing their requirements for floor-space and capital expenditure on equipment.

Liquid handling robots vary in capabilities, and vary in price by several orders of magnitude (Figure 6.1). In the last few years, several

companies have launched relatively inexpensive 'personal' pipetting robots, such as the OpenTrons OT-One, with a price starting at $3,000.

However, laboratory automation is not without disadvantages: systems are often expensive and time consuming to program, and may lack the flexibility to adapt to different types of experiment. Additionally, unusual occurrences that would have been observed by a human experimenter (e.g., two liquids forming separate layers rather than mixing) may go unnoticed by a robot.

6.1.1 *Existing textual representations of protocols*

The software needed to automate experiments can be divided into two parts: the software used to write protocols, and the software that reads a protocol and controls a robotics platform (or instructs a human) to carry it out. Ideally, these would interact via an open standard format which could be generated by any protocol editing tool and read by any manufacturer's control software. Sadly, no such format exists, and there is little compatibility between manufacturers. We do not attempt to solve this problem by producing such a format, as the primary challenge is one of commercial coordination rather than a technical problem of academic interest.

Transcriptic describes Autoprotocol as "an open standard for life science experimental design and automation"[1] but whilst it can be generated by several tools (BioBlocks, Wet Lab Accelerator, and `Lists of Liquids`), the only software that can execute a protocol expressed in it is proprietary and runs only in Transcript's own facility. Synthace's Antha language includes drivers to control liquid handlers from several manufacturers, but whilst the Antha language is released as free software[2] under the GNU General Public License only limited documentation is available, and the corresponding graphical interface is proprietary. Several manufacturers support protocols expressed as CSV formatted 'pick-list' files specifying a sequence of *(source well, destination well, volume)* tuples, but this format cannot

1 http://autoprotocol.org/
2 https://github.com/antha-lang

represent all protocols and offers only limited control of transfers (for example, there is no way to specify when pipette tips should be changed, or what mixing should be performed before or after a transfer).

Other programming-language like textual representations for protocols include BioCoder [3], PaR-PaR [98], Aquarium [83, 162 🎓], Autoprotocol [152], Antha [147], RoboLiq [158] and Symbolic Lab Language [90]. Notably, Aquarium is intended to guide a human technician through executing a protocol, rather than being used to control a robot. There are also several graphical interfaces for constructing protocols (e.g., BioBlocks [59], Wet Lab Accelerator [11], and proprietary tools from robotic systems manufacturers) which are discussed in more detail in Section 6.1.2.

However, these interfaces place the primary focus on the specific liquid handling steps to be performed, rather than their intended results. We present here a graphical interface that reverses this focus: the user expresses what they want to achieve at a high level, and the system determines what movement of liquids between physical locations are necessary to achieve this. The system can produce a range of outputs, including a diagram of the specified operations, a description of the protocol in English, and an executable protocol in various formats (including Autoprotocol-python and OpenTrons).

The decision to make previous interfaces graphical was generally motivated by a desire to avoid requiring users to be familiar with a particular programming language (or any programming language at all) [11, 59]. However, the graphical representation that we have chosen has the additional advantage of producing a diagram with a structure reflecting the protocol's structure, rather than just representing it as a linear sequence of steps; this makes it easier to understand what a protocol is intended to do or identify errors.

6.1.2 *Existing visual representations of protocols*

Most graphical tools for writing protocols provide a block-based interface on top of an underlying textual representation (e.g., Wet Lab

Accelerator [11], Figure 6.3; Tecan Freedom EVOWare, Figure 6.4). Each command in the textual language is replaced by a block: this block is a form, and contains form elements (e.g., input boxes or selection boxes) that are used to enter the command's arguments or parameters. These blocks are typically arranged in a linear order.

Compared to a purely textual interface, block languages do offer some advantages [13]. Rather than *remembering* the name of a command the user merely has to *recognise* it from a list, and the form removes the need to remember the order in which arguments are supplied (or their name, if keyword rather than positional arguments are used); however, similar benefits can be provided by common Integrated Development Environment (IDE) features such as autocompletion on tab and tooltips listing the arguments of a function. By restricting the actions of the user in constructing a protocol, the editor can ensure that it is syntactically valid at all times in a way that would be impossible if the user were able to freely type text. A graphical editor may also appear less intimidating than a text editor or IDE to some novice users. These benefits apply only to *writing* protocols: arranging text in a box rather than as a line of code does not make it significantly easier to read and understand.

BioBlocks ([59], Figure 6.5) is based on Google's Blockly. Blockly is intended as a general purpose visual programming language, and to create BioBlocks it was adapted by adding new block types, and modifying the code-generation to produce output as AutoProtocol code and English language text, rather than as JavaScript code. It also provides an interface in which blocks are form elements, but whereas in most systems (e.g., Wet Lab Accelerator and EVOWare) each block represents an operation, in BioBlocks they can represent a container, an operation, or a complete experiment. They also compose together in a more complex way: the example in Figure 6.5 shows a container connected to the `container input` of a `thermocycling` operation, which is connected to `step 1` of an `experiment`. The blocks thus form a tree visually representing the syntax tree of the protocol being represented, rather than forming a linear chain.

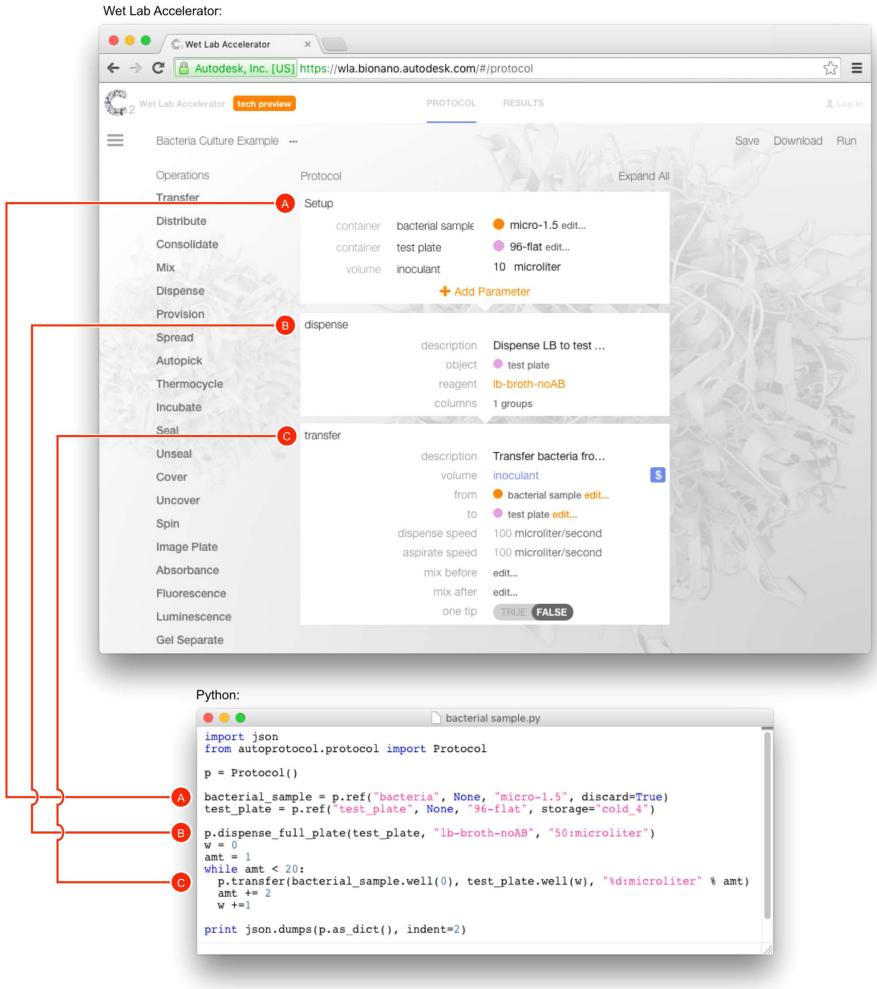


Figure 6.3: Wet Lab Accelerator (reproduced from Figure 3 of [11]). Protocol steps are presented in a linear order, and each maps directly onto a section of Python code (which, when run, uses the `autoprotocol` library to generate a representation of the protocol as Autoprotocol JSON).

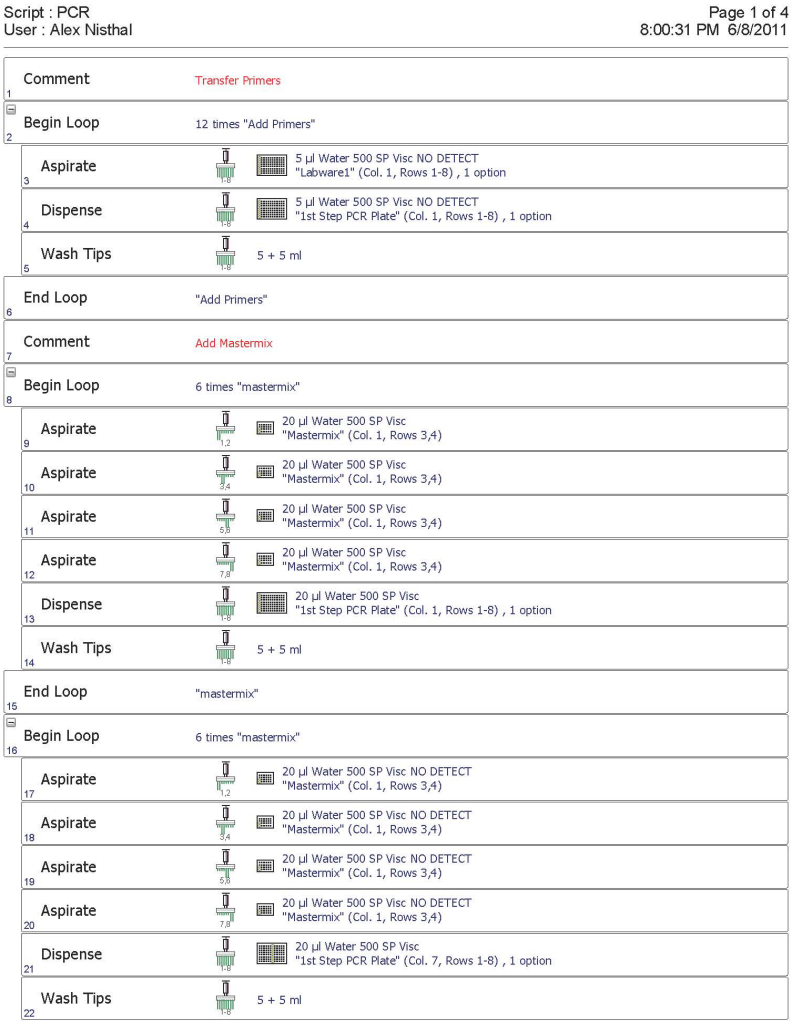


Figure 6.4: Screenshot of Tecan Freedom EVOWare. Protocol steps are presented in a linear order, with indentation used to represent control structures (loops or if-statements). This graphical interface adds little to a purely textual one: each line of code has simply been replaced by a box, which itself contains text.

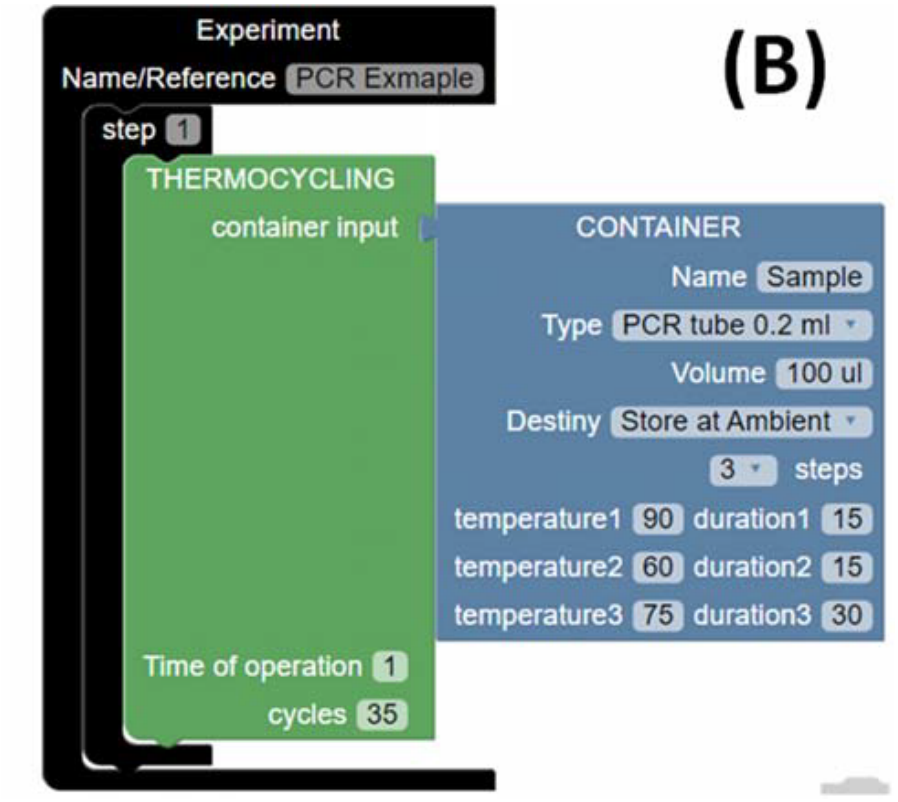


Figure 6.5: Screenshot of BioBlocks, reproduced from Figure 1 of [59]. In this interface the steps of a protocol are arranged in a linear order, but blocks are also used to represent entities other than steps, and are arranged in a way that represents the abstract syntax tree of a protocol.

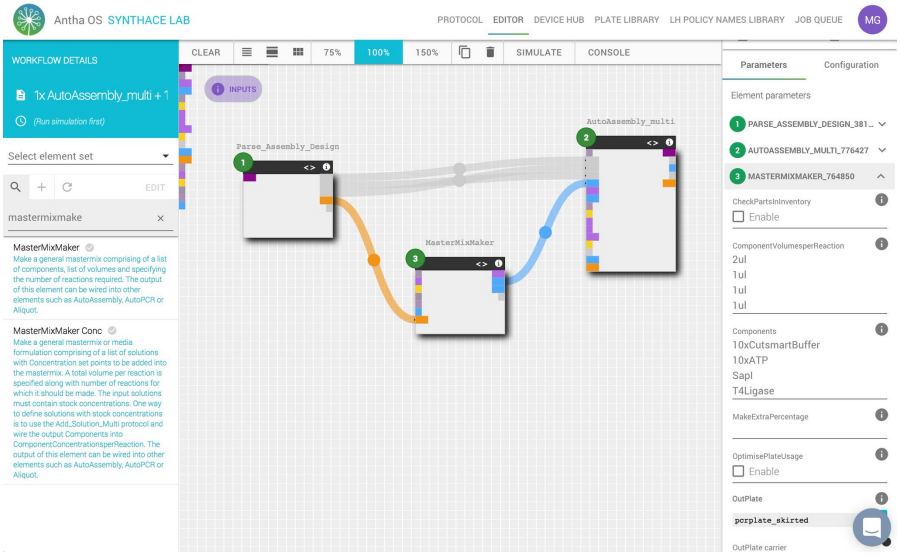


Figure 6.6: Screenshot of AnthaOS, reproduced from Synthace's marketing materials. This interface uses a data-flow paradigm to represent flows between individual protocols that are represented as 'black boxes' (in the sense that their contents are obscured—AnthaOS actually uses a white fill).

6.1.3 *Data-flow systems*

Data-flow languages are a form of visual programming language that has been successfully applied to many domains, including modelling physical systems (SimuLink, Modelica), data acquisition/instrument control (LabVIEW), processing visual media or music (SideFX's Houdini, Pure Data, MAX/MSP, Pure Data), and bioinformatics workflow management systems (Apache Taverna, Galaxy, Unipro UGENE).

The paradigm has also been applied to experimental biological protocols by AnthaOS and Aquarium. AnthaOS is a closed-source commercial graphical interface that represents protocols as 'black boxes' that can be combined into complete *workflows* or experiments. Individual protocol blocks (such as `MasterMixMaker`) are connected by arcs that represent the flow of liquids or parameters (Figure 6.6). The function of blocks are defined by code written in Antha, a Domain Specific Language based on Go.

Aquarium is conceptually similar: it refers to itself as a 'laboratory operating system' that allows protocols written in its own textual languages (Krill, a Domain Specific Language based on Ruby) to be connected together[3]:

> "Protocols define formal unit operation types with typed inputs and outputs – allowing the researcher to construct a complex workflow by linking an output sample of one protocol to an input of another using the Aquarium graphical workflow designer."

A key difference between AnthaOS and Aquarium is that Synthace has invested in creating drivers and middleware to allow the control of a range of liquid handling systems, whereas Aquarium has focused instead on guiding human technicians through protocols using touchscreen monitors.

Such tools are potentially very valuable: they can act as complete Laboratory Information Management Systems, keeping track of samples and reagent stocks, recording provenance information for each

3 https://github.com/klavinslab/aquarium

experiment and measurement, and performing scheduling. However, they are complementary to the problem addressed in this chapter: rather than making it easier to actually write an individual protocol or understand what a protocol does, they allow protocols to be composed together into larger workflows. Indeed, `List of Liquids` could be extended to generate code in the Antha or Krill languages, allowing protocols created with it to be integrated into AnthOS or Aquarium.

6.1.4 *Why a new representation?*

The difficulty of programming laboratory robots is widely recognised. Whilst this thesis was being written, the IT University of Copenhagen announced a new spinout, Flow Robotics, aiming to address this usability challenge using augmented reality and computer vision, and the CEO commented [145]:

> "The existing pipette robots on the market are difficult to program for smaller and changing tasks. So laboratory workers often end up in a situation where they can choose between spending 45 minutes programming the robot or spending 45 minutes just pipetting by hand."

