The Modelling and Synthesis of Chemical Reaction Networks

Max Alexander Norman Whitby

A thesis presented for the degree of
Doctor of Philosophy in Computer Science

Department of Computer Science
Keble College
Submitted Trinity Term 2019

Abstract

Biochemical systems have been influenced, altered, and engineered to produce a myriad of complex behaviours that have far reaching consequences in the realization of computational models and in applications such as pharmaceuticals and cell networks. However, unlike producing desired behaviours in digital systems, the mechanics behind biochemical systems are complex and poorly understood. In particular, gaps in our understanding of the behaviour of cell mechanics or synthetic molecular systems have hindered the production of biochemical circuits that model or enact certain behaviours.

This thesis addresses these issues by accepting the inherent uncertainty and unknowns within biochemical systems, and exploring ways to produce desired behaviours despite these difficulties. This is done through the use of deterministic and stochastic interpretations of Chemical Reaction Networks (CRNs) to describe these biochemical systems. The first part of this thesis considers parameter synthesis of CRNs. A novel sketching language for CRNs that allows for both discrete and continuous parameter declarations is proposed. Often it is not only the correctness of synthesized programs that is important, but also their optimality with respect to a given cost function. Based on a cost function given by the structure of a CRN, there is an attempt to reduce the cost in order to produce a CRN optimal for the given cost function. Synthesis of complex CRN behaviour without structural constraints is seen as intractable, and therefore Syntax-Guided Synthesis (SyGuS) is employed to constrain the search space and allow us to synthesize behaviour of non-linear Ordinary Differential Equations (ODEs). Algorithms and case studies aimed at finding an optimal CRN are provided. Satisfiability Modulo Theory (SMT-ODE) solvers are employed to solve parametric ODEs constructed from a combination of the Linear Noise Approximation dynamics and sketching language choice variables. In our tool, named `CRNSketch`, a generalization of the CRN synthesis problem to include non mass-action kinetics as well as arbitrary continuous functions as inputs to a CRN, is successfully demonstrated. Biologically motivated case studies are provided highlighting the need for the tool, as well as our parameter and structural

synthesis methods in general. An evaluation is provided on the limitations and tractability of our parameter synthesis approach.

The second part of this thesis addresses the problem of control of CRNs, because of the inherent unknowns in biochemical systems. Using a reference CRN which exhibits a desired continuous behaviour, the principles of Proportional-Integral-Derivative (PID) control are employed, in an attempt to control the behaviour of any arbitrary CRN. That is, given an input signal, represented by the concentration level of a species, a series of CRNs are constructed that in turn attempt to control the output concentration of an arbitrary CRN, such that the plant CRN exhibits the behaviour of the reference signal. A novel CRN implementation of a derivative component is provided, and proofs and simulation of its operation are given. This novel derivative component is used as a building block for a PID controller, enabling us to compare this with an existing PI controller, and show how negative feedback with a PID controller can be implemented in CRNs. The effectiveness of this architecture on a microRNA regulated gene expression example is demonstrated, where the time evolution of a protein is controlled by acting on the expression of mRNA and microRNA using a CRN reference signal provided by our synthesis method.

Dedication

I dedicate this thesis to family. To my parents Lynne and Michael who have given love, support and guidance in ways I didn't think possible. To my brothers Brodie and Archie who have always been there for me. To my Auntie Avril who provided love and constant support from afar. To my grandparents Papa Norrie and Bamba, I wish they were alive to see the completion of this. To my family in New Zealand John, Anne, Jessica, James, Charles and Alistair. To all friends and family past and present. Thank you.

Acknowledgements

First and foremost I would like to thank Marta Kwiatkowska. When I was lost she provided wisdom and guidance, when I was sad she provided compassion and friendship, and when I was lazy she gave me many reasons to work. The production of this thesis, papers and presentations would not have been possible without her.

I would like to thank Luca Cardelli for his near limitless time and patience he has gifted to me. His engagement and excitement is clear to all who have meetings with him. Thank you for all the inspiration and ideas.

I would like to thank Luca Laurenti. He provided me with friendship, mentoring and support throughout. I think my life as a DPhil student would have been miserable without him on an academic and personal level.

I would also like to thank Nicola Paoletti and Milan Češka for the hours they invested into my understanding over and above the directions of various projects.
I would like to thank Martin Fränzle, Andrew Turberfield, Chris Myers, Alessandro Abate, Neil Dalchau, and Andreas Eggers for meaningful and fruitful discussion that lead to many of the ideas presented here.
I would like to thank many co-authors: Luca Cardelli (Oxford University), Milan Češka (Brno University of Technology), Martin Fränzle (Carl von Ossietzky University), Marta Kwiatkowska (Oxford University), Luca Laurenti (Oxford University), Nicola Paoletti (Royal Holloway University), Mirco Tribastone (IMT Luca) and Max Tschaikowski (Technical University Vienna) for all of their invaluable contributions.

Contents

CHAPTER 1

Introduction

Biochemical systems have been influenced, altered, and engineered to produce a myriad of complex behaviours that have far reaching consequences in the realization of computational models, and in applications such as pharmaceuticals [1, 2] and cell networks [3, 4, 5]. However, unlike producing desired behaviour in digital systems, the mechanics behind biochemical system processes are complex and poorly understood. Recent advances in experimentation allow for the observation and, indeed, the manipulation of biochemical systems either *in vivo* or *in vitro*, where *in vitro* [6, 7] refers to systems constructed usually by individual molecular design and *in vivo* [8, 9] refers to cell processes altered via internal or external influence.

Biochemical systems can be captured within the formalism of Chemical Reaction Networks (CRNs), a language widely used to describe molecular interactions. A CRN is given by a set of chemical reactions (e.g. $A + B \rightarrow C$), where molecules interact according to reaction laws. Reactions fire in parallel and according to a probability distribution. While several semantics are used to capture inherent noise within a CRN, the CRN formalism itself does not capture unknown or partial information within a system. Few methods address the problem of trying to parameterize unknown information within a CRN and, of the methods that exist, none are scalable [10]. Just as a computer program is written to fulfil a task or specification, a CRN can be modelled to produce a desired outcome. Unfortunately, unlike constructing computer programs for the completion of a task, it is nearly impossible to construct a CRN with a desired output in mind. One solution for addressing this problem is to automate the task of constructing a CRN to match a formal specification, referred to as *synthesis*. With *synthesis*, the CRN is constructed or adapted to produce a desired outcome.

Alternatively, another solution is to externally influence a pre-existing CRN to enact a certain behaviour, referred to as *control*. Both of these avenues are explored within this thesis.

Aims of the Thesis The aim of this thesis is to explore the modelling and synthesis of CRNs in order to implement certain continuous behaviours, whilst also taking into account the inherent uncertainties of the biochemical systems that underlie these CRNs. Previous attempts at synthesizing biochemical systems have either been intractable or lacked expressivity. Similarly, efforts aimed at the control of CRNs to track desired behaviours have lacked sufficient accuracy and stability. The aim of this thesis is to address these problems, providing tractable methods in order to explore synthesis and control of the behaviour of such systems. With these methods, there exists a desire to create small and implementable CRNs that output common desirable continuous behaviours, and to examine systems with sufficient unknown structural information in order to control these. The aim is to provide a general framework for synthesis and control of CRNs that could theoretically be implemented *in vivo* or *in vitro*. The outcomes of this research are important in both systems biology, the field of analysis and modelling of complex biological systems, and synthetic biology, the construction and redesign of biological components that do not already occur in the natural world.