The advice from Transcriptic on how to write a new protocol to run on their platform begins:

> "Using pencil and paper, draw out a flow chart of all of the steps in your current protocol"[4]

If it is necessary to draw a diagram in order to write a protocol, why not make a diagram the primary interface? Can we design a diagrammatic notation that is flexible enough to handle a wide range of experimental protocols whilst also being easily readable by a human, and sufficiently unambiguous to be interpreted by a computer?

4 *How to Write a New Protocol*. Transcriptic. https://developers.transcriptic.com/docs/how-to-write-a-new-protocol

This question has not been fully investigated by previous work. Whilst many software tools have been created, these have not done a good job of exploring the design space of potential representations: most are generally either textual (Section 6.1.1), or graphical interfaces in which textual commands have been replaced by blocks with the same linear ordering as the equivalent textual commands.

BioBlocks presents a protocol as a tree, rather than a linear ordering. However, we argue that this is the *wrong tree* to show the user: the user's real concern is not the syntax tree for the instructions in a protocol, but rather the tree showing how the different samples that are created during the execution of the protocol are derived from each other; the nodes in the tree should represent samples. Despite having child blocks, each step of an experiment exists at the same level and is linearly ordered.

Rather than designing the interface to map cleanly onto the required output format, which makes the job of the computer and tool developer easy, the interface should be designed to map cleanly onto the user's thoughts to make the user's job easy. The translation necessary to handle the mismatch between how a user naturally thinks and the low-level instructions required by a robot should be performed by the computer, rather than the user.

Compared to previous representations, we introduce two main innovations:

- we represent each set of aliquots that exists at any time during the execution of the protocol as a node
- we represent operations as acting on aliquots (which happen to have locations) rather than on locations (which happen to contain aliquots)

As an additional contrast to previous tools, Lists of Liquids is an open-source application that provides a single interface that can be used to control a range of automation platforms.

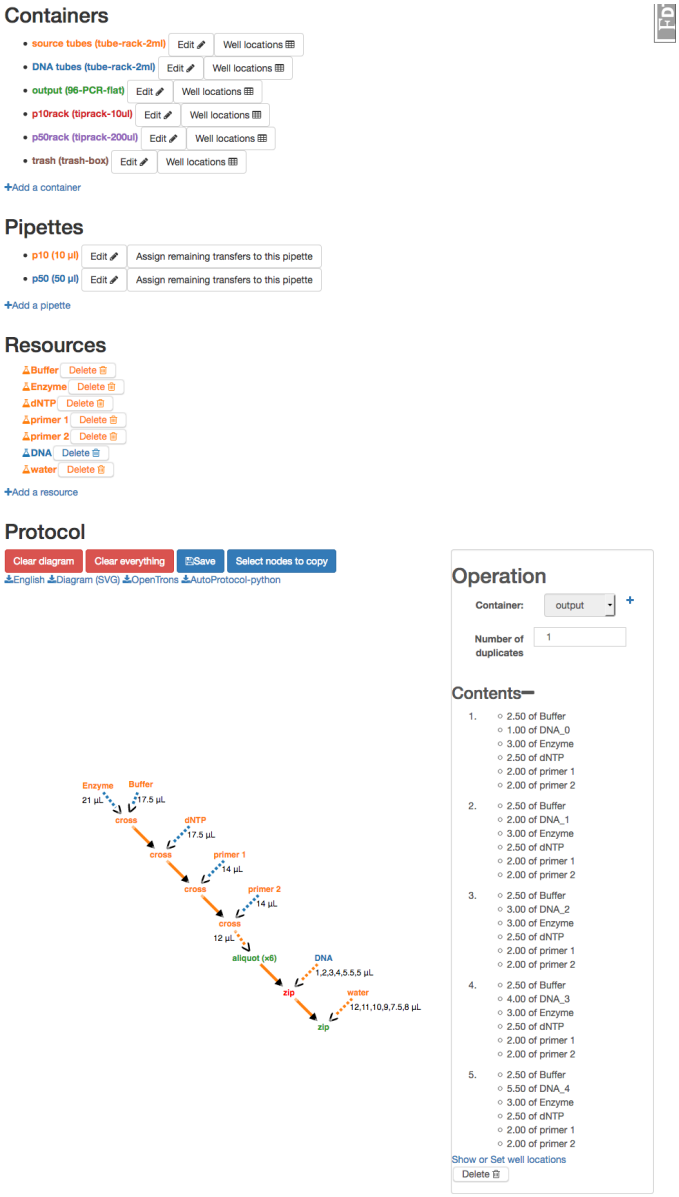


Figure 6.7: Screenshot showing the `List of Liquids` interface, showing a list of the containers, pipettes, and resources required by the protocol, followed by a diagram showing the operations performed when the protocol is carried out. A node labeled `zip` is highlighted in red, and a panel to the right shows its details, including a list of its contents.

6.2 THE list of liquids DIAGRAM

A protocol is represented by a list of required things (containers, pipettes, resources), and a diagram showing how they are used.

The key abstraction is the *list of liquids*, which is a list, each element of which is an aliquot that exists in a distinct physical location, and for which all elements simultaneously exist at some point in time. The composition of each aliquot is defined recursively, in terms of specific volumes from specific aliquots in other lists of liquids, with *resources* as the base-case for this recursion.

- *Containers* are things that can contain liquids (or tip-racks, which contain disposable tips for pipettes, rather than liquids). Some container types consist of a single compartment (e.g., a trash container), whereas others consist of multiple wells (e.g., a 96-well plate).
- *Pipettes* are the pipettes used to physically transfer liquids between locations. To export a protocol in OpenTrons format, the pipette used for each transfer must be specified [5], but for other output formats (e.g., AutoProtocol) this is unnecessary.
- *Resources* are *lists of liquids* that are initially present (rather than being created as a result of following the protocol), and can be regarded as inputs to the protocol. A resource may consist of a single liquid (e.g., if it represents a buffer or reagent), or multiple liquids (e.g., if it represents a set of samples).

The diagram (Figure 6.7) is a node-link representation of a Directed Acyclic Graph representing how particular *lists of liquids* are created from other *lists of liquids* as the protocol is carried out. In this diagram, the nodes represent *lists of liquids*, and the operations that act on them.

The volumes of liquid to be transferred are labeled on the corresponding arrows: this may be a single volume (in which the same volume is taken of all liquids), a list of volumes of the same length

5 The OpenTrons OT-One robots have actuators(s) that each accepts a regular pipettes, so needs to be told what it has been loaded with.

as the list of liquids, or, if a node represents a single liquid, a list of volumes of any length.

A pair of *lists of liquids* can be combined in two logical ways: given two lists x and y, the Cartesian product (`cross`) independently combines every element of x with every element of y, whereas the convolution operation (`zip`) combines each element of x with only the corresponding element of y. If one of the lists consists of only one element, then it is combined with each element of the other list. The operation of combining the lists is represented by a node, labeled as either `zip` or `cross`. As an example, a `zip` operation can be used to combine pairs of Polymerase Chain Reaction (PCR) primers, and a `cross` operation can be used to combine each of the resulting mixtures with each of several samples.

The diagram visually distinguishes between different ways in which things can be combined (Figure 6.8): a solid arrow indicates that the corresponding liquids remained in the same wells, rather than being transferred to new wells. When two liquids are combined, the first could be added to the second (dashed arrow from first, solid arrow from second), the second could be added to the first (dashed arrow from second, solid arrow from first), or both could be added to a new, empty, container (dashed arrows from both).

Iteration is represented by enclosure of the repeated operations in a dashed rectangle, labelled with the number of times that the operations are repeated. An example of this is shown in Figure 6.12, in which absorbance and fluorescence measurement operations are repeated (since these have an associated `incubate-before` time, this creates a time-series of measurements). The repeated operations must be a contiguous sequence of nodes, and must be operations performed on a list of liquids *in situ*, rather than a transfer or mixing of liquids (they cannot be `zip`, `cross`, or `pick`).

6.2.1 *Example*

To illustrate the benefits of the `List of Liquids` representation, we directly compare it to Wet Lab Accelerator. As an example, we use

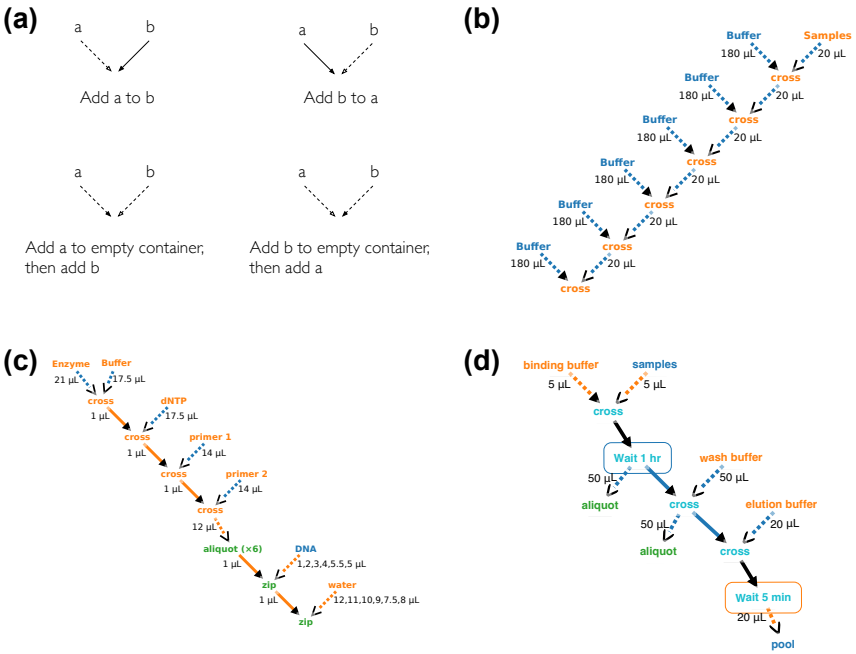


Figure 6.8: Different arrow styles corresponding to 4 ways of mixing (**a**), and diagrams for 3 example protocols, representing serial dilution (**b**), setting up for a PCR run (**c**), and a SequalPrep extraction (**d**).

the bacterial cell transformation example supplied with Wet Lab Accelerator. Transformation is the process by which bacteria take up DNA from their surroundings, and is a commonly used technique for introducing modified DNA into bacteria. It typically involves heat-shocking bacteria by transferring a tube containing bacteria in $CaCl_2$ solution from ice to a water-bath at ~42° for ~45 seconds then back to ice. However, 'Zymo Mix & Go Competent Cells' have been chemically treated in a way that makes them competent to take up DNA without the need for heat shocking [73]: this is particularly convenient for automated experiments, as heating and cooling an plate requires additional equipment.

Reading

We first consider the task of trying to *read and understand* an unfamiliar protocol.

Figure 6.9 shows cropped screen shots of this protocol as it is represented in both Wet Lab Accelerator and `List of Liquids`.

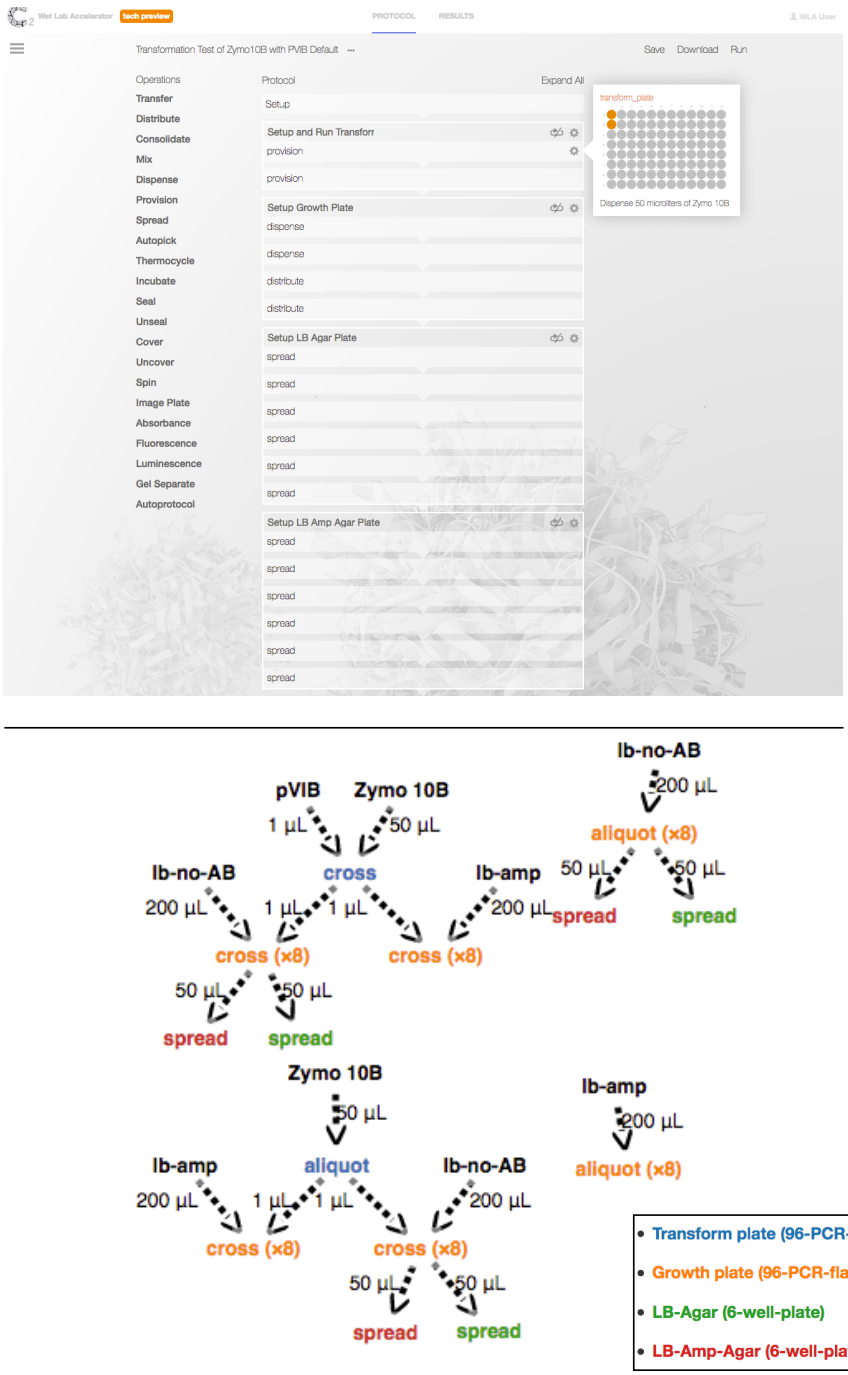


Figure 6.9: Comparison of the same protocol expressed in Wet Lab Accelerator (top) and `List of Liquids` (bottom). This protocol is for the transformation of *E. coli* bacteria with foreign DNA.

Looking at the `List of Liquids` diagram, we can see that there are 4 isolated subgraphs, which correspond to separate experimental controls. Focusing on the top left subgraph, we can see that the key aspect of the protocol is that cells ('Zymo 10B') are combined with DNA (the plasmid 'pVIB') on the transform plate, and then mixed with LB broth (either with or without ampicillin) on the growth plate, before being separately spread on the LB-agar plate and LB-Amp-agar plate. The experimental controls consist of omitting the plasmid to check that the untransformed bacteria are not already antibiotic resistant (bottom left), and of not adding any bacteria to the LB broth to check that it is not already contaminated with bacteria (top right and bottom right).

Looking at the screen shot of Wet Lab Accelerator, we see that the protocol has 4 stages: 'setup and run transfection', 'setup growth plate', 'setup LB agar plate', and 'setup LB Amp Agar plate' (in `List of Liquids`, the number of stages is shown by the height of the tallest subgraph). However, the details of what these involve are obscured: we see only that they involve 2 `provision` operations, 4 `dispense` operations, 6 `spread` operations, and 6 `spread` operations, respectively. There are two problems here: the first, and more easily fixable, problem is that the details of what each operation acts on is by default concealed, and can only be revealed for one operation at a time. The second problem is that when these details are revealed, they are expressed in terms of well locations, and the user must do the work of keeping track of what each well contains through time.

This makes it difficult to answer many questions that are easy to answer using the `List of Liquids` interface. For example, consider the question of which combinations of cells/no-cells, plasmid/no-plasmid, antibiotic/no-antibiotic are plated onto agar. In the list of liquids interface, the answer can be seen by inspection of the diagram (aliquots containing lb-no-ab, lb-no-ab + cells, and lb-no-ab + cells + plasmid are spread). Answering the same question using the Wet Lab Accelerator requires considerably more cognitive effort: it is necessary to individually hover over each instruction to see which wells it transfers liquids between, whilst remembering what has been

previously added to each well; the task exceeds one's working memory, making it necessary to resort to an external aid such as writing a table on paper.

Writing

Now consider the task of trying to *write* this protocol. With Wet Lab Accelerator each command corresponding to a liquid transfer must be told which physical wells to transfer liquid between. Until the locations are specified, the intention behind a "transfer" instruction is unclear; this means it is necessary to decide locations as each instruction is added.

With the `List of liquids` interface, the user can still specify the location of each list of liquids as they add the operation that creates it, but they do not have to—they can instead specify what combinations of liquids to create first, before deciding where each should be positioned. This allows them to delay these decisions, and initially focus only on the higher level structure of the protocol.

This approach also eliminates a class of errors: since the low-level instructions specifying which wells to transfer between are generated by a computer, it is impossible for the user to accidentally enter a set of instructions that refer to the incorrect locations (which could occur due to mis-typing a location, making an off-by-one error in addressing, or by simply becoming confused about what is where).

Editing

The user can change the location of a list of liquids, and have this change automatically propagate throughout the protocol. In contrast, in Wet Lab Accelerator the user would have to separately adjust every instruction that referred to this list of liquids (either to create it or use it).

6.3 THE `list of liquids` EDITOR

As an initial proof of concept, we have created a prototype `List of Liquids` editor. This section describes details of this editor's implementation and user interface, as distinct from the diagrammatic notation described in Section 6.2, which could be implemented in other tools.

6.3.1 *Interactivity*

Defining a reference: *Containers*, *pipettes*, and *resources* can be added by clicking on the 'add' link under the appropriate list. Containers and pipettes can be added at any time: before adding resources, as the diagram is being drawn, or after all operations have been added.

Adding a resource node: When a resource is added, a corresponding node is added to the diagram. Additional nodes can be created by dragging the name of the resource from the list of resources and dropping it on the diagram.

Deleting a node: Any node can be deleted by right-clicking on it, and selecting 'Delete' from the context-menu. An exception is the final node corresponding to a resource, which cannot be deleted. Child nodes of the deleted node, and edges attached to deleted nodes, are also deleted.

Adding or deleting an operation: The user can click on a node and drag onto another to indicate that they should be combined in a new location; this creates nodes corresponding to the combination operation and the resulting object. If the shift key is held down when the mouse is released, this will be interpreted as meaning that the node that was *dragged from* should be *added to* the node that was *dragged to*, rather than both being transferred to new wells. Right-clicking on the node representing the operation opens a context menu that allows the combination type to be changed between `zip` and `cross`.

The context menu can also be used to add nodes representing operations that act on only one node, such as performing an operation other than liquid-handling (e.g., thermocycling) or transfering

something from one location to another without combining with anything else (taking an aliquot, spreading onto agar, or picking colonies). Some operations act on entire containers, rather than particular wells: this is indicated by a rectangle surrounding the node representing the process. Dragging between two nodes representing operations of the same type acting on lists of liquids located in the same container merges them: they are moved into the same rectangle, and changing the options of one in the panel updates the other. When reading the diagram the nodes should be interpreted as representing the list of liquid resulting from performing the corresponding operation on the input list of liquids.

Editing an operation: Clicking on a node selects it, highlights it, and displays its details in an information panel through which they can be edited. For nodes representing a list of liquids, this displays the contents of each element of the list, which cannot be directly edited. For processing operations, this allows the relevant parameters to be edited (e.g., temperatures and durations for thermo-cycling).

Editing link details: Clicking on an edge (or its arrowhead, which provides a larger target) selects it and displays its details in the side panel. It is also possible to change the parent of an edge by right-clicking, selecting 'Change parent' from the context menu, and then clicking on the node that should become the parent.

Creating iteration: after clicking on the "Select nodes to repeat" button, clicking on a node toggles whether it is selected, and multiple nodes can be selected. Once the selection is complete, the iteration can be created by clicking the "Repeat" button. The number of times the operations are repeated can be changed by right-clicking on the rectangle representing the repetition.

Copying part of a diagram: after clicking on the 'select nodes to copy' button, clicking on a node toggles whether it is selected, and multiple nodes can be selected. Clicking on the 'Copy' button then copies each of these nodes, and every link that is incident to one of these nodes. This makes it easy to specify that part of a protocol should be repeated, for example with both experimental samples and controls.

Assigning to containers and wells: The information panel that opens when a node is clicked allows the corresponding container to be selected from a list populated with all the containers that have been defined.

Once a node has been assigned to a container, the specific wells in which it will be located can be set by either clicking on "Show or set well locations" in the information panel for that node, or "Well locations" beside the name of the container in the list of containers. Either of these will open a model window showing the container's wells on the left, and a list of liquids assigned to the container on the right (Figure 6.10). Lists of liquids corresponding to a node are grouped together and assigned the same colour; the constituent liquids are displayed as a numbered list, and their components listed.

A single numbered liquid can be dragged from the list and dropped onto a well. Alternatively, a coloured bar can be dragged to simultaneously assign the entire list of liquids corresponding to a node. In the latter case, clicking on the appropriate icon determines whether these will be placed in a row, column, or rectangle.

Exporting protocols: Above the diagram are a series of links to download the protocol in various formats. If details required to convert the protocol into the desired format are missing (e.g., if a transfer does not have a pipette specified) then a modal dialog will appear listing the errors, and the corresponding parts of the diagram will be highlighted.

6.3.2 *Implementation*

The graphical interface is implemented as a web application; justification for this decision is provided in Section 1.6. The back-end is written in Python using the Flask framework [133], and by default uses a SQLite database for data storage; the front-end is written in JavaScript using the D3.js library [17, 18].

The front-end represents the protocol as a graph: there is a list of `node` objects, a list of `link` objects, and a list of `group` objects. Each `node` object has an `label` (the name that is written on the diagram),

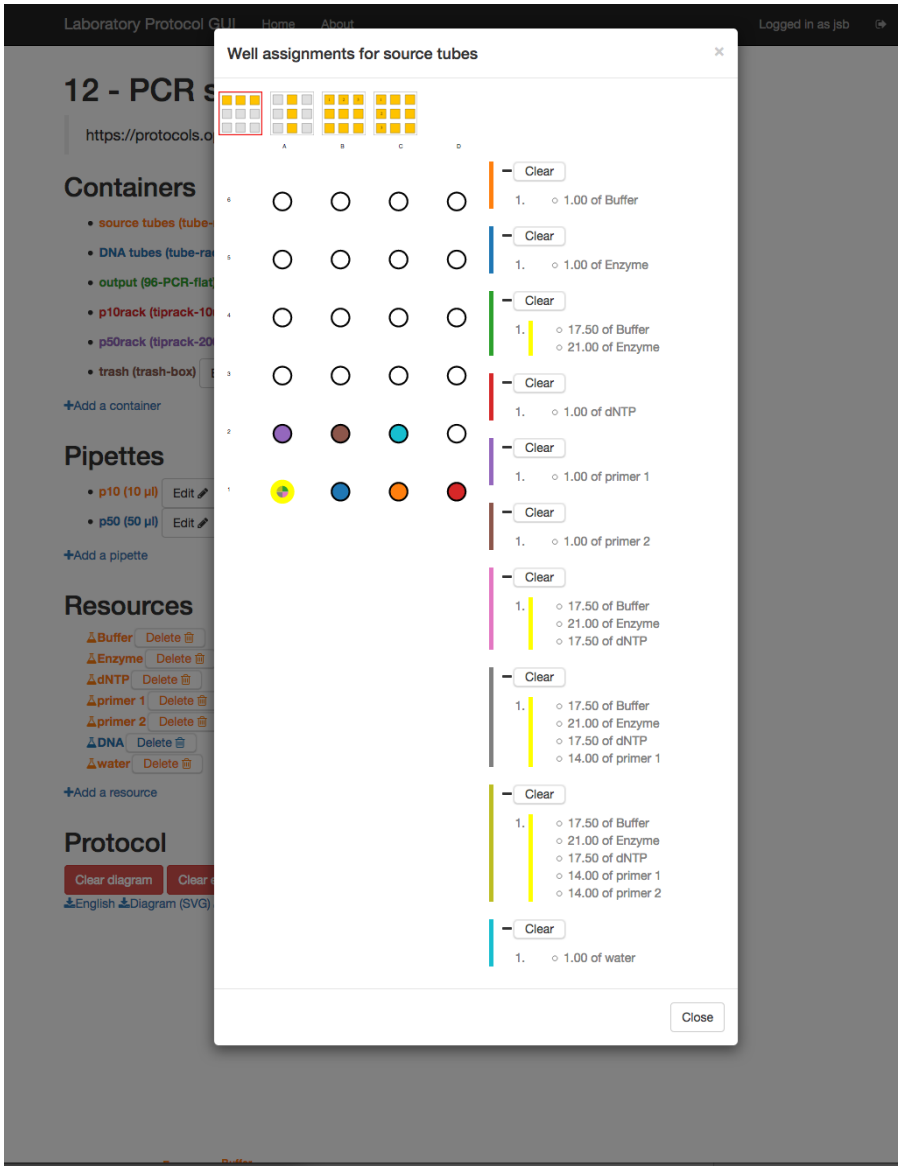


Figure 6.10: Screenshot of `List of Liquids`, showing the modal dialog for assigning liquids to specific wells on a container. The cursor is hovering over one well, causing both it and the corresponding entries in the list on the right (which list its contents at different stages of the protocol) to be highlighted in yellow.

an `x` and `y` position, and a `data` attribute. Each `link` object has a `source_id` and `target_id` attribute, and several attributes recording details of *how* the corresponding operation is to be performed: the transfer volume, which pipette is to be used, whether this is the list of liquids to which another list of liquids is to be added (and, if not, whether this list of liquids is the first to be added to the location where mixing will occur), whether pipette tips must be changed between each transfer, whether the liquid for multiple transfers can be aspirated at once, whether to ensure all liquid is expelled from the pipette with a blowout, and what mixing should be performed. The list of `groups` is used to group distinct nodes corresponding to the same operation acting simultaneously on different lists of liquids in the same container.

Some information is also recorded that is not part of this graph: there are separate lists of pipettes (recording the label, number of channels, min/max volume, and default aspiration and dispensing speeds for each), resources (recording the label, location, and number of wells), and containers (recording the name, type, location on the deck of the liquid handling robot, and which index of which list of liquids has been assigned to each well).

These lists (`node, links, groups, pipettes, resources`) are combined into a single object and serialised to a string in JSON format that is passed between the front-end, back-end and database as required.

The back-end allows a user to register for an account, log-in, and create or delete projects. They can view projects that they created, or have been marked as public. When the page for a particular project is requested, the saved state of the protocol is fetched from the database and included in the page returned as a JSON string. When the page has loaded, this is used to restored the state of the diagram. The JavaScript front-end makes asynchronous requests to the back-end in response to several user actions, including to save a protocol after editing, check that all lists of liquids have been assigned physical locations before the protocol is requested to be exported, and calculate

what is contained in each list of liquids. The set of routes exposed by the back-end is listed in Table 6.1.

6.3.2.1 *Graph layout*

In order to maximise the readability of the rendered diagrams, `List of Liquids` imposes a number of constraints on the network layout. We use the `cola.js` library [35] for network layout, because it provides flexible handling of constraints, and reaches equilibrium faster than implementations of alternative force-based layout algorithms (such as Mike Bostocks' `d3-force`, which uses a velocity-Verlet integrator). `cola.js` performs stress majorization subject to constraints using the Incremental Procedure for Separation Constraint Layout of graphs (IPSep-CoLa) algorithm [36]; for a more detailed discussion of graph drawing, see Appendix B.

Based on our experience of drawing a number of example protocols (Appendix C), we chose the following set of constraints to impose:

1. All nodes must be located in the **viewable** area of the Scalable Vector Graphic (SVG): this is achieved by creating fixed 'dummy' nodes at the upper left and bottom right corner of the SVG which are not drawn, and applying `separation` constraints between these dummy nodes and each actual node. These constraints keep each node above and to the right of the bottom left node, and below and to the left of the top right node.

2. All arrows point **down**. This is achieved by adding a constraint for each edge that imposes a minimum vertical separation between the edge's source and target nodes.

3. If two nodes correspond to the same operation being performed on a container containing multiple lists of liquids, then the nodes are **horizontally aligned**.

4. All solid arrows point **right** (because the corresponding liquid remains in the same location, or is transferred to its new location before the other liquid with which it is being mixed).

Route	Description
/about/ (GET)	About page
/logout/ (GET)	Login
/register/ (POST,GET)	New user registration
/protocols/ (GET)	View list of protocols
/protocols/<int:protocol_id>/ (GET)	View single protocol
/protocols/add (POST,GET)	Create protocol
/protocols/<int:protocol_id>/edit (POST,GET)	Edit details of a protocol (title, description, isPublic)
/protocols/<int:protocol_id>/delete (POST,GET)	Delete protocol
/protocols/<int:protocol_id>/copy (POST,GET)	Copy an existing protocol
/protocols/<int:protocol_id>/english (GET)	Export protocol in English
/protocols/<int:protocol_id>/opentrons (GET)	Export protocol in OpenTrons
/protocols/<int:protocol_id>/autoprotocol (GET)	Export protocol in AutoProtocol
/protocols/<int:protocol_id>/**checkWellsAssigned** (POST,GET)	Check all aliquots in protocol have location assigned
/protocols/<int:protocol_id>/**contents** (POST,GET)	Calculate the composition of each aliquot
/protocols/<int:protocol_id>/**save** (POST)	Saved edited protocol to database
/protocols/static/<path:filename>(GET)	Fetch a file that is static (i.e., not dynamically generated in response to the request)

Table 6.1: Routes provided by the `List of Liquids` editor. Actions that edit the contents of the database require an HTTP POST, rather than GET, request. Requests to most of these routes are made by the user clicking on links, but requests to several (indicated in **bold**) are made by the front-end of the editor in order to save the state of the protocol after editing, or do some processing of the protocol graph.

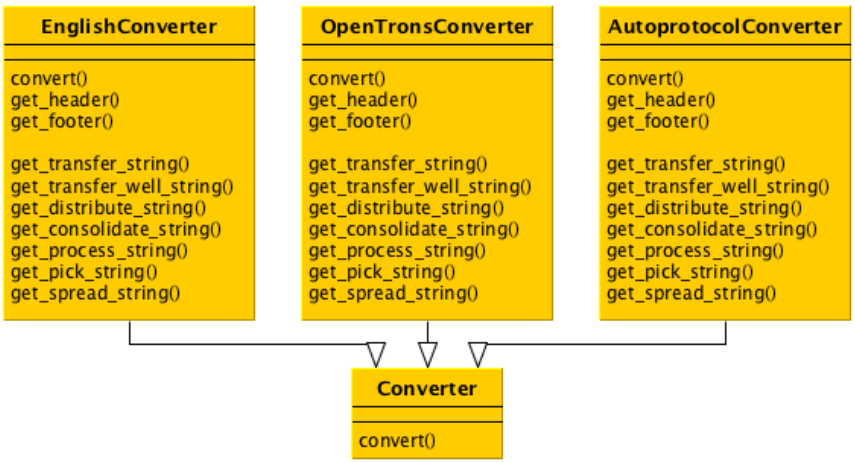


Figure 6.11: The abstract `Converter` class is subclassed by a separate subclass for each output format.

5. All dashed arrows point **left**, but this constraint is applied only if the target node also has an incident solid arrow.

6. If a node has two dashed arrows pointing towards it, and one is a resource, then the arrow from the resource points **right**.

Constraint 1 requires there to be a consistent direction for the flow of time; the choice of which direction should represent time is somewhat arbitrary, but top-to-bottom is commonly used (e.g., in flowcharts, or chronologically ordered lists).

Constraints 4–6 are intended to ensure that regularities in the structure of protocols are reflected by regularities in the structure of the corresponding diagram. For example, in Figure 6.8(d) the overall flow of the protocol is a diagonal from top left to bottom right, with new regents that are being added from the top right and being removed to the bottom left.

6.3.2.2 *Exporting protocols*

Generation of a protocol from a diagram is performed by the abstract `Converter` class, which is subclassed by a class specific to each output type (English text, OpenTrons, AutoProtocol): see Figure 6.11. Each of these provides methods to construct an appropriately formatted string for each command type, as well as the necessary header and

footer strings to be outputted before and after the converted list of instructions.

A topological sort is performed on the protocol graph to determine an order in which operations can be performed. This sorting ensures that:

- An operation is not performed until all the actions needed to produce the lists of liquids that it consumes have been performed.
- An operation that would modify a list of liquids by adding another list of liquids to it is not performed until all other operations that consume that list of liquids have been performed.

Instructions corresponding to each node are then generated in order. If the instruction is a `zip` or `cross`, a list of individual transfers is generated, and then an attempt is made to aggregate these using two heuristics:

- If a multi-channel is pipette is available, and the same volume is transferred from each well in a row to each well in another row, then these individual transfers are aggregated into a single multi-channel transfer.
- If there are several transfers from the same well, then these are aggregated into a single `distribute` instruction if the options indicate that this is permitted.

6.4 EVALUATION

The 'Cognitive Dimensions of Notations' framework [55] established by Thomas Green provides a set of 'discussion tools, descriptions of the artifact-user relationship, intended to raise the level of discourse' about user interfaces [56]. Considering `List of Liquids` in these terms, we could say that it: helps to avoid *premature commitment* by allowing the user to defer the choice of where each operation will take place; decreases *viscosity* by easily accommodating changes in the number of samples or well locations; reduces *error-proneness* by

eliminating errors due to typing mistakes or indexing errors in locations; eliminates the *hard mental operations* of keeping track of the contents of each well as the protocol progresses; and provides a form of *progressive evaluation* by showing the composition of a list of liquids when the corresponding node is selected. One area of weakness is *visibility*, as some details of operations (e.g., what mixing to perform after a liquid transfer) are not shown on the diagram itself, and are displayed only in the information panel that appears when the corresponding node is selected. This could be addressed by replacing the node name with a small form displaying these details, at the cost of increasing the size of the diagram.

6.5 LIMITATIONS AND FUTURE WORK

6.5.1 *Support for more complex protocols*

As a protocol becomes more complex, editing it can become increasingly cumbersome. In a protocol like that shown in Figure 6.12 there is a lot of repeated structure, but each node or link must be edited individually. It would therefore be useful to be able to perform editing operations on a selection consisting of a set of nodes, rather than on individual nodes. For example, if all six of the `Incubate` nodes could be selected, then right-clicking and selecting `Pick` from the context menu could add a separate `pick` node as a descendant of each, and the properties of these newly-created nodes could be edited simultaneously.

An alternative approach is to introduce a new node type that does not represent a physical operation, but rather the logical combination of multiple lists of liquids into a single list of liquids. In the example shown in Figure 6.12, there could be one such node that has all of the green spread nodes as parents, indicating that although each of the three lists of liquids was produced in a different way, they are to be subsequently processed in the same way[6].

6 This provides another example of how a `List of Liquids` diagram might not be a tree, but merely a Directed Acyclic Graph, precluding the use of a layout algorithm specialised for trees.