Approach This thesis focuses on modelling and synthesizing behaviours of Chemical Reaction Networks. CRNs have been used to model many distributed systems, which feature in different scenarios of computer science, engineering and biology [4, 11]. Focus will be given to deterministic CRN models as Ordinary Differential Equations (ODEs), which are governed by the laws of mass action kinetics and sometimes also work with the stochastic representation as a Continuous-Time Markov Chain (CTMC), which is derived from the Chemical Master Equation (CME). CRNs have the advantage of being compact, and are suitable to be used to model biochemical systems, or as a programming language for biochemical devices [12, 7]. This thesis uses CRNs, like any other formal language, to model a range of biochemical systems that exhibit desirable [13], or otherwise commonly found behaviours within natural systems, demonstrating the diverse range and applications of CRNs.

Although in previous works [14, 15] CRN designs that satisfy discrete Petri-net specifications are provided, this thesis primarily focuses on CRNs whose outputs satisfy specifications that model continuous behaviours. The need for modelling of continuous behaviours such as these is made apparent in [16], where a continuous sigmoidal response function

is required for building a complex nucleic acid neural network. CRNs that produce Gaussian (bellshape), sigmoidal and Poisson outputs, which have applications in more complex systems such as neural networks, are examined. In addition to studying behaviours irrespective of input conditions, CRN responses to input behaviours are explored, including the synthesis of switches, and the control of gene expression networks.

Using partially defined CRN models, the problem of parameter synthesis is studied for deterministic and stochastic CRNs. Parameter synthesis has applications in synthetic and systems biology, for example, in the design automation for molecular devices [17], or the construction of predictive molecular models [18]. The field of automated synthesis is concerned with finding satisfying constraints in order for a problem formulation to satisfy a specification. In this case, the specification is a fragment of Metric Temporal Logic (MTL), which is a time constrained temporal logic including the standard temporal operators i.e. *until*. Parameter synthesis using MTL is demonstrated in several works both in a biological context [19] and in wider non-linear ODE systems (see literature review in [20]). Previously, automated synthesis of CRNs was generally limited to the estimation or parameter synthesis of rate parameters [21, 22], which neglect the network structure, and suffer from scalability issues [10]. Scalability issues are circumvented by using the Linear Noise Approximation (LNA) for approximating the stochastic interpretation of a CRN, rather than the classically used CME which involves constructing and evolving a large CTMC. The LNA, and the use of ODE-based semantics in general, allows us to encode the synthesis problem as a Satisfiability Modulo Theories (SMT)-ODE problem. SMT-ODE problems extend SMT problems, which are decision problems for first-order logic, to include reasoning over sets of parametric ODEs.

This thesis reasons over the discrete structure of models, as well as continuous parameters. Considered is a subset of parameter synthesis that also examines structural synthesis called Syntax-Guided Synthesis (SyGuS). The field of SyGuS [23] is based on the idea of providing both a specification, and a syntactic template that describes a high-level structure of the program and constrains the space of allowed programs. Applications range from bit-streaming programming [24] and concurrent data structures [25], to computational biology [26]. Synthesis of complex CRN behaviour without such structure is seen as intractable [10], and therefore SyGuS is employed to constrain the search space and allow us to synthesize behaviour of non-linear ODEs.

Often it is not only the correctness of synthesized programs that is important, but also their optimality with respect to a given cost [27]. Based on a cost function given by the structure of a CRN, a cost function is reduced in order to produce a CRN optimal for the given cost function. In contrast to previous parameter estimation work [19], a sketching

language for CRNs is proposed that allows for discrete and continuous parameter constraints. This sketching language can adequately and concisely capture the syntactic constraints, while at the same time allowing under-specification of the network.

The use of SMT-ODE solvers are employed to solve parametric ODEs constructed from a combination of the LNA dynamics and sketching language choices. In our tool `CRNSketch`, the generalization of the problem to include non mass-action kinetics as well as ODE-inputs to a CRN network is successfully demonstrated. However, with more challenging behaviours and more complex CRNs, the limit to which our method is scalable has been reached. Therefore an alternative approach is explored, where a system with unknown variables is forced to track a reference signal which can be seen as analogous to a continuous specification.

The theory of control engineering is borrowed and applied it to CRNs. Using a reference CRN which exhibits a desired behaviour, the principles of negative feedback are employed, specifically Proportional-Integral-Derivative (PID) control, to attempt to control any arbitrary CRN. That is, given an input signal (represented by the concentration level of species), a series of CRNs are constructed that in turn attempt to control the output concentration of an arbitrary CRN, such that the plant CRN exhibits the behaviour of the input. Previous attempts at formalizing negative feedback control in CRNs consist only of a chemical Proportional-Integral controller. This thesis provides a novel CRN implementation of a derivative component and provide proofs and simulation of its operation.

A derivative CRN is provided such that, given an input signal the output is the concentration of two species whose difference gives the derivative of the input signal. This novel derivative component is used as a building block for a PID controller, enabling us to compare this with an existing PI controller, and show how negative feedback with a PID controller can be implemented in CRNs. The effectiveness of this architecture is demonstrated on a microRNA regulated gene expression example [28, 29], where the time evolution of a protein is controlled by acting on the expression of mRNA and microRNA.

Contributions

In summary, the following contributions are made:

- The first sketching language for CRNs that supports partial specifications of the topology of the network and structural dependencies among species and reactions is proposed.
- Using the LNA as a continuous approximation of stochastic dynamics, a novel optimal synthesis problem is formulated that can be solved as a parametric SMT-ODE problem over the reals. Due to this approximation, the approach offers superior scalability with

respect to the number of parameters and size of the overall system. Importantly, our method supports not only the synthesis of rate parameters but also the discrete CRN structure. Several synthesis algorithms are designed and discussed, which ensures the optimality of the solution, and motivate the choices of the SMT solver.

- A tool is demonstrated that successfully captures and generalizes the above, with both a GUI and an API, for bio-chemical engineers and computer scientists. The tool extends the above framework to include customized kinetics, enabling wider bio-chemical applications. The tool also supports input ODEs, allowing for analysis of an outcome of a CRN based upon an input equation, something previously not supported even in simulation tools like `Visual-GEC`. Novel CRNs are presented based upon MTL specifications for several in-depth case study examples. These examples include non-linear functions, enzyme-reaction systems and switches that respond to input behaviour.

- Methods for controlling the behaviour of arbitrary CRNs are presented, in order to track an input behaviour. The first derivative action for a PID controller is given and comparisons with an equivalent PI controller are shown. The application of such a controller on gene expression models is demonstrated.

Structure

Chapter 2 reviews related works which discuss CRNs, their semantics and application in biochemical systems. It covers parameter synthesis, SyGuS, control theory, and application both in biochemical systems and other areas. Chapter 3 reviews the technical background, where the theory underpinning the contributions of this thesis is discussed. Chapter 4 introduces both the syntax and different semantics of CRNs. Chapter 5, 6, and 7 are the main contribution chapters of the thesis. Chapter 5 discusses optimal syntax-guided synthesis of CRNs, including CRN sketches and encoding of CRNs as a synthesis problem. Chapter 6 outlines our implementation of this theory in our tool `CRNSketch`, expanding on the work in Chapter 5 and demonstrating the working of the theory on several models. Chapter 7 discusses our work on control of CRNs, and more specifically introduces a derivative action for a PID controller. Chapter 8 concludes and reflects on contributions to the fields of Chemical Reaction Networks, SyGuS problems, and control of bio-chemical networks. Possible extensions and future works building upon our results are discussed.