This example also highlights a second kind of repetition: as well as repetition of *structure* there is repetition of *values*: for example, the same volume (20 µl) is used of each kind of DNA (engineered plasmid, positive control, negative control). In Wet Lab Accelerator, this was represented as a parameter `DNA_volume`, both helping to avoid errors (by ensuring that these volumes are equal) and facilitating changes (as only one value now needs to be adjusted). `List of Liquids` could be extended to allow parameters to be defined as top-level objects (alongside resources, containers, and pipettes), and then used in place of numeric values in any field of the details panel for any node or edge.

Iteration

`List of Liquids` currently supports only a limited form of iteration, in which the repeated operations do not perform liquid transfers, and have only a single list of liquids as an 'input'.

However, some protocols involve iterated actions that do not fit this restriction (e.g., "wash beads three times with 20 µL of buffer, discarding the waste buffer after each wash"). When written as a list of actions involving wells, this involves repetition of the same sequence of instructions (transfer 20 µL of buffer to each well, then transfer 20 µL from each well to the liquid waste container), and so can be concisely written as a loop. However, the `List of Liquids` interface represents all intermediate lists of liquids that are created as a protocol is executed with a separate node, and it is unclear how to unambiguously eliminate the references to each intermediate state.

One possible solution (Figure 6.13) would be to decompose the protocol into sub-modules with ports, so that the output of some modules could be attached to the inputs of others. A connection could be labeled with a number of repeats, indicating that the protocol surrounded by this loop should be repeated that number of times, with the output of each iteration being fed back as the input to the next.

However, whilst reducing the number of nodes and edges in the diagram, such a convention would make the semantics more complex,

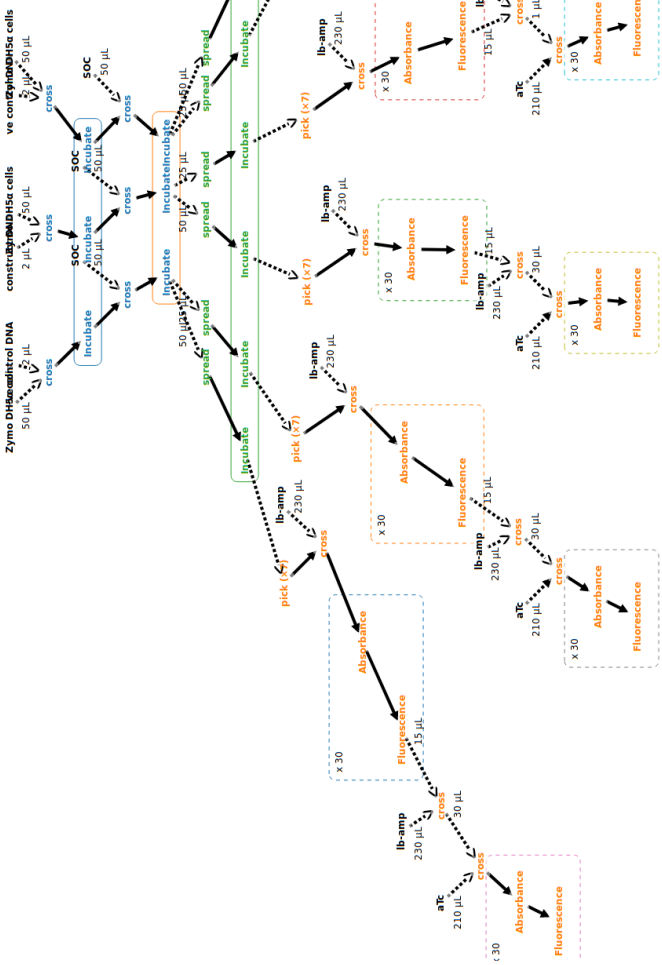


Figure 6.12: Screenshot of `List of Liquids` editor displaying a more complex protocol. Whilst this view allows the structure of the protocol to be understood, imperfections in the automatically generated graph layout become apparent, and editing starts to become cumbersome.

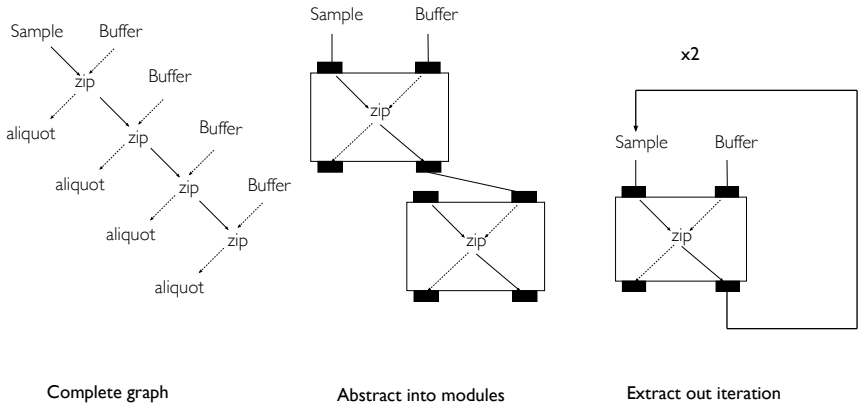


Figure 6.13: One way in which `List of Liquids` could be extended to handle iteration: the protocol could be partitioned into submodules, with a feedback connection labeled by the number of iterations.

and it is not obvious that the result would be an improvement. It is difficult to determine what proportion of protocols would benefit, as there is no comprehensive database of experimental protocols.

Note that in some software systems iteration is commonly used to represent transfers that can be directly encoded as a `zip` or `cross` in `List of Liquids`. However, these iterations are often of a qualitatively different nature, as each stage can be performed in parallel rather than sequentially: each iteration does not rely on changes to the system's state having been made by previous iterations.

Comments

Several existing systems allow the user to provide some kind of textual annotation to particular steps. These may take the form of a description attribute on each instruction block (as in Wet Lab Accelerator, Figure 6.3), or separate blocks that contain only comments (as in EVOWare, Figure 6.4). Comments are probably less important in `List of Liquids`, as the representation used makes the overall structure of protocols clearer, but they could be added as a feature. A rectangle could be drawn enclosing one or more steps, and labelled with a comment, in the same that rectangles representing iteration are labelled with the number of repeats.

Additional output formats

`List of Liquids` is already capable of generating output code to integrate with OpenTrons and AutoProtocol. However, support could be added for additional languages, such as the Krill language for Aquarium and the Antha for AnthaOS (see Section 6.1.3).

Textual interface

`List of liquids` currently provides a graphical editor that can export code to control a specific robotic platform as text. However, many protocols have already been written as text, and `List of liquids` could be extended to automatically import such protocols (for example, saving time that would otherwise be spent manually converting protocols. This could be done by defining a module that exposes the same interface as the equivalent module controlling a robot, but which records operations as they are requested, rather than actually physically carrying them out, an approach that would be simpler than directly parsing computer programs encoding protocols.

A textual language could be defined that implements the main ideas of the graphical interface (*lists of liquids*, and `zip` and `cross` operations on them). In a textual language it may be harder to *read* a protocol, as the structure that was represented externally by the node-link diagram must now be kept in the user's head, but it may also be easier to *edit* the protocol, as the text can be directly edited without needing to select nodes or use form elements. It would also be possible to exploit the advantages of both textual and diagrammatic representation by presenting them side by side, synchronized so that any edit to one is reflected by a corresponding change to the other.

Representing times

Whilst `List of Liquids` allows a duration to be specified for nodes representing incubation operations, it does not currently provide a way to specify constraints on maximum or minimum times that may elapse between operations. Such constraints may be useful when composing sub-protocols into larger protocols, and when determining which steps can be performed concurrently. These could be represented as labels on vertical lines linking the operations whose relative times are constrained.

6.6 SUMMARY & CONTRIBUTION

The main contributions described in this chapter are:

- identifying limitations in existing systems for expressing experimental protocols
- suggesting a new diagrammatic representation for experimental protocols
- creating a prototype protocol editor implementing this representation
- realistically demonstrating the usefulness of the editor, by directly comparing the representation of the same protocol in `List of Liquids` and an earlier tool (Section 6.2.1), and showing that it is able to encode a number of protocols that are in use (further examples are provided in Appendix C)

The source code, and link to a demo, is available at `https://github.com/jamesscottbrown/list-of-liquids`.

Part IV

CONCLUSION

7

CONCLUSIONS

The focus of this thesis has been on providing visual tools for several tasks in the synthetic biology design-build-test cycle. Specifically, these tools have been intended to enable the expression of temporal logic specifications, the identification of models and parameters that cause these specifications to be satisfied, and the expression of laboratory protocols that could be used to experimentally test the resulting design. This chapter summarises the main conclusions of the thesis, and then outlines some directions for potential further research.

7.1 SUMMARY OF CONTRIBUTIONS

A key contribution of this thesis is the introduction of two diagrammatic representations: the `TimeRails` representation of specifications, and the List of Liquids representation of laboratory protocols. Diagrams in each of these formats could be drawn on a computer using any existing vector graphics application, or using pen and paper. However, we also created prototypes of specialized editors for these diagrams, which are aware of their semantics and so can provide additional functionality (such as synthesizing example trajectories that meet a specification, or showing what mixtures of liquids are represented by a node on a list of liquids diagram). Additionally, these editors are integrated into software that allows the user to actually use them to *complete a task*: the user can synthesize a system that meets a `TimeRails` specification (Chapter 3, Chapter 4) or export a protocol that can be used to control a laboratory robot from `List of Liquids`.

This thesis has presented several types of visualization, which are necessary because thinking about the objects that they represent is otherwise difficult. `TimeRails` (Chapter 2) allows the representation of *sets* of signals: the size of these sets precludes the direct represen-

tation of each individual signal, and forces the introduction of an abstraction. In representing protocols with `List of Liquids` (Chapter 6), we again introduce an abstraction, but not because the number of objects to represent is too large (existing representations directly show each step) but rather because the cognitive load of keeping track of the state of a system after each step is too large. `Bayesian Designer` (Chapter 6) relies on interactivity and multiple coupled representations because the amount of data to present is large (perhaps 100 or 1,000 samples at each generation, and a simulation time-series associated with each), and multi-dimensional (each sample is associated with a parameter vector, a multi-dimensional simulation time-series, a distance/error score, and a model). These data must be presented in a form that enables several tasks: confirming that the sampling procedure has run correctly, as well as understanding the structure of the posterior distribution of the parameters.

INTRODUCTION Chapter 1 gave a brief overview of the field of synthetic biology, and the molecular biology on which it is based. It discussed ways in which circuits in synthetic biology can be modelled mathematically, and existing methods for automatically designing genetic circuits.

CHAPTER 2: FORMAL SPECIFICATIONS This chapter introduced a new diagrammatic notation for formulae in STL, and an interactive editor that can be used to create, edit, and interact with such formulae. In contrast to previous work, the notation proposed in this chapter can represent any Signal Temporal Logic specification in a consistent way.

CHAPTER 3: CRN SYNTHESIS BY SAT MODULO ODE This chapter described a tool that allows a user to automatically synthesize a CRN that meets a specification expresssed using `TimeRails`, by using the method originally described in [23] to formulate an SMT-ODE problem that can be solved by existing solvers. It then extends this method to handle input species whose concentrations have a given profile over

time, enabling the synthesis of CRNs with particular responses to a given input function. The user can either enter a fixed topology, and synthesize only the parameter values, or enter a parameteric *sketch* of the CRN topology and synthesize the topology and parameter values simulataneously.

CHAPTER 4: BAYESIAN DESIGN This chapter described visual analytics tools to visualise the results of parameter estimation performed using ABC-SMC, enabling the investigation of how parameter values affect the behaviour of a system. The chapter demonstrates that the ABC-SMC method can be combined with the robustness degree of a specification expressed in Signal Temporal Logic to estimate the posterior distribution of parameters, *given that a specification is met*. The tool allows a user to visually draw a specification using `TimeRails`, and then visually explore the sets of parameters for which this specification is satisfied.

CHAPTER 5: MODEL COMPARISON This chapter described `sbml-diff`, a tool for visually comparing models of biochemical reaction systems in SBML format. It can be used to visually compare different revisions of the same model, or two or more distinct models. Unlike some existing tools, it is able to represent `rules` and `events`, not just `reactions`. It can produce output showing a range of levels of detail and abstraction. `sbml-diff` is re-used as a component in the system for Bayesian Design described in Chapter 4.

CHAPTER 6: EXPERIMENTAL PROTOCOLS discussed the limitations in existing systems for expressing experimental protocols, and suggests a new diagrammatic representation. It then describes a prototype protocol editor implementing this representation and realistically demonstrates the usefulness of the editor, by directly comparing the representation of the same protocol in List of Liquids and an earlier tool (Section 6.2.1), and showing that this is able to encode a number of protocols that are in use.

7.2 OUTLOOK

EXTENSIONS AND FEATURE ADDITIONS Several of the projects described in this thesis could be further extended. `sbml-diff` could better support comparisons between large SBML models by hiding most of the network, displaying only changes and their neighborhood: this would not only decrease visual clutter, but also reduce the time required to compute a good layout.

As discussed in Section 6.5, the List of Liquids system could be extended to allow comments and iteration that includes liquid transfers, provide support for additional output formats, and improve the representation of identical operations being performed on multiple lists of liquids.

`Bayesian Designer` could be extended to support more complex specifications. In particular, it could support specifications that describe acceptable outputs in response to *multiple* choice of input—for example, requiring that the output be above or below a threshold depending on some Boolean function of the inputs.

The usefulness of the CRN synthesis approach described in Chapter 3 is currently limited by the performance and scalability of SMT-ODE solvers, but this is an active active of research and performance is expected to improve.

FACILITATING USE The tools described in this thesis have had only limited external use. Further action could be taken to make them more useful to the wider community, and increase the impact of the work that has already been done. An obvious first step is to complete the process of publishing the papers describing `List of Liquids` and `Bayesian Designer`.

`sbml-diff` would be most useful if it was directly integrated with the BioModels database. This could be achieved by creating an API that would accept a BioModels identifier (rather than an uploaded file) and diagram options, and would return diagrams for the latest version for the corresponding model, or a diagram showing the changes made between versions. A link to this API endpoint could

then be added to the model view, exposing a graphical representation of model history with a single click. This would make `sbml-diff` a resource that could be browsed, rather than a tool that must be explicitly run (either on a user's local machine or a web form).

An instance `List of Liquids` could be hosted, but this would be equivalent to users running the tool locally. The other projects are not feasible to offer as a service, due either to their computational cost (`Bayesian Designer`, `crn-designer`), or the fact that they are a component that wold typically be integrated into a larger system (`TimeRails`).

TRANSFERABLE IDEAS In addition to the potential for direct continuation and extension of the work described in this thesis, there is potential to apply similar ideas to other areas of synthetic biology. For example, there is currently a lack of diagrammatic tools for the design of finite-state machine circuits, and the recent introduction of RNA sequencing as a tool for 'debugging' synthetic genetic circuits could be facilitated by specialised tools for data visualization. Chapter 5 provided an example of how visual abstraction can make aspects of an SBML model easier to identify, and showed the usefulness of being able to automatically generate several different visual representations of the same circuit; similar visual abstractions could be applied to browsing catalogs of genetic circuit parts such as SynBio Hub. Chapter 2 also touched upon the querying or searching based upon drawing a diagram (specifically, a `TimeRails` diagram representing a temporal property); a tool could also be created to allow the searching of catalogs of genetic circuits based upon diagrams of the topology of sub-circuits.

It should be noted, however, that these fundamental ideas (visual abstraction, query by search, etc) have a long history and are not claimed to originate in this thesis. In contrast, the idea of representing temporal logic specifications using rectangles sliding along rails *is* novel, and could be incorporated into other tools in other domains, either by embedding the `TimeRails` widget or by creating a new equivalent.

A general conclusions is that the development of specialised forms of visual representation for synthetic biology is both possible and useful. To some readers of this thesis, this statement will seem self-evident, but it is not universally accepted.

A

INTRODUCTION TO PROBABILITY[1]

Probabilities can be interpreted in two ways: as statements of the frequency of random experiment, or as statements about the degree of belief in propositions.

The probability of an event A being true is written as $P(A)$. For a discrete random variable x, the probability of x taking the value a is given by $P(x = a) = f(a)$, where $f()$ is the **probability distribution function**. For a continuous random variable x, the probability of x being in a particular interval $a \leq x \leq b$ is defined in terms of the **probability mass function** $f()$ by the integral $P(a \leq x \leq b) = \int_a^b f(x)\ dx$. Both probability mass and distribution function must be non-negative throughout their domains, and be appropriately normalised: $\sum_{x \in \chi} p(x) = 1$ for a p.m.f, or $\int p(x)\ dx = 1.$ for a p.d.f.

In the following, $\mathcal{H}$ represents assumptions on which the probabilities are based (this is sometimes referred to as the **background** or **context** and denoted I).

The **conditional probability** of event A being true given that event B is true is written as $p(x|y)$.

The joint probability of two events is given by the **product rule**:

$$p(x \wedge y|\mathcal{H}) = p(x, y|\mathcal{H}) = p(x|y, \mathcal{H})p(y|\mathcal{H})$$

The **sum rule** states that

$$\begin{aligned} P(x|\mathcal{H}) &= \int P(x, y|\mathcal{H})\ dy \\ &= \int P(x|y, \mathcal{H})P(y|\mathcal{H})\ dy \end{aligned}$$

1 This appendix provides introductory information that is needed to understand Chapter 4.

Together, these two rules imply **Bayes' theorem**:

$$p(x|y,\mathcal{H})p(y|\mathcal{H}) = p(x,y|\mathcal{H}) = p(y|x,\mathcal{H})p(x,\mathcal{H})$$

$$\implies \overbrace{p(y|x,\mathcal{H})}^{\text{Posterior}} = \frac{\overbrace{p(x|y,\mathcal{H})}^{\text{Likelihood}}\overbrace{p(y|\mathcal{H})}^{\text{Prior}}}{\underbrace{p(x|\mathcal{H})}_{\text{Marginal}}}$$

Note that the likelihood is not a probability distribution function, because it is not correctly normalized: $\int p(x|y,\mathcal{H})\, dy \neq 1$.

B

DRAWING GRAPHS, TREES AND NETWORKS[1]

There are three main approaches to visualizing relational data:

- Nodes can be represented as glyphs or points, with edges represented by *connections* between these (node-link diagrams)
- The adjacency *matrix* can be visualized. This is matrix A in which the element A_{ij} corresponds to connections from node i to node j.
- In the case of trees, nodes can be represented by shapes, which are *enclosed* by shapes representing the parent nodes of the corresponding nodes. Shapes may be scaled in proportion to quantities associated with each node (treemaps).

Each of these options has advantages and disadvantages. Node-link diagrams can be effective in showing topology, but for large graphs they can degenerate into an uninterpretable 'hairball' of overlapping lines. Matrix visualisations avoid the occlusion problems of node-link diagrams, but which patterns are visible is highly dependent on the ordering of nodes that is chosen. Treemaps can only represent tree structures.

LAYING OUT NODE-LINK DIAGRAMS

There are many ways to arrange the nodes in a node-link diagram. The development of such methods is a major field of study, with its own conference (GD: the International Symposium on Graph Drawing & Network Visualization) and textbooks [12 📖, 77 📖, 149 📖].

The nodes can be arranged in three-dimensional space (but the third dimension is of limited use due to problems such as occlusion),

1 This appendix provides background information for Chapter 4, Chapter 5, and Chapter 6.

in two-dimensional space on a plane, or in some lower-dimensional subspace (most commonly in a line or circle, yielding a Circos plot or arc diagram).

To emphasise hierarchical structure in a directed graph, nodes can be first assigned to rows in a way that maximises the proportion of edges that are directed downwards, and then positioned horizontally within their row in a way that minimises edge-crossings (the Sugiyama method).

A layout in which the position of each node is the barycentre of its neighbours (barycentric embedding/Tutte embedding) can be found by solving a linear system of equations, assuming that the graph is planar [153]; if the graph is not planar, then it is by definition impossible to construct any two-dimensional layout in which no edges cross.

Another approach is that of *force-directed layouts*, which adopt a physical metaphor and define an energy function that includes attractive forces between pairs of nodes that are linked by edges and/or repulsion between either all nodes or all nodes that are not connected. An arrangement of nodes that minimises the corresponding energy is then sought, either by optimization, or by simulating the evolution of the nodes as if they were particles acted on by the energy function using numerical integration. In the latter case, the nodes are assigned random initial positions, and the system is allowed to evolve until it has either reached equilibrium or the number of iterations has reached a threshold. The layout is not deterministic, as the initial condition is random and the system typically gets stuck in a local optimum, rather than reaching the global optimum. Additionally, the final layout depends on the choice of values for several parameters.

Building upon previous work by Eades (which used $f_a(d) = k_a \log d$ $d_r(d) = k_r/d^2$), Fruchterman and Rhinegold [45] used a position updating rule of this form:

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t - \sum_{j \neq i} \frac{\boldsymbol{x}_j^t - \boldsymbol{x}_i^t}{||\boldsymbol{x}_j^t - \boldsymbol{x}_i^t||} f_r(||\boldsymbol{x}_j^t - \boldsymbol{x}_i^t||) + \sum_{j \in N(i)} \frac{\boldsymbol{x}_j^t - \boldsymbol{x}_i^t}{||\boldsymbol{x}_j^t - \boldsymbol{x}_i^t||} f_a(||\boldsymbol{x}_j^t - \boldsymbol{x}_i^t||)$$

$$f_a(d) = d^2/k$$

$$f_r(d) = -k^2/d$$

This corresponds to Euler integration of a system which is physically inspired, but not quite physical: to avoid oscillations, the 'forces' f_a and f_r directly affect the velocity $\dot{\boldsymbol{x}}_i$ rather than the acceleration $\ddot{\boldsymbol{x}}_i$.

Some other force-directed layout algorithms do use physically correct dynamics in which forces affect accelerations in accordance with Newton's second law ($\boldsymbol{f} = \frac{d(m\boldsymbol{v})}{dt}$); for example `d3-force` simulates a system with Newtonian mechanics by numerically integrating using the velocity Verlet method[2].

Kamada and Kawai [75] use the Newton-Rhapson method[3]to locally minimize the global stress, which is defined as

$$\text{stress}(X) = \sigma(X) = \sum_{i<j\leq n} w_{ij}(||X_i - X_j|| - \delta_{ij})^2$$

where δ_{ij} is the ideal distance between nodes i and j, usually chosen as the graph-theoretic distance (that is, the number of edges in the shortest path connecting them), and $w_{ij} = \delta_{ij}^{-\alpha}$ is a normalization constant; usually $\alpha = 2$.

At each iteration, they chose the particle j as

$$j = \arg\max_i \left(\frac{\partial \sigma}{\partial x_i}\right)^2 + \left(\frac{\partial \sigma}{\partial y_i}\right)^2$$

and use the Newton-Rhapson method to find a new position for this node (x_j, y_j) such that $\frac{\partial \sigma}{\partial x_j} = \frac{\partial \sigma}{\partial y_j} = 0$.

2 The velocity Verlet method integrates the system $\dot{\boldsymbol{x}}(t) = \boldsymbol{v}(t) = \boldsymbol{a}(\boldsymbol{x}t))$ using the formulae:

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t) + \boldsymbol{v}(t)\Delta t + \frac{1}{2}\boldsymbol{a}(t)\Delta t^2$$

$$\boldsymbol{v}(t + \Delta t) = \boldsymbol{v}(t) + \frac{\boldsymbol{a}(t) + \boldsymbol{a}(t + \Delta t)}{2}\Delta t$$

3 Suppose that we want to find a root of a function of n variables, $F(x_1, \ldots, x_n)$. The Taylor series expansion is given by $\boldsymbol{F}(\boldsymbol{x} + \delta\boldsymbol{x}) = \boldsymbol{F}(\boldsymbol{x}) + \boldsymbol{J} \cdot \delta\boldsymbol{x} + O(\delta\boldsymbol{x}^2)$, where $\boldsymbol{J}$ is

Stress majorization [48] minimises the same stress function using the method of majorization, which bounds it with a series of functions $F^Z(X) \geq \text{stress}(X)$ that achieve equality when $Z = X$, and iteratively takes $X(t+1)$ as the minimizer of $F^{X(t)}(X)$. The bounding function is defined as

$$F^Z(X) = \sum_{i<j} w_{ij} d_{ij}^2 + \text{Tr}(X^T L^w X) - 2\text{Tr}(X^T L^Z Z)$$

where the weighted Laplacian L_{ij}^w is defined as $\sum_{k \neq i} w_{ik}$ for $i = j$, and $-w_{ij}$ for $i \neq j$. By differentiating and equating to zero, it is found that the minimum of $F^Z(X)$ is given by $L^w X = L^Z Z$, and so in the absence of constraints stress majorization reduces to the iterative solution of a series of linear equations.

Dwyer *et al.* [36] introduced an algorithm for efficiently applying stress majorization subject to constraints IPSep-CoLa. A generic solver can still be used to solve the problem if constraints are applied, but will take longer. The solution at one iteration is known to be a feasible solution to the next problem and, since the two problems are similar, is likely to be close to its solution. However, interior point solvers are generally not compatible with using a guessed solution as a starting point for a 'warm-start' optimization. IPSep-CoLa therefore uses the *gradient-projection* algorithm, which alternately performs a gradient-descent step, followed by a projection step that projects back onto the feasible region. This projection step is computationally simple to perform, because of the types of constraints that it supports.

---

the Jacobian matrix $\boldsymbol{J}_{ij} = \frac{\partial F_i}{\partial x_j}$. Neglecting terms of order higher than $\delta \boldsymbol{x}$ and equating $\boldsymbol{F}(\boldsymbol{x} + \delta \boldsymbol{x}) = 0$ gives $\boldsymbol{J} \cdot \delta \boldsymbol{x} = -\boldsymbol{F}$. At each iteration, the Newton-Rhapson method solves this linear system, and applies the correction $\delta \boldsymbol{x}$ to obtain an improved estimate of the root [122 📖, Section 9.6].

C

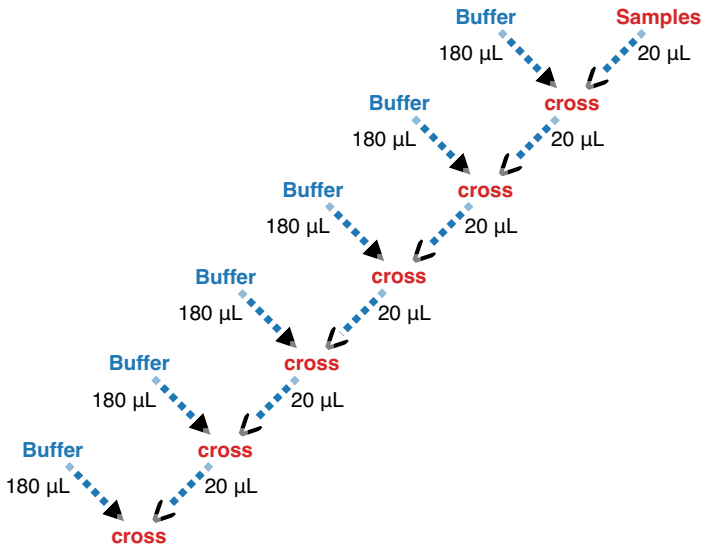
EXAMPLE PROTOCOLS[1]

This section provides examples of how a range of protocols (based on those in the OpenTrons Protocol Library[2]) can be represented in the List of Liquids system.

The diagrams and OpenTrons Python code are exactly as exported from List of Liquids. In some cases, this code is quite verbose: the focus of work so far has been on visual representation and user interface, but we intend to apply further optimizations during code generation.

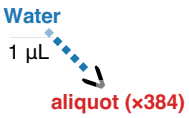
1 This appendix is a supplement to Chapter 6.
2 http://protocols.opentrons.com/