Publications

The work presented in this thesis has been previously published in jointly authored papers. In particular, the thesis focuses on [13, 30, 31], in their entirety and [14, 15] partially. In [13], Optimal SyGuS for CRNs are studied. I jointly contributed the sketching language and the cost function for optimality. I was almost entirely responsible for the synthesis algorithms and contributed to the formulation as an SMT-ODE problem. I was responsible for the three case studies and the results sections. The results demonstrate the novel synthesis of certain continuous behaviours that were not possible previously. I presented the paper at Computer-Aided Verification 2017. In [31], I contributed the back end to the tool `CRNSketch`, which embodies the work of [13] and builds in new functionality to accommodate ODE input to CRNs and n-th order symbolic derivatives of CRN behaviour.

I led the project [30], contributed to the overall design of the PID controller, and contributed the results section in its entirety. Whilst only the diagrammatic language for CRNs is taken from [14, 15], these works, published during my DPhil, discuss the design and verification of CRNs which implement the behaviour of asynchronous circuits. These works expanded upon my Master's thesis, aiming to verify CRN circuits against a Petri net specification.

Tools

Parameter Synthesis of CRNs (`CRNSketch`) has on an online deployment at `https://crnsketch.com/`. The code is conveniently split into the back-end (`https://github.com/max1s/CRNSynthesis`), for which I was the sole developer, and the front-end user interface (`https://github.com/jamesscottbrown/crn-designer`), which James Scott-Brown wrote in full. These tools rely heavily on the `iSAT` [32] and `dReach` [33] tools for parameter synthesis of non-linear ODEs.

All the software developed for [13] is available, in addition to the tool, at `https://github.com/max1s/CRNSketchGenerator`. This includes examples of the Optimal Synthesis algorithm discussed extensively in Chapter 5. This code and repository was written by myself in full.

The code for the PID controller, including multiple reference signals and gene expression plants, is available at `https://github.com/max1s/CRNPIDController`, which I contributed in full.

CHAPTER 2

Literature Review

Contents

Reaction kinetics are the cornerstone of much of our understanding of both biological and chemical systems as we have understood them since the 19^{th} century. They are used to describe the time evolution of a biochemical process. The equations under which reaction kinetics operate are derived from experimental observation. Early works traditionally focused on *analysis*: the study of trying to simulate and model the behaviour of natural reactions [34, 35]. The Chemical Master Equation, a stochastic interpretation of the evolution of a chemical system, has been used to analyze many stochastic processes, from intercellular dynamics [36] to gene expression [37]. Mass action kinetics, a deterministic interpretation of the evolution of chemical systems, have been used for the analysis of biochemical systems, for example for studying oscillations [38] or multiple equilibria systems [39]. Reaction kinetics are often captured syntactically by a mathematical formalism known as Chemical Reaction Networks (CRNs). Now, computer scientists are exploring the use of CRNs as a biological programming language for the construction of biochemical systems, a goal that has far reaching implications in chemical circuit design, applications of which include gene expression and in-vitro DNA shape construction [12, 40]. A challenge of using CRNs is that there is no intuitive way of predicting the semantic behaviour of a CRN given its syntax, i.e. it is challenging to intuitively invent a CRN to model a certain

behaviour because of the inherent complexity of the reactions and dependency on rates and initial conditions. One approach to this problem is to automatically construct a CRN, given a specified behaviour. This borrows from the field of program synthesis, where a computer program is automatically constructed from a user defined specification. Another approach to this problem is to try to control a CRN externally in order to track a behaviour. This chapter addresses the literature on CRNs (Section 2.1), the semantics that they describe, and the realization of such systems in the natural world. Also summarized is the literature with regards to program synthesis, in particular Syntax-Guided Synthesis (SyGuS) and parameter synthesis techniques (Section 2.6). Lastly, works relating to control theory and its application in the control of biochemical processes are covered.

2.1 Chemical Reaction Networks

CRNs are traditionally used to capture the behaviour of inorganic and organic chemical reactions in a well-mixed solution [14, 41, 42]. Recently, a paradigm shift in the scientific community has seen the use of CRNs extend to that of a high-level programming language for molecular computing devices [43]. When used as such, the fundamental computational process differs from conventional digital electronics in that it involves transformation of input chemicals into output via reaction rules, as opposed to processing discrete signals (voltage bands) interpreted as Boolean values.

CRNs are closely related to existing computational models, and indeed several other works have shown complete reductions from one model to another. Such models include Fractran [44], Vector Addition Systems (VASs) [45], Petri nets [46], and Register Machines [47]. For many of these systems we can also consider stochastic or non-deterministic model variants. CRNs, because they can be viewed as a distributed system, have also been analyzed through a class of models called population protocols [48]. If we say that a molecular input can be decided true or false and we use a notion of a semi-linear map defined in [49], then it has been shown that CRNs are able to compute a sub class of first order predicates, known as semi-linear predicates. This has been extended further to show that CRNs with a stochastic representation, because they are equivalent to population protocols, can compute a function, if and only if, the graph of the function is a semi-linear set. The relationship between computational models and CRNs is shown formally in [4] by means of a process algebra, where a translation between process algebra and CRNs is given considering both deterministic and stochastic semantics. [4] presents a precise connection between process algebra models of biochemical systems and more traditional models based on chemistry.

Looking at it from a design perspective there are two methods for computing with CRNs [50]. First, the reaction stoichiometry transforms discrete quantities of reactants to products. As an example $A \rightarrow 2B$ makes two units of B for every unit of A. Second, the reaction rate laws govern a continuous transformation of reactants to products. In [50], they focus on stoichiometry alone, effectively ignoring the continuous dynamics of a CRN. This means that, as a design target, these CRNs are more widely applicable as they ignore domain specific rate constants.

In non-uniform general CRN computation, a CRN computes a function over a finite domain [51]. This is comparable to computation in Boolean circuits, since only finite binary inputs of a certain length are computed by a given circuit and making the input more verbose or complex requires greater numbers of reactions [52]. By contrast, if the computation is uniform, a single CRN computes all possible permutations of inputs, equivalent to Turing Machines which deal with arbitrary sized inputs placed on their (unbounded) input tape [51].

The time and space requirements for CRNs undertaking computation, compared to a Turing Machine, are a simple polynomial slowdown in time complexity, but an exponential increase in space, where space is broadly defined as the increase in number of species [53, 52]. Questions of reachability of the behaviour of CRN models, are decidable, whereas the corresponding questions of probabilistic reachability are undecidable [54, 52]. This results in a conclusion that; when answers must be guaranteed to be correct, the computational power is limited, but when an arbitrarily small error probability can be tolerated, the number of functions that can be computed increases.

CRN Semantics

It is known that the computational power of CRNs is affected by the choice of the semantics, whether deterministic or stochastic. In particular, assuming a small probability of error, (finite) stochastic CRNs have been shown to be Turing universal [52]. The deterministic semantics interprets the reactions as a system of differential equations, which describe the evolution of the system as a vector of real-valued species concentrations over time [55]. The stochastic semantics, on the other hand, views the state of the system as a vector of (non-negative) integer molecular counts and state transitions as reactions which have a non-zero probability of occurring [43]. The stochastic evolution of the system over time is obtained as a solution of the Chemical Master Equation (CME) [56]. It is well known that the deterministic semantics is not accurate for small populations. While the stochastic semantics is exact, it is intractable for large molecular counts and Gillespie simulation is employed. One scalable alternative is the Linear Noise Approximation, which is a real-valued

approximation of the CME [57]. The correctness of the behaviour of a circuit described by a finite CRN can be analyzed by inspecting its stochastic and deterministic evolution over time. All of the semantics discussed can be viewed as a hierarchy in which deterministic semantics can be viewed as the simplest continuous abstraction, farthest from the physical system and stochastic semantics can be seen as the closest representation of the physical system. The LNA, like the Langevin semantics, can be seen as intermediates between the two.