C.0.1 *Serial Dilution*



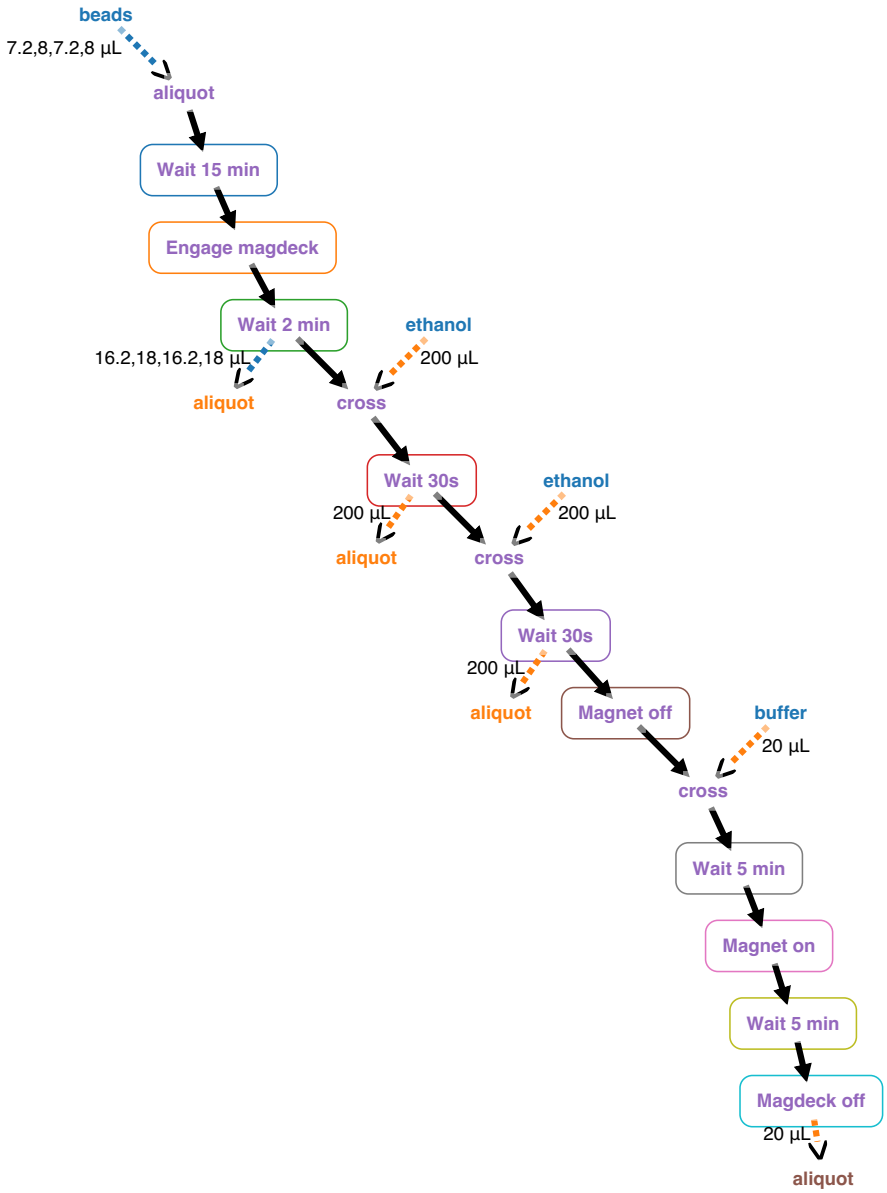
Generated OpenTrons python code

C.0.2 *Filling a 384-well plate*



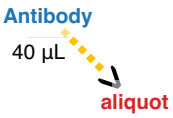
Generated OpenTrons python code

C.0.3 *DNA Extraction using Magnetic AMPureXP beads*



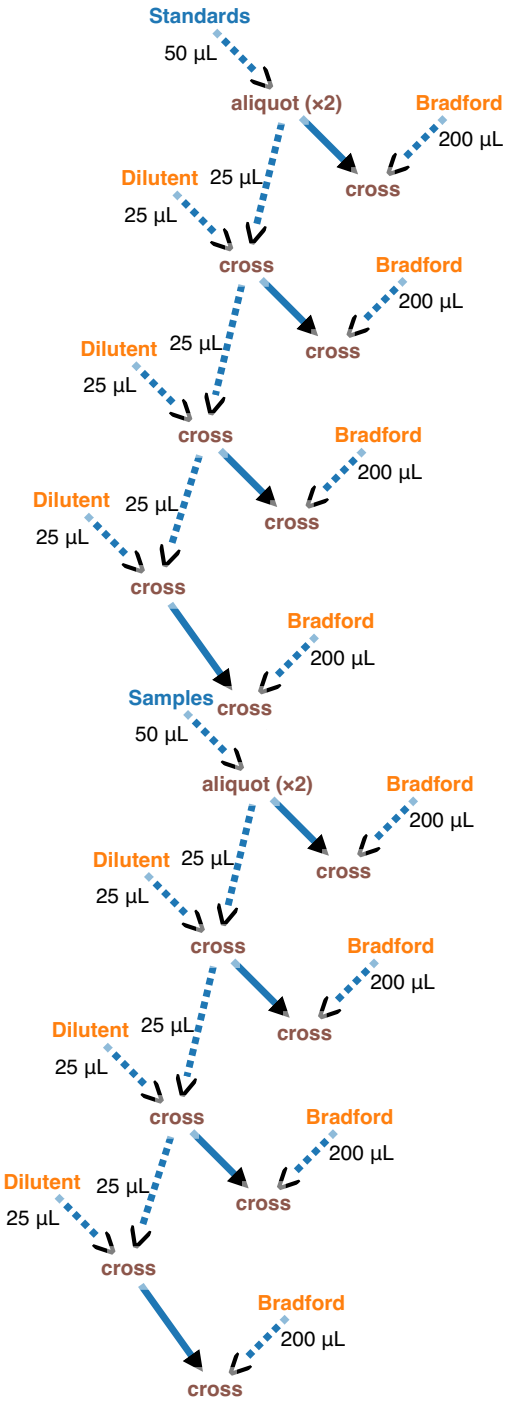
Generated OpenTrons python code

C.0.4 *Transfer from tube rack to 384-well plate*



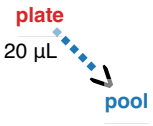
Generated OpenTrons python code

C.0.5 *Bradford Assay*



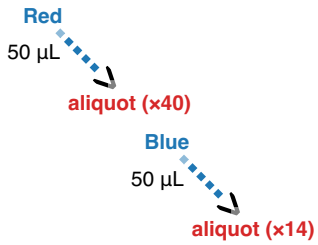
Generated OpenTrons python code

C.0.6 *Consolidate from 96-well plate into tube*



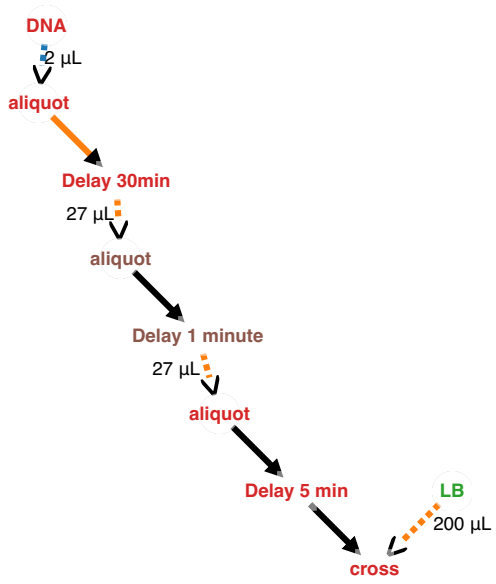
Generated OpenTrons python code

C.0.7 *Draw a dinosaur in a 96-well plate*



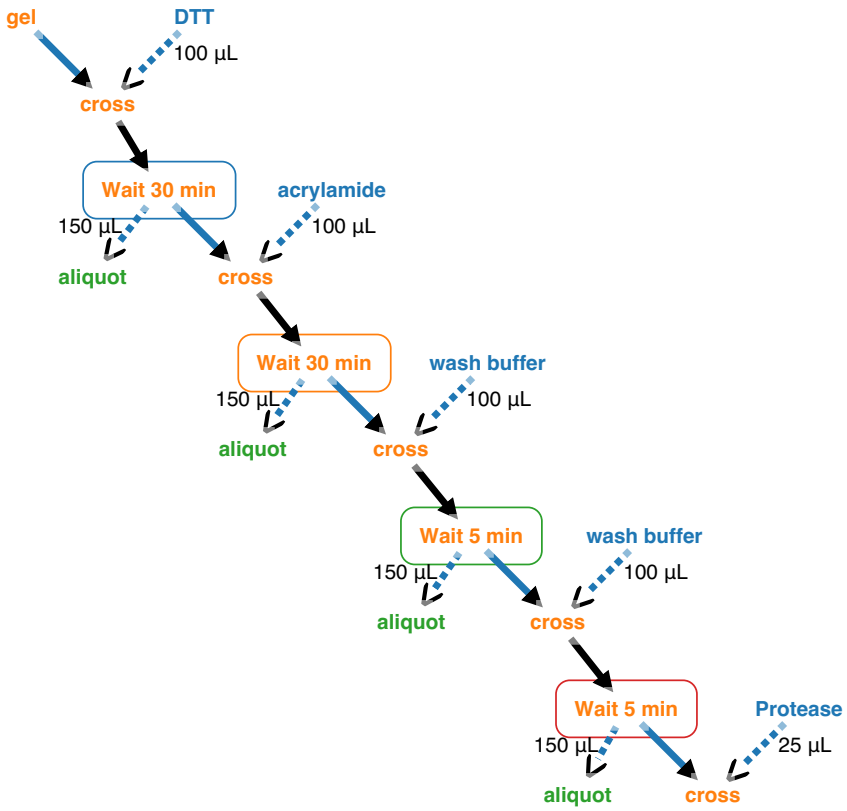
Generated OpenTrons python code

C.0.8 *Heat shock transformation*



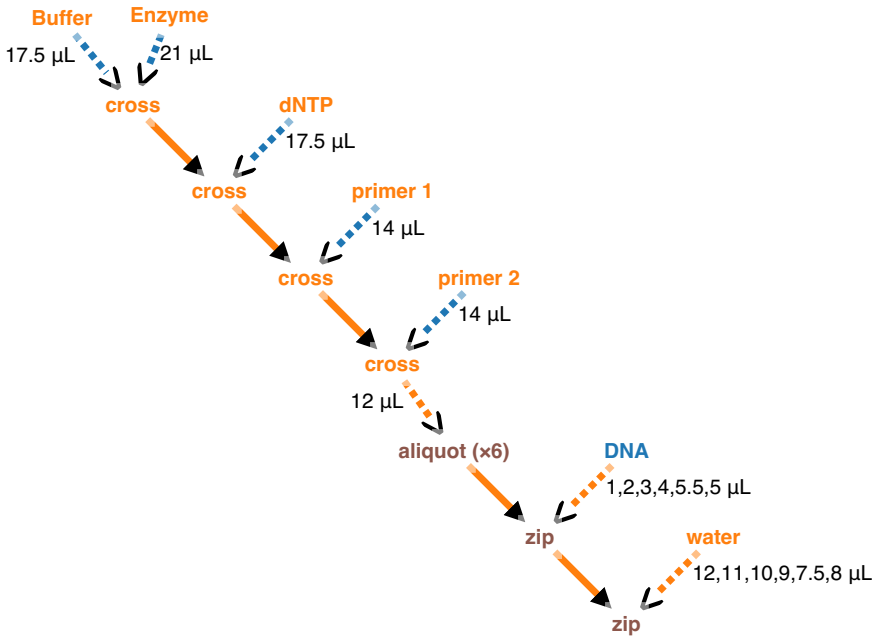
Generated OpenTrons python code

C.0.9 *In-gel digest*



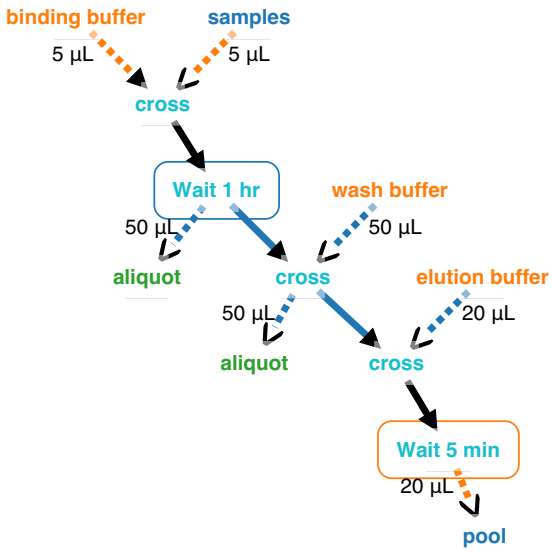
Generated OpenTrons python code

C.0.10 *PCR setup*



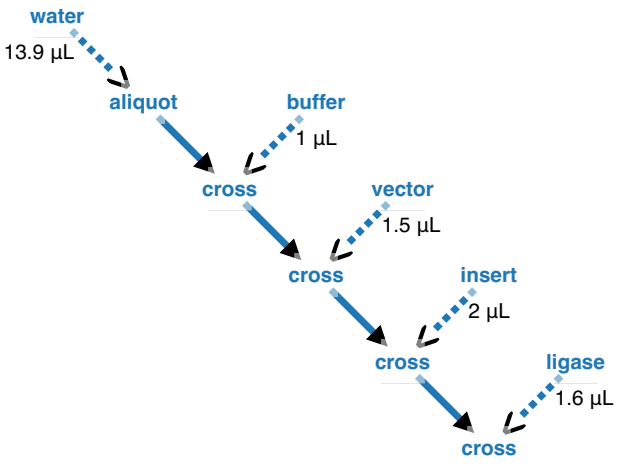
Generated OpenTrons python code

C.0.11 *Purification using SequalPrep kit*



Generated OpenTrons python code

C.0.12 *T4 DNA ligation*



D

EXAMPLE TEXTUAL DIFF BETWEEN SBML MODELS[1]

This is the output of the command `diff toggle.xml repressilator.xml` (for concision, an initial section corresponding to "<notes>" has been removed):

Listing D.1: Textual diff between SBML models of a repressilator and a toggle switch.

```
77c323,341
<
---
>             </species>
>       <species metaid="PZ" hasOnlySubstanceUnits="true"
    initialAmount="0" sboTerm="SBO:0000252" compartment="cell"
     name="cI protein" id="PZ">
>         <notes>
>           <p xmlns="http://www.w3.org/1999/xhtml">
>         lambda repressor</p>
>
>         </notes>
>         <annotation>
>         <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf
    -syntax-ns#" xmlns:bqbiol="http://biomodels.net/biology-
    qualifiers/">
>       <rdf:Description rdf:about="#PZ">
>       <bqbiol:is>
>       <rdf:Bag>
>       <rdf:li rdf:resource="http://identifiers.org/uniprot/P
    03034"/>
>       </rdf:Bag>
>       </bqbiol:is>
>       </rdf:Description>
>       </rdf:RDF>
>       </annotation>
>             </species>
```

1 This appendix provides supplementary information for Chapter 5

```
118c382,401
<
---
>       <species metaid="_905802" hasOnlySubstanceUnits="true"
     initialAmount="0" sboTerm="SBO:0000250" compartment="cell
    " name="cI mRNA" id="Z">
>         <annotation>
>         <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf
    -syntax-ns#" xmlns:bqbiol="http://biomodels.net/biology-
    qualifiers/">
>       <rdf:Description rdf:about="#_905802">
>               <bqbiol:isVersionOf>
>       <rdf:Bag>
>       <rdf:li rdf:resource="http://identifiers.org/chebi/
    CHEBI:33699"/>
>       <rdf:li rdf:resource="http://identifiers.org/kegg.
    compound/C00046"/>
>       </rdf:Bag>
>       </bqbiol:isVersionOf>
>
>       <bqbiol:encodes>
>       <rdf:Bag>
>       <rdf:li rdf:resource="http://identifiers.org/uniprot/P
    03034"/>
>       </rdf:Bag>
>       </bqbiol:encodes>
>       </rdf:Description>
>       </rdf:RDF>
>       </annotation>
>             </species>
387c670,694
<
---
>       <reaction name="degradation of CI transcripts" id="
    Reaction3" metaid="_905862" reversible="false" sboTerm="
    SBO:0000179">
>         <annotation>
>         <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf
    -syntax-ns#" xmlns:bqbiol="http://biomodels.net/biology-
    qualifiers/">
>       <rdf:Description rdf:about="#_905862">
>       <bqbiol:isVersionOf>
>       <rdf:Bag>
```

```
>       <rdf:li rdf:resource="http://identifiers.org/obo.go/GO
    :0006402"/>
>       </rdf:Bag>
>       </bqbiol:isVersionOf>
>       </rdf:Description>
>       </rdf:RDF>
>       </annotation>
>               <listOfReactants>
>         <speciesReference metaid="_421025" species="Z"/>
>         </listOfReactants>
>         <kineticLaw metaid="_421038" sboTerm="SBO:0000049">
>           <math xmlns="http://www.w3.org/1998/Math/MathML">
>             <apply>
>               <times/>
>               <ci> kd_mRNA </ci>
>               <ci> Z </ci>
>             </apply>
>           </math>
>                 </kineticLaw>
>       </reaction>
444c751,778
<
---
>       <reaction name="translation of CI" id="Reaction6"
    metaid="_905923" reversible="false" sboTerm="SBO:0000184">
>         <annotation>
>         <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf
    -syntax-ns#" xmlns:bqbiol="http://biomodels.net/biology-
    qualifiers/">
>       <rdf:Description rdf:about="#_905923">
>       <bqbiol:isVersionOf>
>       <rdf:Bag>
>       <rdf:li rdf:resource="http://identifiers.org/obo.go/GO
    :0006412"/>
>       </rdf:Bag>
>       </bqbiol:isVersionOf>
>       </rdf:Description>
>       </rdf:RDF>
>       </annotation>
>               <listOfProducts>
>         <speciesReference metaid="_421124" species="PZ"/>
>         </listOfProducts>
>         <listOfModifiers>
```

```
>           <modifierSpeciesReference metaid="_421136" species
    ="Z" sboTerm="SBO:0000461"/>
>         </listOfModifiers>
>         <kineticLaw metaid="_421148" sboTerm="SBO:0000049">
>           <math xmlns="http://www.w3.org/1998/Math/MathML">
>             <apply>
>               <times/>
>               <ci> k_tl </ci>
>               <ci> Z </ci>
>             </apply>
>           </math>
>                 </kineticLaw>
>       </reaction>
495c829,853
<
---
>       <reaction name="degradation of CI" id="Reaction9"
    metaid="_905982" reversible="false" sboTerm="SBO:0000179">
>         <annotation>
>         <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf
    -syntax-ns#" xmlns:bqbiol="http://biomodels.net/biology-
    qualifiers/">
>       <rdf:Description rdf:about="#_905982">
>       <bqbiol:isVersionOf>
>       <rdf:Bag>
>       <rdf:li rdf:resource="http://identifiers.org/obo.go/GO
    :0030163"/>
>       </rdf:Bag>
>       </bqbiol:isVersionOf>
>       </rdf:Description>
>       </rdf:RDF>
>       </annotation>
>               <listOfReactants>
>           <speciesReference metaid="_421208" species="PZ"/>
>         </listOfReactants>
>         <kineticLaw metaid="_421220" sboTerm="SBO:0000049">
>           <math xmlns="http://www.w3.org/1998/Math/MathML">
>             <apply>
>               <times/>
>               <ci> kd_prot </ci>
>               <ci> PZ </ci>
>             </apply>
>           </math>
```

```
>                 </kineticLaw>
>       </reaction>
539c897
<                       <ci> PY </ci>
---
>                       <ci> PZ </ci>
600c958,1009
<
---
>       <reaction name="transcription of CI" id="Reaction12"
    metaid="_906042" reversible="false" sboTerm="SBO:0000183">
>         <annotation>
>         <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf
    -syntax-ns#" xmlns:bqbiol="http://biomodels.net/biology-
    qualifiers/">
>       <rdf:Description rdf:about="#_906042">
>       <bqbiol:isVersionOf>
>       <rdf:Bag>
>       <rdf:li rdf:resource="http://identifiers.org/obo.go/GO
    :0006351"/>
>       </rdf:Bag>
>       </bqbiol:isVersionOf>
>       </rdf:Description>
>       </rdf:RDF>
>       </annotation>
>               <listOfProducts>
>           <speciesReference metaid="_421304" species="Z"/>
>         </listOfProducts>
>         <listOfModifiers>
>           <modifierSpeciesReference metaid="_421317" species
    ="PY" sboTerm="SBO:0000536"/>
>         </listOfModifiers>
>         <kineticLaw metaid="_421329">
>           <math xmlns="http://www.w3.org/1998/Math/MathML">
>             <apply>
>               <plus/>
>               <ci> ao_tr </ci>
>               <apply>
>                 <divide/>
>                 <apply>
>                   <times/>
>                   <ci> a_tr </ci>
>                   <apply>
```

```
>                     <power/>
>                     <ci> KM </ci>
>                     <ci> n </ci>
>                   </apply>
>                 </apply>
>                 <apply>
>                   <plus/>
>                   <apply>
>                     <power/>
>                     <ci> KM </ci>
>                     <ci> n </ci>
>                   </apply>
>                   <apply>
>                     <power/>
>                     <ci> PY </ci>
>                     <ci> n </ci>
>                   </apply>
>                 </apply>
>               </apply>
>             </math>
>                 </kineticLaw>
>       </reaction>
```

BIBLIOGRAPHY

[1] Bruce Alberts. *Molecular biology of the cell*. New York, NY: Garland Science, Taylor and Francis Group, 2015. ISBN: *978-0815345244*.

[2] Daniel Alcaide and Jan Aerts. "MCLEAN: Multilevel Clustering Exploration As Network." In: *PeerJ Computer Science* 4 (2018), e145. DOI: 10/gd87q2.

[3] Vaishnavi Ananthanarayanan and William Thies. "BioCoder: A programming language for standardizing and automating biology protocols." In: *Journal of Biological Engineering* 4.1 (2010), p. 13. DOI: 10/bh96t4.

[4] James A. J. Arpino, E J Hancock, James Anderson, M Barahona, Guy-Bart V. Stan, Antonis Papachristodoulou, and K Polizzi. "Tuning the dials of Synthetic Biology." In: *Microbiology* 159.7 (2013), p. 1236. DOI: 10/gd87qz.

[5] Christel Baier. *Principles of model checking*. Cambridge, Mass: MIT Press, 2008. ISBN: *978-0-262-02649-9*.

[6] John A. Ball. "Memes as replicators." In: *Ethology and Sociobiology* 5.3 (1984), pp. 145–161. DOI: 10/b697bt.

[7] C. P. Barnes, D. Silk, and M. P. H. Stumpf. "Bayesian design strategies for synthetic biology." In: *Interface Focus* 1.6 (2011), pp. 895–908. DOI: 10/cvczr4.

[8] C. P. Barnes, D. Silk, X. Sheng, and M. P. H. Stumpf. "Bayesian design of synthetic biological systems." In: *Proceedings of the National Academy of Sciences* 108.37 (2011), pp. 15190–15195. DOI: 10/czk26d.

[9] Clark Barrett and Cesare Tinelli. "Satisfiability Modulo Theories." In: *Handbook of Model Checking*. Ed. by Ed Clarke, Thomas Henzinger, and Helmut Veith. In preparation. 2018.

URL: http://www.cs.stanford.edu/~barrett/pubs/BT14.pdf.

[10] Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. "Satisfiability Modulo Theories." In: *Handbook of Satisfiability*. IOS Press, 2008. URL: https://people.eecs.berkeley.edu/~sseshia/pubdir/SMT-BookChapter.pdf.

[11] Maxwell Bates, Aaron J. Berliner, Joe Lachoff, Paul R. Jaschke, and Eli S. Groban. "Wet Lab Accelerator: A Web-Based Application Democratizing Laboratory Automation for Synthetic Biology." In: *ACS Synthetic Biology* 6.1 (2017), pp. 167–171. DOI: 10/f9ndsq.

[12] Giuseppe Battista. *Graph drawing : algorithms for the visualization of graphs*. Upper Saddle River, N.J: Prentice Hall, 1999. ISBN: *978-0-13-301615-4*.

[13] David Bau, Jeff Gray, Caitlin Kelleher, Josh Sheldon, and Franklyn Turbak. "Learnable programming: Blocks and Beyond." In: *Communications of the ACM* 60.6 (2017), pp. 72–80. DOI: 10/gd87qp.

[14] Bernadette Bensaude-Vincent. "Biomimetic Chemistry and Synthetic Biology: A Two-Way Traffic Across the Borders." In: *Hyle* 15.1 (2009), pp. 31–46. URL: http://www.hyle.org/journal/issues/15-1/bensaude.pdf.

[15] Jeremy Berg. *Biochemistry*. New York: W.H. Freeman & Company, a Macmillan Education Imprint, 2015. ISBN: *978-1464126109*.

[16] Lacramioara Bintu, Nicolas E Buchler, Hernan G Garcia, Ulrich Gerland, Terence Hwa, Jané Kondev, Thomas Kuhlman, and Rob Phillips. "Transcriptional regulation by the numbers: applications." In: *Current Opinion in Genetics & Development* 15.2 (2005), pp. 125–135. DOI: 10/cc924j.