Deterministic Semantics Deterministic semantics, in general, approximates the species concentrations over time as a solution of rate equations [56] assuming a continuous state space. The most common deterministic scheme for CRNs is mass action kinetics, which assumes that the rate of a chemical reaction is proportional to the product of the concentrations of reacting chemical species. In [58, 59] the relationship between the structure of a CRN and its deterministic semantics is defined as a set of autonomous ODEs. Mass action kinetics have been used to study a wide range of biochemical systems from logical circuits to oscillatory systems [59, 38, 39]. Whilst solving the system of equations that underpins mass action kinetics is fast, this semantic interpretation is commonly used for high molecular counts. It is widely known that for low molecular counts the deterministic semantics fails to capture inherent stochastic behaviour. In deterministic semantics, the steady-state of a system is independent of initial concentration of molecules for each species [59]. In [60, 61] comparisons are made between the divergent behaviour of the stochastic semantics, where the deterministic semantics fails to observe a behavioural change under low molecular count.

Stochastic Semantics Under low molecular counts, the importance of stochasticity in molecular interaction becomes increasingly important, hence the need to consider stochastic semantics of CRNs. CRNs under stochastic semantics, a fixed number of species, with low error probability, can perform Turing-universal computation [52]. That is, CRNs, under stochastic semantics, can compute any computable function with probability of error less than e for any arbitrary $e > 0$, but for $e = 0$ universal computation is impossible [53, 52]. The stochastic semantics of a CRN traditionally are given in terms of a CTMC, whose transient evolution is described by the CME [56]. A derivation of the CME is also given in [62], where it is shown that the CME is exact for any well-mixed, thermally equilibrated system. Relationships between deterministic and stochastic semantics of a CRN are discussed in [63], where they derive the deterministic semantics of a CRN as an approximation of the

stochastic semantics, where the approximation error approaches zero with infinitely large molecular count.

Although the stochastic semantics accurately encapsulates a multitude of molecular behaviours in solution, the reason it is sometimes disfavoured is due to computational complexity. Solving the CME can require the solution of a large or near infinite number of differential equations, so that solving the CME is generally seen as intractable [64]. As for the CTMC representation of the CME, this is also limited as the size of the matrix is proportional to the initial molecular count. Methods, including the Chemical Langevin Equation (CLE), have been derived to address this. The CLE is an approximation technique that allows for fast checking of the accuracy of the evolution of these equations. As well as the CLE, other moment closure techniques have been derived to give continuous approximations of the CME up to a certain moment order [65, 66].

LNA Semantics A stochastic approximation of the CME is possible using the LNA [56], which assumes Gaussian distributions for variance. The LNA is derived by applying the Taylor expansion to the CME to the second order, the first order being mass action kinetics and the second order being the LNA. This is a continuous approximation, the complexity of which is independent of initial concentration of molecules for each species and hence scalable and is accurate in sufficiently high concentrations of molecules. The LNA is represented, like the deterministic semantics, by a system of ODEs which also include time evolution of variance. LNA was recently adapted to provide stochastic analysis of the evolution of populations of molecular species of CRNs [57].

The LNA has advantages and disadvantages due to the fact that it is characterized by a Gaussian process [67]. It is fully characterized by the first two moments (mean and covariance), and the resulting distribution is symmetric and uni-dimensional, hence it is sufficiently compact. The LNA is not accurate for multi-modal dynamics or very small molecular populations, which unfortunately are both common features of biochemical systems [68]. Groups have tried to address this issue by modelling a CRN as a hybrid stochastic model [69], where for low molecular accounts the CME is used and for high molecular counts the LNA is used. However, this is still intractable for large systems [70]. The LNA has been used to accurately model cell signalling [57], strand displacement systems [57], genetic low-pass filters [29] and gene expression networks [71].

Michaelis Menten and Hill Kinetics CRNs have also been given semantics in many other settings. For example, the study of the steady-state kinetic behaviour of enzymatic reactions has been codified by the Michaelis-Menten (MM) kinetics for over a century. The

MM kinetics outlines enzymatic reactions as follows: a substrate binds reversibly with a free enzyme to form a substrate-enzyme complex which in turn reacts irreversibly to yield product [72]. It is an accurate deterministic approximation of a complex stochastic process. Like MM, Hill kinetics also relate to binding, but in this case between a ligand and a macromolecule. Hill kinetics have found wide modelling applications and are commonly used to approximate interaction between proteins in cell pathways [73]. In [18] parameter estimation techniques are used to model unknown protein interaction in the MAPK/ERK signalling pathway, which they abstract with positive and negative Hill equations. Both can be used as the kinetic rate parameters in a CRN, and as such CRNs have modelled both MM and Hill kinetic systems. The MM and Hill schemes can be represented by 4 mass action kinetic bi-molecular reactions and indeed can be seen as an abstraction of mass action kinetics [74].

2.2 CRN Applications and Realizations

Principles of engineering, applied to biomolecules, have been used to convert light to fuels [75], boost microbial hosts for immune systems [76] and help sustain life support systems within spacecraft [77]. Synthetic systems have been designed to break down toxins in waste [78] and even to find useful recyclable heavy metals also in refuse [79]. Lastly, in the field of medicine, bacteria has been engineered to colonize and detect tumours [80]. Killing agents for cells have also been built into genetic circuits with the aim of eradicating cancer [81].

The computational power of CRNs, for example their use as a programming language for engineering biochemical systems, has been the subject of much research, notably [43, 55]. In particular, CRNs have the power to simulate Boolean circuits, molecular machines, or distributed algorithms [82, 52, 83]. Because CRNs have been proven to be very expressive, it is a natural progression to apply this to the construction of chemical computational devices. Just as electronic devices are composed of logic gates and circuits, similar ideas have been theorized using CRNs.

CRN Circuit Design CRNs have been used to implement their digital analogue from logic circuits [15], to neural networks [16] and even security protocols [84]. Since the behaviour of CRNs are asynchronous, a fact evident through their equivalence with Petri net models [85], the main difficulty with programming them is the need to control the order of reactions. In [43] it is suggested that this "uncontrollability" can be handled by changing rate constants, an idea followed up in [86], where CRN designs for basic arithmetic are given based on two rate constants, "fast" and "slow". We see the construction and composition

of simple logic gates based upon catalytic reactions in the context of a single rail setting. In [87], the authors propose CRNs for an inverter, an incrementer, a decrementer, and a copier; their designs are based on two rate constants, "fast" and "slow", and are therefore not rate-independent. A system of actual chemical reactions is found in [88], where a precise molecular implementation is given for gates complete with a thermodynamic analysis of how the system would evolve. An implementation of individual dual-rail logic gates that are rate-independent is given in [50]. Other work shows the practical construction of *'for'* and *'while'* loops [89]. Some works even go as far as to show signal processing operations such as filtering [90]. The problem with these constructions is that they depend on specific rate categories for reactions. They are also, for the most part, without rewritable memory which could simplify a lot of their designs and constructions. Our own work [15] presents the first designs for an asynchronous computational device constructed from CRNs whilst also providing rate independent logic circuits.