[17] Michael Bostock. *D3.js - Data-Driven Documents*. URL: http://d3js.org/.

[18] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. "D^3: Data-Driven Documents." In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), p. 2301. DOI: 10/b7bhhf.

[19] Robert K. Brayton, Gary D. Hachtel, Curtis T. McMullen, and Alberto L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Springer US, 1984. ISBN: *978-1-4612-9784-0*. DOI: 10/fhk8vw. URL: https://doi.org/10.1007%2F978-1-4613-2821-6.

[20] Stephen P. Brooks. "Markov Chain Monte Carlo method and its applications." In: *The Statistician* 47 (1998). DOI: 10/bw3c5k.

[21] Jennifer A N Brophy and Christopher A Voigt. "Principles of genetic circuit design." In: *Nature Methods* 11.5 (2014), pp. 508–520. DOI: 10/f5294s.

[22] D. Ewen Cameron, Caleb J. Bashor, and James J. Collins. "A brief history of synthetic biology." In: *Nature Reviews Microbiology* 12.5 (2014), pp. 381–390. DOI: 10/f5x7h7.

[23] Luca Cardelli, Milan Češka, Martin Fränzle, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, and Max Whitby. "Syntax-Guided Optimal Synthesis for Chemical Reaction Networks." In: *Computer Aided Verification*. Springer International Publishing, 2017, pp. 375–395. DOI: 10/gd87qc.

[24] Arturo Casini et al. "A Pressure Test to Make 10 Molecules in 90 Days: External Evaluation of Methods to Engineer Biology." In: *Journal of the American Chemical Society* 140.12 (2018), pp. 4302–4316. DOI: 10/gdcqxj.

[25] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. "Programmable chemical controllers made from DNA." In: *Nature Nanotechnology* 8.10 (2013), pp. 755–762. DOI: 10/gd87qf.

[26] Autumn A. Cuellar, Catherine M. Lloyd, Poul F. Nielsen, David P. Bullivant, David P. Nickerson, and Peter J. Hunter.

"An Overview of CellML 1.1, a Biological Model Description Language." In: *SIMULATION* 79.12 (2003), pp. 740–747. DOI: 10/cz4526.

[27] Neil Dalchau, Niall Murphy, Rasmus Petersen, and Boyan Yordanov. "Synthesizing and Tuning Chemical Reaction Networks with Specified Behaviours." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2015, pp. 16–33. DOI: 10/gd87qd.

[28] Madhukar S Dasika and Costas D Maranas. "OptCircuit: an optimization based method for computational design of genetic circuits." In: *BMC Systems Biology* 2.1 (2008), p. 24. URL: 10.1186/1752-0509-2-24.

[29] Jamie Davies. "Using synthetic biology to explore principles of development." In: *Development* 144.7 (2017), pp. 1146–1158. DOI: 10/f94sp5.

[30] Martin Davis, George Logemann, and Donald Loveland. "A machine program for theorem-proving." In: *Communications of the ACM* 5.7 (1962), pp. 394–397. DOI: 10/b4rbrz.

[31] Martin Davis and Hilary Putnam. "A Computing Procedure for Quantification Theory." In: *Journal of the ACM* 7.3 (1960), pp. 201–215. DOI: 10/bw9h55.

[32] M. Omar Din et al. "Synchronized cycles of bacterial lysis for in vivo delivery." In: *Nature* 536.7614 (2016), pp. 81–85. DOI: 10/bmw6.

[33] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. "Metric Interval Temporal Logic Specification Elicitation and Debugging." In: *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*. 2015. DOI: 10/gd87px.

[34] Alexandre Donze and Oded Maler. "Robust Satisfaction of Temporal Logic over Real-Valued Signals." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 92–106. DOI: 10/fpqb5g.

[35] Tim Dwyer. *cola.js: Constraint-Based Layout in the Browser*. http://marvl.infotech.monash.edu/webcola/. [Online; accessed 06-Nov-2017].

[36] Tim Dwyer, Yehuda Koren, and Kim Marriott. "IPSep-CoLa: An Incremental Procedure for Separation Constraint Layout of Graphs." In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 821–828. DOI: 10/fpq3c3.

[37] Andreas Eggers. "Direct Handling of Ordinary Differential Equations in Constraint-Solving-Based Analysis of Hybrid Systems." PhD thesis. 2014. URL: https://www.uni-oldenburg.de/fileadmin/user_upload/informatik/dissertation_andreas_eggers_final.pdf.

[38] Hana El Samad, Mustafa Khammash, Linda Petzold, and Dan Gillespie. "Stochastic modelling of gene regulatory networks." In: *International Journal of Robust and Nonlinear Control* 15.15 (2005), pp. 691–711. DOI: 10/c4x27j.

[39] Michael B Elowitz and Stanislas Leibler. "A synthetic oscillatory network of transcriptional regulators." In: *Nature* 403.6767 (2000), p. 335. DOI: 10/b843mp.

[40] Georgios E. Fainekos. "Robustness of Temporal Logic Specifications." PhD thesis. University of Pennsylvania, 2008. URL: http://www.public.asu.edu/~gfaineko/pub/fainekos_thesis.pdf.

[41] Georgios E. Fainekos and George J. Pappas. "Robustness of Temporal Logic Specifications." In: *FATES/RV (Formal Approaches to Software Testing and Runtime Verification) 2006*. Vol. 4262. LNCS. Springer, 2006, pp. 178–192. DOI: 10/fhthgw.

[42] Georgios E. Fainekos and George J. Pappas. "Robustness of temporal logic specifications for continuous-time signals." In: *Theoretical Computer Science* 410.42 (2009), pp. 4262–4291. DOI: 10/b7jsgj.

[43] Chiara Fracassi, Lorena Postiglione, Gianfranco Fiore, and Diego di Bernardo. "Automatic Control of Gene Expression

in Mammalian Cells." In: *ACS Synthetic Biology* 5.4 (2016), pp. 296–302. DOI: 10/gd87qv.

[44] Martin Franzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure." In: *Journal on Satisfiability, Boolean Modeling and Computation* 1 (2007), pp. 209–236.

[45] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph drawing by force-directed placement." In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164. DOI: 10/bkwtc2.

[46] Akira Funahashi, Mineo Morohashi, Hiroaki Kitano, and Naoki Tanimura. "CellDesigner: a process diagram editor for gene-regulatory and biochemical networks." In: *BIOSILICO* 1.5 (2003), pp. 159 –162. ISSN: 1478-5382. URL: http://www.sciencedirect.com/science/article/pii/S1478538203023709.

[47] H. Gray Funkhouser. "A Note on a Tenth Century Graph." In: *Osiris* 1 (1936), pp. 260–262. DOI: 10/btp2h4.

[48] Emden R. Gansner, Yehuda Koren, and Stephen North. "Graph Drawing by Stress Majorization." In: *Graph Drawing*. Springer Berlin Heidelberg, 2005, pp. 239–250. DOI: 10/fhx96j.

[49] Emden R. Gansner and Stephen C. North. "An open graph visualization system and its applications to software engineering." In: *Software Practice and Experience* 30.11 (2000), pp. 1203–1233. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.5621&rep=rep1&type=pdf.

[50] T S Gardner, C R Cantor, and J J Collins. "Construction of a genetic toggle switch in Escherichia coli." In: 403.6767 (2000), p. 339. DOI: 10/ds7bc6.

[51] T S Gardner, C R Cantor, and J J Collins. "Construction of a genetic toggle switch in Escherichia coli." In: *Nature* 403.6767 (2000), p. 339. DOI: 10/ds7bc6. URL: http://eutils.

ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed\&id=10659857\&retmode=ref\&cmd=prlinks.

[52] Colin S. Gillespie, Darren J. Wilkinson, Carole J. Proctor, Daryl P. Shanley, Richard J. Boys, and Thomas B. L. Kirkwood. "Tools for the SBML Community." In: *Bioinformatics* 22.5 (2006), pp. 628–629. DOI: 10/drx4xq.

[53] Daniel T. Gillespie. "Exact stochastic simulation of coupled chemical reactions." In: *The Journal of Physical Chemistry* 81.25 (1977), pp. 2340–2361. DOI: 10/drnsjm.

[54] Daniel T. Gillespie. "The chemical Langevin equation." In: *The Journal of Chemical Physics* 113.1 (2000), p. 297. DOI: 10/fv7j55.

[55] T.R.G. Green. "Cognitive dimensions of notations." In: *People and Computers V: Proc. British Computer Society HCI'89 Conference*. Ed. by A. Sutcliffe and L. Macaulay. Springer, 1989.

[56] T.R.G. Green and M. Petre. "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework." In: *Journal of Visual Languages & Computing* 7.2 (1996), pp. 131–174. DOI: 10/cc49kd.

[57] Sarah Guiziou, Federico Ulliana, Violaine Moreau, Michel Leclere, and Jerome Bonnet. "An Automated Design Framework for Multicellular Recombinase Logic." In: *ACS Synthetic Biology* 7.5 (2018), pp. 1406–1412. DOI: 10/gd87qw.

[58] Chinmaya Gupta, José Manuel López, Robert Azencott, Matthew R. Bennett, Krešimir Josić, and William Ott. "Modeling delay in genetic networks: From delay birth-death processes to delay stochastic differential equations." In: *The Journal of Chemical Physics* 140.20 (2014), p. 204108. DOI: 10/f66hxq.

[59] Vishal Gupta, Jesús Irimia, Iván Pau, and Alfonso Rodríguez-Patón. "BioBlocks: Programming Protocols in Biology Made Easier." In: *ACS Synth. Biol* (2017). DOI: 10/gd87p4.

[60] E Hairer. *Solving ordinary differential equations I: Nonstiff Problems*. Berlin New York: Springer-Verlag, 1993. ISBN: *978-3-540-56670-0*.

[61] Jason M. Haugh, Najaf A. Shah, and Casim A. Sarkar. "Robust Network Topologies for Generating Switch-Like Cellular Responses." In: *PLoS Computational Biology* 7.6 (2011), e1002085. DOI: 10/bsjv7t.

[62] Desmond J. Higham. "Modeling and Simulating Chemical Reactions." In: *SIAM Review* 50.2 (2008), pp. 347–368. DOI: 10/cnt2mk.

[63] Alan C. Hindmarsh. "ODEPACK, A Systematized Collection of ODE Solvers." In: *Scientific Computation*. IMACS Transactions on Scientific Computation (1983), pp. 55–64.

[64] Harry Hochheiser. "Interactive Graphical Querying of Time Series and Linear Seqeunce Data Sets." PhD thesis. 2003. URL: http://hcil.cs.umd.edu/trs/2003-20/2003-20.pdf.

[65] Harry Hochheiser and Ben Shneiderman. "Dynamic query tools for time series data sets: Timebox widgets for interactive exploration." In: *Information Visualization* 3.1 (2004), pp. 1–18. DOI: 10/fdp885.

[66] Paulien Hogeweg. "The Roots of Bioinformatics in Theoretical Biology." In: *PLoS Computational Biology* 7.3 (2011). Ed. by David B. Searls, e1002021. DOI: 10/crqhvh.

[67] Bardh Hoxha. "Formal Requirements-Driven Analysis of Cyber Physical Systems." PhD thesis. Arizona State University, 2017. URL: http://www2.cs.siu.edu/~bhoxha/papers/PhD_Diss_Hoxha.pdf.

[68] Bardh Hoxha, Nikolaos Mavridis, and Georgios Fainekos. "VISPEC: A graphical tool for elicitation of MTL requirements." In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Institute of Electrical and Electronics Engineers (IEEE), 2015. DOI: 10/gd87pt.

[69] Bardh Hoxha, Nikolaos Mavridis, and Georgios Fainekos. "VISPEC: A graphical tool for elicitation of MTL requirements." In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Institute of Electrical and Electronics Engineers (IEEE), 2015. DOI: 10/gd87pt.

[70] Bardh Hoxha, Hoang Bach, Houssam Abbas, Adel Dokhanchi, Yoshihiro Kobayashi, Georgios Fainekos, Arizona State University, Tempe, AZ, and U.S.A. "Towards Formal Specification Visualization for Testing and Monitoring of Cyber-Physical Systems." In: *Design and Implementation of Formal Tools and Systems [DIFTS], Lausanne, Switzerland*. 2014. URL: http://www.public.asu.edu/~gfaineko/pub/difts2014.pdf.

[71] M. Hucka et al. "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models." In: *Bioinformatics* 19.4 (2003), pp. 524–531. DOI: 10/fm33b7.

[72] In Young Hwang, Mui Hua Tan, Elvin Koh, Chun Loong Ho, Chueh Loo Poh, and Matthew Wook Chang. "Reprogramming Microbes to Be Pathogen-Seeking Killers." In: *ACS Synthetic Biology* 3.4 (2014), pp. 228–237. DOI: 10/gd87p2.

[73] *Instruction Manual: Premade Mix & Go Competent E. Coli cells*. URL: https://www.zymoresearch.eu/media/amasty/amfile/attach/_T3015_T3007_T3009_T3010_T3011_T3013_T3003_T3005_T3017_T3019_T3020_Premade_Mix_Go_Competent_E._coli_Cells_Instructions_ver.1.0.9_Final_.pdf.

[74] Hidde de Jong. "Modeling and Simulation of Genetic Regulatory Systems: A Literature Review." In: *Journal of Computational Biology* 9.1 (2002), pp. 67–103. DOI: 10/b5h4ww.

[75] Tomihisa Kamada and Satoru Kawai. "An algorithm for drawing general undirected graphs." In: *Information Processing Letters* 31.1 (1989), pp. 7–15. DOI: 10/b6jdgb.

[76] Jonathan R. Karr, Jayodita C. Sanghvi, Derek N. Macklin, Miriam V. Gutschow, Jared M. Jacobs, Benjamin Bolival, Nacyra Assad-Garcia, John I. Glass, and Markus W. Covert. "A Whole-Cell Computational Model Predicts Phenotype from Genotype." In: *Cell* 150.2 (2012), pp. 389–401. DOI: 10/f34z7k.

[77] Michael Kaufmann. *Drawing graphs : methods and models*. Berlin New York: Springer, 2001. ISBN: *978-3-540-42062-0*.

[78] Jay Keasling. "The Promise of Synthetic Biology." In: *The Bridge: Linking Engineering and Society* 35.4 (2005), pp. 18–21. DOI: 10/d6cq92.

[79] Eamonn J. Keogh, Harry Hochheiser, and Ben Shneiderman. "An Augmented Visual Query Mechanism for Finding Patterns in Time Series Data." In: *Proceedings of the 5th International Conference on Flexible Query Answering Systems*. FQAS '02. 2002, pp. 240–250. ISBN: *3-540-00074-7*.

[80] JaeKyoung Kim, Krešimir Josić, and MatthewR. Bennett. "The Validity of Quasi-Steady-State Approximations in Discrete Stochastic Simulations." In: *Biophysical Journal* 107.3 (2014), pp. 783–793. DOI: 10/f6c99r.

[81] S Kirkpatrick, C D Gelatt, and M P Vecchi. "Optimization by Simulated Annealing." In: 220.4598 (1983), p. 671. DOI: 10/cn7jh2. URL: http://www.sciencemag.org/cgi/doi/10.1126/science.220.4598.671.

[82] Tasuku Kitada, Breanna DiAndreth, Brian Teague, and Ron Weiss. "Programming gene and engineered-cell therapies with synthetic biology." In: *Science* 359.6376 (2018), eaad1067. DOI: 10/gd87pz.

[83] Eric Klavins. *Aquarium*. http://klavinslab.org/aquarium.html. [Online; accessed 06-Nov-2017]. 2016.

[84] Andreas Klöckner, Nicolas Pinto, Yunsup Lee, Bryan Catanzaro, Paul Ivanov, and Ahmed Fasih. "PyCUDA and Py-

OpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation." In: (2009). DOI: 10/c8hxzm.

[85] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. "dReach: δ-Reachability Analysis for Hybrid Systems." In: *Tools and Algorithms for the Construction and Analysis of Systems*. 2015, pp. 200–205. DOI: 10/cvhq.

[86] Zhaodan Kong, Austin Jones, Ana Medina Ayala, Ebru Aydin Gol, and Calin Belta. "Temporal Logic Inference for Classification and Prediction from Data." In: *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*. HSCC '14. ACM, 2014, pp. 273–282. ISBN: 978-1-4503-2732-9. DOI: 10/gd87pv.

[87] M. Konig, A. Drager, and H.-G. Holzhutter. "CySBML: a Cytoscape plugin for SBML." In: *Bioinformatics* 28.18 (2012), pp. 2402–2403. DOI: 10/gd87p7.

[88] Jocelyn Krebs. *Lewin's genes XI*. Burlington, Mass: Jones & Bartlett Learning, 2014. ISBN: 978-1449659851.

[89] Thomas G. Kurtz. "The Relationship between Stochastic and Deterministic Models for Chemical Reactions." In: *The Journal of Chemical Physics* 57.7 (1972), p. 2976. DOI: 10/d4b5dr.

[90] Emerald Cloud Lab. *How It Works*. http://emeraldcloudlab.com/how-it-works. [Online; accessed 06-Nov-2017]. 2017.

[91] Camille Laibe and Nicolas Le Novère. "MIRIAM Resources: tools to generate and resolve robust cross-references in Systems Biology." In: *BMC Syst. Biol.* 1.1 (2007), p. 58. DOI: 10/dw7spn.

[92] Nicolas Le Novère. "Model storage, exchange and integration." In: *BMC Neurosci.* 7.Suppl 1 (2006), S11. DOI: 10/fcn6tt.

[93] Nicolas Le Novère et al. "BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems." In: *Nucleic Acids Research* 34.Database issue (2006), pp. D689–D691. DOI: 10/d9vt7v.

[94] Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, and Hanspeter Pfister. "UpSet: Visualization of Intersecting Sets." In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 1983–1992. DOI: 10/f3ssr5.