Strand Displacement Systems Realizations of CRNs have been the target behaviour of DNA Strand Displacement (DSD) systems, resulting in a novel theory of computation for such devices. [12] shows that any programmable CRN can theoretically be implemented as a strand displacement device. In [40], Cardelli shows that, by a translation to an intermediary labelled transition graph, the same CTMC can be described with a stochastic CRN or Strand Algebra, hence proving that any CRN has a corresponding Strand Algebra. [91] presents a DSD implementation of a stack machine capable of energy efficient Turing universal computation. In their paper, they describe a reversible DSD mechanism capable of implementing the CRN $A+B \leftrightharpoons C+D$, which can be biased dependent on the concentration of reactants involved. There have been many DSD schemas which perform some elementary operation such as transduction, forking or catalysis, each of which has a corresponding CRN [91, 40, 43, 92].

Signalling Pathways Signalling pathways transfer input chemicals or proteins by specific gene expression in order to edit the genome, via complex and often misunderstood mechanisms. Due to varying time scales for in vitro or in vivo observation it is hard to observe exact dynamics of signalling proteins [93]. Modelling of pathways ranges from Boolean Networks [94] to Process Algebras [95], however the classical models are usually given as an ODE-based system [96]. The MAPK/ERK pathway, which signals to the nucleus of a cell from a cell surface, is a pathway which models a chain of proteins. It is modelled and experimentally analyzed in [74, 96], where the authors detail receptor activation mechanisms.

By influencing particular parts of the receptor/adaptor proteins, they implement a chemical negative feedback.

Biochemical Circuit Design Realizations of CRNs have been simulated by partitioning of molecules into membrane compartments [97, 43]. This allows unbounded computation to be performed by molecular systems containing only limited types of enzyme and basic signal-carrying molecular components. Realizations have also been simulated via polymer [98, 99] and via cellular automata simulation in self-assembly [100]. Designs for a Muller C-element, a key component for asynchronous computation though, have been constructed from genetic logic gates [101] and a genetic toggle switch [102].

A series of papers implemented a variety of gene regulatory networks *in vivo* [68, 103, 104]. In [105] they seek to design and build biochemical reaction networks that implement digital logic, and are thus capable of carrying out arbitrary computational functions. They claim that this allows the fitting of biological cells with digital prosthesis that enable the cells to perform user-specified computational processes. This is achieved by synthesising rates of DNA binding proteins as logic signals. Since DNA binding proteins can function as transcriptional repressors, the effect of one protein on the transcription rate of another can represent the flow of logical information.

2.3 Model Checking and Synthesis

Model checking is a set of techniques to automatically verify properties of systems specified usually in temporal logic. These verification methods can be categorized by the kind of system they consider, i.e. discrete vs continuous time, finite vs infinite state, or linear vs non-linear dynamics. They can also be categorized by the type of properties they can verify. The most common analysis performed upon models in order to verify a given property is *reachability* - the property that a certain set of states can be reached. Formal verification methods are commonly embodied in the design process of biochemical models and molecular devices [106, 73, 107], since they help designers to rigorously validate the models and reason about system correctness and performance.

Temporal logics are generalised powerful specification languages compared to domain specific languages such as optimization theory and dynamical systems theory [108]. This is because temporal logics can express quantitative properties of time and system variables (e.g. a molecular count is less than 35) as well as qualitative properties (e.g. some macromolecule eventually degrades). Compare this with optimization theory which deals only with quantitative properties such as curve fitting, and dynamic systems theory which deals

only with qualitative properties such as multistability, existence of oscillations, and naturally temporal logics have wider expressive power. Temporal logics then, are effective for use in systems biology which has amassed a great deal of quantitative information that is incomplete, uncertain or imprecise. Using temporal logics in systems biology requires a logical framework, which consists of making the following identifications set out in [109]:

$$\text{biological model} = \text{transition system}$$
$$\text{biological properties} = \text{temporal logic formulae}$$
$$\text{biological validation} = \text{model checking}$$

Many applications of temporal logics are possible in this paradigm: for example, they can be used for the validation of biological models, as a specification languages of biological properties identified by experimentation [109], or as a query language of large biological interaction maps or databases (for example Kohns map of the cell cycle [110]).

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 categorized 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. A specification over one such logic, Linear Temporal Logic (LTL), in contrast to a branching temporal logic reasons over a single future or successor state for each current state. In LTL 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. Metric Temporal Logic, extends LTL to add timing constraints. MTL and Metric Interval Temporal Logic (MITL) share a syntax, however the first is for continuous-time systems such as timed automata whereas MITL is used for discrete-time models Signal Temporal Logic (STL), like MTL, is a time bounded logic. STL extends Metric Temporal Logic (MTL) [111] for the verification of real-valued signals. Often the syntax of MTL and STL is used interchangeably [112]. STL is a temporal logic for specifying desired properties of real-valued signals [113] in order to analyse real-time systems, however the computability of real-valued signals is seen as intractable. STL has been recently extended to STL* [114], where the authors introduce the freezing operator, which enables freezing the value of a signal and then reusing it for the verification of a formula. The Boolean satisfaction value of an STL formula can be enriched by a quantitative measure of the robustness of the satisfaction of the formula. In [112], the authors present several variants of robustness measures that indicate how far a given trajectory lies, in space and time, from satisfying or violating a property. This has been

used for local search methods for optimising parameters within models based upon a 'distance' metric from the satisfaction of a property [109]. STL with its quantitative measure, can be used in order to analyze deterministic or hybrid systems.

The verification of logics under parameteter uncertainty in non-linear hybrid systems has largely included symbolic methods [115] or focused on restricted classes of models such as linear hybrid systems [116, 117, 118]. The advantage of symbolic methods for hybrid systems is that they are exhaustive, that is they give provably correct results. However, generally, these methods are only scalable to systems of a small size. Works on gene regulatory networks have included verifying reachability and liveness of LTL properties [119]. They apply model checking techniques on a discrete abstraction of the continuous dynamics. In [120] this work is extended to include arbitrary non-linear systems but excludes liveness properties. However, in [121] methods are proposed for handling liveness in non-linear ODE systems, although the work is theoretical and does not consider practical implementation.

2.3.1 Syntax-Guided Synthesis

Seen as the dual of program verification, program synthesis aims to synthesize a program implementation that satisfies a given correctness specification. Program synthesis was viewed originally in the domain of deductive theorem proving: a program is derived from the constructive proof of a theorem where, given all inputs, there exists one or many outputs where a desired correctness specification holds [27]. For instance, a deductive approach to synthesising recursive programs is based on a first-order logic specification [122]. However, after soon realising the intractability and limited application of this formalization, due to near infinite decisions within a search space, attention was turned to constraining the search space via constraining the list of all possible programs.

The SKETCH system, a method for introducing partial specifications of an implementation, was introduced in [123], where there are reportedly three major implementation advantages. Firstly, the implementation of SKETCH is such that the programmer has greater descriptive flexibility. That is, a desired solution can be described using syntactic and semantic constraints combined. In the design of specifications, this greater descriptive flexibility can allow for more intuitive programming. Secondly, due to the syntactic template limiting the search space of potential implementations, the problem of synthesis becomes more tractable. Thirdly, the syntactic template can be used to constrain the space of implementations for the purpose of optimization on performance. Rather than worrying about low-level specifics, this allows the programmer to focus on exploiting broader algorithmic concepts.