[95] J. Liepe, C. Barnes, E. Cule, K. Erguler, P. Kirk, T. Toni, and M. P. H. Stumpf. "ABC-SysBio–approximate Bayesian computation in Python with GPU support." In: *Bioinformatics* 26.14 (2010), pp. 1797–1799. DOI: 10/bg26b6.

[96] Juliane Liepe, Paul Kirk, Sarah Filippi, Tina Toni, Chris P Barnes, and Michael P H Stumpf. "A framework for parameter estimation and model selection from experimental data in systems biology using approximate Bayesian computation." In: *Nature Protocols* 9.2 (2014), pp. 439–456. DOI: 10/f5rpq2.

[97] WendellA. Lim, ConnieM. Lee, and Chao Tang. "Design Principles of Regulatory Networks: Searching for the Molecular Algorithms of the Cell." In: *Molecular Cell* 49.2 (2013), pp. 202–212. DOI: 10/f4mssw.

[98] Gregory Linshiz, Nina Stawski, Garima Goyal, Changhao Bi, Sean Poust, Monica Sharma, Vivek Mutalik, Jay D. Keasling, and Nathan J. Hillson. "PR-PR: Cross-Platform Laboratory Automation System." In: *ACS Synth. Biol* 3.8 (2014), pp. 515–524. DOI: 10/gd87p5. URL: Array.

[99] Wenzhe Ma, Ala Trusina, Hana El-Samad, Wendell A. Lim, and Chao Tang. "Defining Network Topologies that Can Achieve Biochemical Adaptation." In: *Cell* 138.4 (2009), pp. 760–773. DOI: 10/bn32f5.

[100] David MacKay. *Information theory, inference, and learning algorithms*. Cambridge, UK New York: Cambridge University Press, 2003. ISBN: *9780521642989*.

[101] Jean-Michel Marin, Pierre Pudlo, Christian P. Robert, and Robin J. Ryder. "Approximate Bayesian computational methods." In: *Statistics and Computing* 22.6 (2011), pp. 1167–1180. DOI: 10/fn98pf.

[102] Eva Charlotte Mayer. "Mutation-based Validation of Temporal Logic Specifications with Guarantees." Bachelor's Thesis. 2017. URL: https://github.com/ec-m/BachelorsThesis/blob/master/thesis.pdf.

[103] Harley H. McAdams and Adam Arkin. "Simulation Of Prokaryotic Genetic Circuits." In: *Annual Review of Biophysics and Biomolecular Structure* 27.1 (1998), pp. 199–224. DOI: 10/dpn6mh.

[104] Michael C. Minnotte and David W. Scott. "The Mode Tree: A Tool for Visualization of Nonparametric Density Features." In: *Journal of Computational and Graphical Statistics* 2.1 (1993), pp. 51–68. DOI: 10/gd87q3.

[105] Stuart Moodie, Nicolas Le Novère, Emek Demir, Huaiyu Mi, and Alice Villéger. "Systems Biology Graphical Notation: Process Description language Level 1 Version 1.3." In: *Journal of integrative bioinformatics* 12.2 (2015), p. 263. DOI: 10/gd87p8. URL: http://www.pubmed.org/26528561.

[106] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. "Sequential Monte Carlo samplers." In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.3 (2006), pp. 411–436. DOI: 10.1111/j.1467-9868.2006.00553.x.

[107] Leonardo De Moura and Nikolaj Bjørner. "Satisfiability modulo theories." In: *Communications of the ACM* 54.9 (2011), p. 69. DOI: 10/bjzhmw.

[108] Leonardo de Moura and Nikolaj Bjorner. "Z3: An Efficient SMT Solver." In: *TACAS 2008*. Ed. by C.R. Ramakrishnan and J. Rehof. Vol. 4963. LNCS. Springer, 2008, pp. 337–340. DOI: 10/fj7gzb.

[109] Leonardo de Moura and Nikolaj Bjørner. "Satisfiability Modulo Theories: An Appetizer." In: *Formal Methods: Foundations and Applications, 12th Brazilian Symposium on Formal Methods, SBMF 2009, Gramado, Brazil, August 19-21, 2009, Revised Selected Papers*. Vol. 5902. Lecture Notes in Computer Science. Springer, 2009, pp. 23–36. DOI: 10/fs6jfz.

[110] Chris J. Myers, Nathan Barker, Kevin Jones, Hiroyuki Kuwahara, Curtis Madsen, and Nam-Phuong D. Nguyen. "iBioSim: a tool for the analysis and design of genetic circuits." In: *Bioinformatics* 25.21 (2009), pp. 2848–2849. ISSN: 1367-4803. DOI: 10/b563c4.

[111] Radford M. Neal. "Probabilistic Inference Using Markov Chain Monte Carlo Methods." In: *University of Toronto Computer Science Technical Report* CRG-TR-93-1 (1993). URL: http://www.utstat.utoronto.ca/~radford/sta4503/review.pdf.

[112] Nedialko S. Nedialkov. *VNODE-LP – a validated solver for initial value problems in ordinary differential equations*. Tech. rep. Technical Report CAS-06-06-NN. Hamilton, Ontario, Canada: Department of Computing and Software, McMaster University, 2006. URL: http://www.cas.mcmaster.ca/~nedialk/vnodelp/doc/vnode.pdf.

[113] A. A. K. Nielsen, B. S. Der, J. Shin, P. Vaidyanathan, V. Paralanov, E. A. Strychalski, D. Ross, D. Densmore, and C. A. Voigt. "Genetic circuit design automation." In: *Science* 352.6281 (2016), aac7341–aac7341. DOI: 10/bdvg.

[114] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. "Abstract DPLL and Abstract DPLL Modulo Theories." In: *Proc. LPAR'04, LNCS 3452*. Springer, 2005. DOI: 10/bwwcsw.

[115] Irene Otero-Muras and Julio R Banga. "Multicriteria global optimization for biocircuit design." In: *BMC Systems Biology* 8.1 (2014). DOI: 10/gb3qxh.

[116] Irene Otero-Muras and Julio R. Banga. "Optimization Based Design of Synthetic Oscillators from Standard Biological Parts." In: *Computational Methods in Systems Biology*. Lecture Notes in Computer Science 8859 (2014), pp. 225–238. DOI: 10/gd87qq.

[117] Irene Otero-Muras, David Henriques, and Julio R. Banga. "SYNBADm: a tool for Optimization-based Automated De-

sign of Synthetic Gene Circuits." In: *Bioinformatics* (2016), btw415. DOI: 10/f9b4k5.

[118] *Over £60 million for synthetic biology (Press Release)*. Department for Business, Innovation & Skills, 2013. URL: https://www.gov.uk/government/news/over-60-million-for-synthetic-biology.

[119] Erin L. O'Brien, Elizabeth Van Itallie, and Matthew R. Bennett. "Modeling synthetic gene oscillators." In: *Mathematical Biosciences* 236.1 (2012), pp. 1–15. DOI: 10/fznnzb.

[120] Johan Paulsson. "Models of stochastic gene expression." In: *Physics of Life Reviews* 2.2 (2005), pp. 157–175. DOI: 10/bsww4t.

[121] Linda Petzold. "Automatic Selection of Methods for Solving Stiff and Nonstiff Systems of Ordinary Differential Equations." In: *SIAM Journal on Scientific and Statistical Computing* 4.1 (1983), pp. 136–148. DOI: 10/bk72v5.

[122] William Press. *Numerical recipes in C : the art of scientific computing*. Cambridge Cambridgeshire New York: Cambridge University Press, 1992. ISBN: *0521431085*.

[123] Jacqueline Y. Quinn et al. "SBOL Visual: A Graphical Language for Genetic Designs." In: *PLOS Biology* 13.12 (2015), e1002310. DOI: 10/f77vpz.

[124] K. F. Riley. *Mathematical methods for physics and engineering*. Cambridge: Cambridge University Press, 2006. ISBN: *9780521679718*.

[125] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. "Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures." In: *Theoretical Computer Science* 412.26 (2011), pp. 2827–2839. DOI: 10/dfq.

[126] Dae-Kyun Ro et al. "Production of the antimalarial drug precursor artemisinic acid in engineered yeast." In: 440.7086 (2006), p. 940. DOI: 10/dr3xnt.

[127] G. Rodrigo, J. Carrera, and A. Jaramillo. "Genetdes: automatic design of transcriptional networks." In: *Bioinformatics* 23.14 (2007), pp. 1857–1858. DOI: 10/dqdnf2.

[128] Guillermo Rodrigo and Alfonso Jaramillo. "AutoBioCAD: Full Biodesign Automation of Genetic Circuits." In: *ACS Synthetic Biology* 2.5 (2013), pp. 230–236. DOI: 10/gd87qs.

[129] Nicolas Rodriguez et al. "The systems biology format converter." In: *BMC Bioinf.* 17.1 (2016), pp. 1–7. ISSN: 1471-2105. DOI: 10/f8vnmq.

[130] Hendrik Roehm, Thomas Heinz, and Eva Charlotte Mayer. "STLInspector: STL Validation with Guarantees." In: *Computer Aided Verification*. Springer International Publishing, 2017, pp. 225–232. DOI: 10/gd87qt.

[131] Nicholas Roehner, Zhen Zhang, Tramy Nguyen, and Chris J. Myers. "Generating Systems Biology Markup Language Models from the Synthetic Biology Open Language." In: *ACS Synth. Biol.* 4.8 (2015), pp. 873–879. DOI: 10/gd87p9.

[132] Nicholas Roehner et al. "Sharing Structure and Function in Biological Design with SBOL 2.0." In: *ACS Synth. Biol.* 5.6 (2016), pp. 498–506. DOI: 10/f9tdcs.

[133] Armin Ronacher. *Flask (A Python Microframework)*. http://flask.pocoo.org/. [Online; accessed 06-Nov-2017].

[134] Yolanda Schaerli, Andreea Munteanu, Magūi Gili, James Cotterell, James Sharpe, and Mark Isalan. "A unified design space of synthetic stripe-forming networks." In: *Nature Communications* 5 (2014), p. 4905. DOI: 10/f6kpdj.

[135] Waltemath D. Scharm M. Wolkenhauer O. "An algorithm to detect and communicate the differences in computational models describing biological systems." In: *Bioinformatics* 32 (2016). DOI: 10/f8c9k4.

[136] Martin Scharm. "Improving Reproducibility and Reuse of Modelling Results in the Life Sciences." PhD thesis. University of Rostock, 2018. URL: http://rosdok.uni-rostock.

de/file/rosdok_disshab_0000001974/rosdok_derivate_0000057395/Dissertation_Scharm_2018.pdf.

[137] James Scott-Brown, Thomas Prescott, and Anronis Papachristodoulou. "TEBio - Tools for Engineering Biology." In: *IWBDA '16*. 2016. URL: http://www.iwbdaconf.org/2016/docs/IWBDA_2016_Proceedings.pdf.

[138] Ferdinand Sedlmayer, Dominique Aubel, and Martin Fussenegger. "Synthetic gene circuits for the detection, elimination and prevention of disease." In: *Nature Biomedical Engineering* 2.6 (2018), pp. 399–415. DOI: 10/gd87qg.

[139] Najaf A. Shah and Casim A. Sarkar. "Computationally Guided Design of Robust Gene Circuits." In: *Methods in Molecular Biology*. 2014, pp. 167–178. DOI: 10/gd87qn.

[140] P. Shannon. "Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks." In: *Genome Res.* 13.11 (2003), pp. 2498–2504. DOI: 10/b7kgpg.

[141] Sun-Joo Shin. *The logical status of diagrams*. Cambridge England New York: Cambridge University Press, 1994. ISBN: *978-0521461573*.

[142] Scott. A Sisson and Yanan Fan. "Likelihood-Free MCMC." In: *Handbook of Markov Chian Monte Carlo*. Ed. by Steve Brooks, Andrew Gelman, and Galin L. Jones, pp. 313–335. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.754.2427&rep=rep1&type=pdf.

[143] D. Soloveichik, G. Seelig, and E. Winfree. "DNA as a universal substrate for chemical kinetics." In: *Proceedings of the National Academy of Sciences* 107.12 (2010), pp. 5393–5398. DOI: 10/cdvspr.

[144] Orkun S. Soyer, Marcel Salathé, and Sebastian Bonhoeffer. "Signal transduction networks: Topology, response and biochemical processes." In: *Journal of Theoretical Biology* 238.2 (2006), pp. 416–425. DOI: 10/b2dv8p.

[145] *Spinout from ITU automates strenuous lab work*. IT University of Copenhagen, 2018. URL: https://en.itu.dk/about-itu/press/news-from-itu/2018/spinout-from-itu-automates-strenuous-lab-work.

[146] Jacob Stewart-Ornstein, Susan Chen, Rajat Bhatnagar, Jonathan S. Weissman, and Hana El-Samad. "Model-guided optogenetic study of PKA signaling in budding yeast." In: *Molecular Biology of the Cell* 28.1 (2016). Ed. by Doug Kellogg, pp. 221–227. DOI: 10/f9m6ks.

[147] Synthace. *Antha*. http://www.antha-lang.org. [Online; accessed 06-Nov-2017]. 2017.

[148] "Synthetic Biology Open Language Visual (SBOL Visual) Version 2.0." In: *Journal of Integrative Bioinformatics* 15.1 (2018). DOI: 10/gd87qx.

[149] Roberto Tamassia. *Handbook of graph drawing and visualization*. Boca Raton: CRC Press, Taylor & Francis Group, 2014. ISBN: *9781584884125*. URL: https://cs.brown.edu/~rt/gdhandbook/.

[150] T. Toni and M. P. H. Stumpf. "Simulation-based model selection for dynamical systems in systems and population biology." In: *Bioinformatics* 26.1 (2009), pp. 104–110. DOI: 10/fcpvgm.

[151] Tina Toni, David Welch, Natalja Strelkowa, Andreas Ipsen, and Michael P.H Stumpf. "Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems." In: *Journal of The Royal Society Interface* 6.31 (2009), pp. 187–202. DOI: 10.1098/rsif.2008.0172.

[152] Transcriptic. *Autoprotocol*. http://autoprotocol.org. [Online; accessed 06-Nov-2017]. 2016.

[153] W. T. Tutte. "How to Draw a Graph." In: *Proceedings of the London Mathematical Society* s3-13.1 (1963), pp. 743–767. DOI: 10/cs6mnx.

[154] John J Tyson, Katherine C Chen, and Bela Novak. "Sniffers, buzzers, toggles and blinkers: dynamics of regulatory and signaling pathways in the cell." In: *Current Opinion in Cell Biology* 15.2 (2003), p. 221. DOI: 10/cdxs9x.

[155] Domitilla Vecchio. *Biomolecular feedback systems*. Princeton, New Jersey Oxford, England: Princeton University Press, 2015. ISBN: 9780691161532.

[156] James Watson. *Molecular biology of the gene*. Boston: Pearson, 2014. ISBN: 978-0321762436.

[157] "What's in a name?" In: *Nature Biotechnology* 27.12 (2009), pp. 1071–1073. DOI: 10/bddjsp.

[158] Ellis Whitehead, Fabian Rudolf, Hans-Michael Kaltenbach, and Jörg Stelling. "Automated Planning Enables Complex Protocols on Liquid-Handling Robots." In: *ACS Synthetic Biology* 7.3 (2018), pp. 922–932. DOI: 10/gd87qb.

[159] Darren Wilkinson. *Stochastic modelling for systems biology*. Boca Raton: CRC Press/Taylor & Francis, 2012. ISBN: 9781439837726.

[160] Mae L. Woods, Miriam Leon, Ruben Perez-Carrasco, and Chris P. Barnes. "A Statistical Approach Reveals Designs for the Most Robust Stochastic Gene Oscillators." In: *ACS Synthetic Biology* 5.6 (2016), pp. 459–470. DOI: 10/gd87qj.

[161] Z. Xie, L. Wroblewska, L. Prochazka, R. Weiss, and Y. Benenson. "Multi-Input RNAi-Based Logic Circuit for Identification of Specific Cancer Cells." In: *Science* 333.6047 (2011), pp. 1307–1311. DOI: 10/bftjqw.

[162] Yaoyu Yang. "Design, Construction, and Validation of Dynamic Gene Circuits in Yeast." PhD thesis. University of Washington, 2016. URL: https://digital.lib.washington.edu/researchworks/handle/1773/38114.

[163] Haifeng Ye and Martin Fussenegger. "Synthetic therapeutic gene circuits in mammalian cells." In: *FEBS Letters* 588.15 (2014), pp. 2537–2544. DOI: 10/gd87qh.

[164] Y. Zhou, J. Liepe, X. Sheng, M. P. H. Stumpf, and C. Barnes. "GPU accelerated biochemical network simulation." In: *Bioinformatics* 27.6 (2011), pp. 874–876. DOI: 10/bvx363.

[165] Ali R. Zomorrodi and Costas D. Maranas. "Coarse-grained optimization-driven design and piecewise linear modeling of synthetic genetic circuits." In: *European Journal of Operational Research* 237.2 (2014), pp. 665–676. DOI: 10/gd87qr.

[166] Curtis Madsen *et al.* "Phoenix: A Systems Engineering Approach to Genetic Circuit Synthesis." In: *IWBDA '17*. 2017. URL: http://www.iwbdaconf.org/2017/docs/IWBDA_2017_Proceedings.pdf.

[167] Péter Érdi. *Stochastic chemical kinetics : theory and (mostly) systems biological applications*. New York, NY: Springer, 2014. ISBN: *978-1-4939-0387-0*.

COLOPHON

This document was typeset using the `classicthesis` LaTeX package developed by André Miede and Ivo Pletikosić.

Hermann Zapf's *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL*) are used. The "typewriter" text is typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.)

UML Diagrams were drawn using the yEd Graph Editor from yWorks. Other diagrams were created using Graphic for Mac from Picta Inc.

Final Version as of February 19, 2020 (`classicthesis` version 4.4).