A sketch compiler should be syntactically aware, i.e. it should reject syntactically incorrect sketches, thus improving reliability and allowing the implementer to quickly evaluate implementation ideas [124]. In [125], they direct the focus of this formalism at defining sketches over the C programming language. Here the program selects options on types, i.e. integers/floats and statements, i.e. loops/branching, in order to construct a program that satisfies a specification.

Many further program implementations combine the idea of specification with a program template [123, 23, 126]. In [127] we see the synthesis of Excel macros based upon a selection of around 50 templates. A major offshoot has been the synthesis of loop bodies and invariants including pre and post conditions. More recently [126], the SKETCH system has been used to to create 'programming by example' methods for DATALOG, in which programs are synthesized via a series of input-output examples.

These SKETCH synthesis problems have been categorised and unified into a framework called Syntax-Guided Synthesis (SyGuS). A SyGuS problem is specified by a context-free grammar describing the search space of programs, and a logical formula describing the specification. There is now wide syntactic implementation for the SKETCH format within solvers [123] and programmers compete annually [128] which has brought to fruition different solving methods for SKETCH-ing [129]. In [130], they describe an approach to reactive synthesis based upon an STL specification. They do this by encoding STL specifications, regarding road safety in autonomous driving, as mixed integer-linear constraints on the variables of a discrete-time model of the system and environment dynamics, and solve a series of optimization problems to yield a satisfying vehicle control sequence.

The optimal synthesis problem, involves creating a function or program that respects two constraints. Firstly, that it is correct with respect to a (logical) specification and secondly that it is optimal with respect to a given cost function. Current tools that exist for optimal synthesis are highly specialized, that is, given a large space of candidate programs, they employ custom search strategies to quickly identify the optimal solution. Perhaps the most important framework for optimal synthesis is that of metaSketches [27], a general framework for specifying and solving optimal synthesis problems. metaSketches present the search strategy as part of the problem definition by specifying a fragmentation of the search space into an ordered set of classic sketches. The improvements in expressive power of (optimal) SyGuS when employing metaSketches, is twofold. Firstly, metaSketches are able to express richer candidate spaces than is possible with either sketches or context-free grammars alone. In comparison to a classic sketch, a metaSketch is able to capture an infinite space of candidate programs, where a classic sketch is not. Moreover, a metaSketch is

able to express syntactic constraints on the search space that are context sensitive, where a context-free grammar cannot.

Secondly, `metaSketches` are unlike other forms of syntactic templates, in that they describe both a search space and a search strategy. A programmer specifies the candidate space as an ordered set of sketches, provides a decomposition of the problem into independent parts, and gives an order in which those parts are to be explored. We can think of metasketches at two polar extremes for usage [27] : at one extreme, if the search space is all possible programs then the `metaSketch` is an implementation of a brute force search strategy. At the other extreme, if the possible solutions contain only one sketch implementation, the programmer is choosing to address the program with monolithic solver calls. `metaSketch` can easily capture search strategies that exist between the two extremes outlined. Take adaptive concretization [131] for example, which, given holes in a sketch, randomly replaces these holes with concrete values. The result is a family of sketches of roughly equal complexity - where complexity is measured by the number of holes - that are solved independently. Other algorithms have been provided for solving domain-specific problems attributable to `metaSketch`-ing [27, 132].

2.3.2 SMT-based Solving

The solution of a `metaSketch` or general SyGuS formulation usually requires the use of an Satisfiability Modulo Theory (SMT) solver. An SMT solver determines the correctness of a given logical formula built from logical connectives of typed variables and typical operations such as arithmetic and pointers [23]. SMT solvers, despite solving an intractable problem, can solve SMT problems with thousands of variables due to community sustained improvements in the fundamentals such as algorithms and improving data structures. One such progression of the community is the SMT-LIB (`www.smt-lib.org`) which sets out a list of standards and benchmarks for SMT solvers to adhere to. Becuase of these benchmarks there is also an annual competition (`www.smtcomp.org`). The majority of SMT solvers employ the famous DPLL algorithm used for satisfiability (SAT) solving. SMT-ODE decision procedures, those used to reason about the satisfiability of ODE systems, still follow standard DPLL Interval Constraint Problem (DPLL(ICP)) framework, a framework that takes an ICP as input to the DPLL algorithm.

Following standard convention, SMT-ODE problems are also existentially quantified, which we discuss in greater detail in a later chapter as a limitation for its application in biochemical systems. The need for SMT-ODE based solving is driven by the need for reasoning over the combined continuous dynamics and finite automata of hybrid systems. Formal verification of general hybrid systems requires reasoning about logic formulas over the reals

that contain ODE constraints. `iSAT` [32] and `dReach` [33] are examples of SMT solvers for non-linear formulas over the reals. The tools can handle various non-linear real functions such as polynomials, trigonometric functions and exponential functions. Given the undecidability of trigonometric functions in theories over the reals it would seem near impossible to solve formulas containing ODEs [133]. Much of this difficulty is addressed through the formalism of δ-satisfiability [133], which is employed directly within the `dReach` tool, and although not formalized completely, is discussed in detail in the thesis of Eggers [134], creator of the `iSAT` tool. An algorithm is δ-satisfiable for a set of SMT formulas, if it correctly decides whether a formula is unsatisfiable or $\delta-$satisfiable (where δ is positive). This allows for precision up to an arbitrary δ for determining whether a formula is satisfiable.

Nonlinear ODE and hybrid models are common in systems and synthetic biology, although very few methods exist to address the problem of synthesis, partly due to the complexity of such systems. In [135] the authors contribute practical algorithms that handle these complexities. However, they provide case studies that have limited transferability to general biological domains. They combine sensitivity analysis with an efficient search over parameters, assuming the model has affine dynamics. A series of papers attempt to address the problem with local search over parameters by defining a distance metric for satisfiability [136]. One difficulty with local search methods is that these distance metrics are arbitrarily defined from problem to problem, and therefore a 'nearly satisfied' solution may be imprecise. Another biologically motivated example is [137] where a general workflow for parameter synthesis based on model checking is given. They provide an extended analysis of a genetic switch controlling the regulation in mammalian cell cycle phase transition and a synthetic pathway for the biodegradation of a toxic pollutant in E. coli. However, they admit that the method is not tractable.

Related to parameter synthesis are the concepts of parameter identification and parameter estimation. Parameter identification is the problem of finding a parameter set of a system which satisfies a given property. In [138], the authors circumvent the complexity of non-linear dynamics in parameter identification by considering a restricted class of hybrid automata (multi-affine hybrid automata). They consider a gene expression model where the dynamics are first abstracted and then approximated with various linear hybrid automata. Parameter estimation is the problem of approximating or fitting parameters based upon pre-existing experimental data. In [139] they use a variation of the Kalman filter that is particularly well suited to biological applications to obtain a first guess for unknown parameters in an Ėcoli system. Secondly, they employ an a posteriori identifiability test to check the reliability of the estimates.

The tool STORM acts as a catchall, providing an extensive toolbox for probabilistic modelling checking [140]. STORM's infrastructure contains many known solvers that have been used for synthetic biology problems (. For instance, solvers are available for sets of linear or Bellman equations (both using sparse matrices as well as MTBDDs), (mixed-integer) linear programming (MILP) and satisfiability modulo theories (SMT) solving.

2.4 Control Theory and Feedback in Biological Systems

Control theory is a corpus of work that embodies conceptual and general design strategies with the aim of optimizing the robustness, stability and performance in continuous systems. Applications range from power networks, space systems and chemical processes [141]. Negative feedback is when a proportion or function of the output of a system is fed back into the system order to reduce error or stabilise the output. Instability of the system can be caused by the input signal (i.e. an external influence) or by other internal disturbances. The most common example of negative feedback in biological systems is that of homeostasis. Homeostasis is the process of regulation that occurs within most living systems, for example temperature or fluid regulation. At a cellular level examples of negative feedback or homeostasis include the bararoflex [142] in blood pressure regulation or Erythropoiesis [143] - the production of red blood cells. Implementation of negative feedback systems in synthetic biology fall into two different categories: *in-cell* feedback control and *In silico* [141]. *In-cell* feedback control is where both the controller and the process (or plant) are housed in the same compartment, i.e. a cell. This implementation is suitable for *in-cell* programming applications such as in medical applications where modified cells are injected into patients or bioremediation [144], where toxic components within a cell can be marked for degradation. *In silico* feedback control [145] separates this notion of compartments where the cell is the process and is influenced by external chemical or electronic input. This is most suitable in only slight genetic modifications of a cell, such as cell reprogramming i.e. changing whether a cell is pluripotent or specialized [146]. A larger body of work to date exists for in cell feedback.

Synthetic cell feedback control is also present as a strategy in engineering metabolic pathways. As an example a heterologous pathway [147] can produce harmful intermediaries which can be eradicated via degradation control. As another example, feedback control can balance production and degradation of metabolities [148]. However, more abstracted control systems of quantifiable properties are rare. It is crucial and indeed an active area of biological research to find existing or synthesized feedback control designs that are abstractable, optimisable, tunable and for general use in metabolic engineering.

A type of negative feedback, Proportional-Integral-Derivative (PID) control systems, are widely used in engineering to control the dynamics of a system due to their ability to achieve accurate set-point tracking and robustness to disturbances, even with only partial knowledge of the system. PID follows a standard form, where an output of a process is compared against a desired point, generating an error signal. This error signal is then processed by a control system, which produces a corrective output to the original process. The theory behind it is standardized and provides general control strategies for many technological applications, from aircraft altitude to water tank pressure. Due to PID controller robustness, such mechanisms have also been applied with success in the construction of synthetic biomolecular systems [141, 149]. Moreover, molecular implementation of control systems has been shown to naturally occur in living organisms [150, 151, 152].

Integral control is perhaps the most commonly implemented control in biochemical systems due to the fact it can drive the error function to zero [153, 154]. However, integral control in such systems requires zeroth order kinetics in the controller which is violated due to dilution of solution caused by cell growth [155]. Because of this, a robust widespread implementation of integral control is highly desirable. An example of stable naturally occuring integral control occurs in *E.coli* chemotaxis [8, 9] - the movement of E.coli in response to a chemical stimulus.

As a consequence of the potential applications, CRN designs that implement control mechanisms are sought for [9, 141]. CRNs implementing proportional and integral control have been proposed [154, 9]. However, a CRN implementation of a full PID control is still missing due to the lack of a CRN implementing the derivative component. In [156] a biologically inspired implementation of a system that computes the derivative of an input is provided, however it is presented as non-mass action Hill functions.

2.5 Summary

In this section we summarize the state of the art which influenced the work we present in Chapters 5,6 and 7 and which help to build the background theory in Chapters 3 and 4.

With applications in everything from design of macro-molecules for waste disposal to neural networks, CRNs as a formalism are used to describe a wide range of bio-chemical processes. The need for CRN design has been made clear through the works of [91, 40] where CRNs have been used as a high-level programming language in order to engineer biochemical systems.

Program synthesis, the automatic construction of programs from high-level specifications, has made recent progress through the field of syntax-guided program synthesis [23], based on the idea of supplementing the specification with a syntactic template that describes

a high-level structure of the program and constrains the space of allowed programs. Applications range from bit-streaming programming [24] and concurrent data structures [25], to computational biology [26] where the authors look at the synthesis of biological models via mutation experiments. Often not only the correctness of synthesized programs is important, but also their optimality with respect to a given cost [27] which in turn can speed up the synthesis process. The solution of a `metaSketch` or general SyGuS formulation usually requires the use of an Satisfiability Modulo Theory (SMT) solver. An SMT solver determines the correctness of a logical formula over a program built from logical connectives of typed variables and typical operations such as arithmetic and pointers [23]. `iSAT` [32] and `dReach` [33] are examples of SMT solvers for non-linear formulas over the reals. The tools take as input various non-linear real functions such as polynomials, trigonometric functions and exponential functions.

With regards to specific CRN synthesis, the necessity exists because there is still a costly gap between the design and verification process for CRNs. This is particularly the case when stochasticity is considered, which is typically the case for molecular computation. Automated synthesis of CRNs is limited to a few works, none of which reason over the network structure of a CRN. In [21, 22] the authors consider synthesis of rate parameters within stochastic encodings of a CRN using CTMCs. Recently, this has been shown to suffer from scalability issues [10] where the authors attempt to synthesize two rate parameters. The algorithm terminates after 3 hours.

In control theory, the principles of PI control have been applied with success in the construction of synthetic bio-molecular systems [141, 149]. Further to this, molecular implementation of control systems has been shown to naturally occur in living organisms [150, 151, 152]. CRNs implementing proportional and integral control have been proposed [154, 9]. However, a biochemical implementation of a full PID control is still missing due to the lack of a CRN implementing the derivative component.

CHAPTER 3

Logic, Synthesis and Control

Contents

The ability to synthesize and control behaviours for Chemical Reaction Networks requires a narrowing of focus in theories. First Metric Temporal Logic (MTL) is introduced in Section 3.1. Fragments of this logic are later used to describe specifications for the synthesis of CRNs. Section 3.2 outlines the theory behind Syntax-Guided Synthesis (SyGuS) - a framework that places syntactic constraints upon a synthesis problem in order to reduce possible search space. This is expanded to look at a notion of optimality over a set of SyGuS problems with respect to a cost function. This provides background for Chapters 5 and 6 which develop a notion of sketches for CRNs which makes the problem of CRN synthesis tractable. Using SyGuS we look at methods for solving constraints upon ordinary differential equation (ODE) based systems in Section 3.3. This provides background for our tool `CRNSketch`, which is discussed in detail in Chapter 6. Finally, Section 3.4 outlines the basics of control theory and the theory behind Proportional-Integral-Derivative (PID) control which is used extensively in Chapter 7.

3.1 Temporal Logics

Definition 1. Metric Temporal Logic [157] - Syntax 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 categorized 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. A specification written in Linear Temporal Logic (LTL) considers only a single possible successor state for each current state, time is ordinal and signals are Boolean. Metric Temporal Logic (MTL) is an extension of LTL where time can be included within logical specifications. Note that MTL is considered over Signal Temporal Logic (STL) due to the analysis of real-valued signals is seen as intractable. Given a set P of atomic propositions, the formulas ϕ of MTL are built from P using Boolean connectives and time-constrained versions of the until operator U as follows:

$$\phi ::= \top \mid \mu \mid \neg\phi \mid \phi \wedge \phi \mid \phi \, U_I \, \phi,$$

where $\mu \in P$ and $I \subseteq (0, \infty)$ is an interval of reals with endpoints in $\mathbb{N} \cup \{\infty\}$. Intuitively, the meaning of $\phi_1 \, U_I \, \phi_2$ is that ϕ_2 will hold at some time in the interval I, and until then ϕ_1 holds.

Further connectives can be derived, namely $\bot$ (false) from $\neg\top$ propositions, and the disjunction $\vee$ from De Morgan's law. We can also express the *constrained eventually operator* $\Diamond I\phi \equiv \top \, U_I \, \phi$, the *constrained always* operator $\Box_I\phi \equiv \neg\Diamond_I\neg\phi$, and the *constrained dual until* operator $\phi_1 \dot{U}_I \phi_2 \equiv \neg((\neg\phi_1)U_I(\phi_2))$. By admitting only $\dot{U}_I$ as an extra connective one can transform any MTL formula into an equivalent negative normal form in which negation is applied to propositional variables. Note that in some literature a 'since' connective is sometimes used. Because connectives referring to events that happen prior to a time point in question, known as 'past' connectives, are not universally accepted within MTL we exclude this from the definition.

Definition 2. Metric Temporal Logic [158] - Point-wise Semantics The point-wise semantics of MTL has an inductive definition similar to that of LTL and is used for timed automata with labels on transitions. In point-wise semantics we reason over *timed words* which are pairs of events and times associated with those events. Note, for the continuous semantics of MTL [157] they reason over the continuous signals directly. A $signal$ is a function $f : \mathbb{R}^+ \to 2^P$ that maps a time $t \in \mathbb{R}^+$ to the set $f(t)$ of propositions holding at time t. Given a signal f and $r \in \mathbb{R}$, we define the satisfaction relation $f, r \vDash \phi$ (to be read f satisfies ϕ at position r) by induction over ϕ as follows:

- $f, r \vDash p$ iff $p \in f(r)$,
- $f, r \vDash \neg\phi$ iff $f, r \nvDash \phi$,
- $f, r \vDash \phi_1 \wedge \phi_2$ iff $f, r \vDash \phi_1$ and $f, r \vDash \phi_2$,

- $f, r \vDash \phi_1 U_I \phi_2$ iff there exists $t > r$ such that $t - r \in I, f, t \vDash \phi_2$ and $f, u \vDash \phi_1$ for all $u, r < u < t$.

Example 3.1.1. MTL - an Example Suppose we have a cellular signalling process where after receiving a protein signal a cell must produce another protein within a certain time frame. We will call the process of detecting a specific protein *detection* and the process of producing a protein *production*. We formulate the following example:

$$\Box_{[0,10]}(\textit{detection} \rightarrow \Diamond_{[0,1]}\textit{production}), \tag{3.1}$$

which translates to globally within the time frame $[0, 10]$, detecting the protein implies that eventually within the time frame $[0, 1]$ the production of another protein will occur. Note that all operators are time bounded.

3.2 Syntax Guided Synthesis

Definition 3. Syntax-Guided Synthesis (SyGuS) [23] Informally, functional synthesis consists of finding a function f such that a logical formula ϕ capturing the correctness of f is valid. SyGuS constrains the problem in three ways:

- the logical symbols and their semantic meaning are restricted to some *background theory*,
- the specification ϕ is limited to a first order formula in the background theory and all of its variables must be universally quantified,
- All the possible functions f are restricted by a grammar which describes all syntactic expressions.

Background Theory As proposed in [23], the syntax for writing specifications is usually a variant upon a first order logic, but formulas are also evaluated with respect to a specified background theory T. The theory provides expressiveness for all predicate symbols, functions and values for each type within the vocabulary. Most theories T require the availability of well-understood decision procedures in order to determine satisfaction modulo T.

Example 3.2.1. Background Theory An example is the background theory over the reals ($T_{\mathcal{R}}$), where variables are either real, integer, or Boolean typed, the vocabulary consists of real, integer or Boolean constants, real connectives including addition/subtraction $(+, -)$, comparison $(\leq)$, and conditionals (if-then-else, referred to as ITE). The background theory

can be a combination of logical theories, for instance, $T_{\mathcal{R}LIA}$ would combine the theory of reals with Linear Integer Arithmetic (another popular Background Theory).

Correctness Specification As proposed in [23], for the function f to be synthesized, we are given the type of f and a formula ϕ as its correctness specification. The formula ϕ is a combination of predicates from the background theory, symbols from the background theory, and the function symbol f, in a way that is type-consistent.

Example 3.2.2. Background Theory and Correctness Specification We will use the common background theory over the reals $T_{\mathcal{R}}$ which has a signature $\{0, 1, +, -, \leq, ITE\}$ and the vocabulary consists of reals and integers. We can extend this background theory to include a custom operator '$\circ$' which can be a binary operator unique to our problem domain (i.e. the binding of a protein). We shall call this new background theory $T_{\mathcal{R}\circ}$. Consider the specification Φ of a function f of type $\mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$:

$$\Phi : f(x, y) = f(y, x) \circ f(x, y) \geq 2.$$

A given function f satisfies the above specification if the quantified formula $\forall x, y.\phi$ holds or equivalently if the formula ϕ is valid. In our problem domain, one such interpretation of the correctness specification might be that the product of the binding operator applied to a function f representing a protein should be greater than the constant 2 for all inputs x, y where x, y could be anything from orientation in 2-D space to concentrations of molecules. The free variables in the specification are assumed to be universally quantified.

Set of Candidate Expressions As proposed in [23], SyGuS admits structural constraints on the set of possible functions f. The structural constraints are imposed by restricting f to the set L of functions defined by a given context-free grammar G_L. Each expression in L has the same type as that of the function f, and uses the symbols in the background theory T along with the variables corresponding to the formal parameters of f.

Example 3.2.3. Example Candidate Expressions Suppose the background theory is our custom $T_{\mathcal{R}\circ}$ and the type of function f is again $\mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$. We can restrict the set of expressions $f(x, y)$ to be expressions of the inputs by syntactically restricting the body of possible functions to expressions described by the grammar G_L below:

exp := x | y | CONSTANT | exp $\circ$ exp.

An example of such a expression on the inputs would be $f(x \circ y, 1)$. We can alternatively alter $f(x, y)$ to include conditional with our custom operator by restricting the body terms to only be conditional upon the $\circ$ operator:

$$\text{Term} := x \mid y \mid \text{CONSTANT} \mid \text{ITE(Cond, Term, Term)},$$
$$\text{Cond} := \text{Term} \circ \text{Term} < \text{Term}.$$

An example of a possible function f would be $f(x, ITE(x \circ y \geq 2, x, y))$. Grammars can be used to limit the depth or size of the desired expression given the use of constraints.

3.2.1 Summary Formulation

Given the correctness specification Φ and the set L of candidates formulations from the grammar G_L, we wish to find a candidate expression $e \in L$ such that if e is used as an implementation of the function f, the specification ϕ is valid. Substituting each occurrence of the function symbol f in ϕ with the expression e is denoted by $\phi[f/e]$. Note that e 'binds' inputs such that if f has two inputs x and y, then the occurrence $f(e_1, e_2)$ in the formula ϕ must be replaced with the expression $e[x/e_1, y/e_2]$ obtained by replacing x and y in e by the expressions e_1 and e_2, respectively. Now we can define the SyGuS problem precisely:

Given a background theory T, a typed function symbol f, a formula ϕ over the vocabulary of T along with f, and a set L of expressions over the vocabulary of T and of the same type as f, find an expression $e \in L$ such that the formula $\phi[f/e]$ is valid modulo T.

Definition 4. SKETCHing Because it becomes cumbersome to write candidate formulations $e \in L$ in terms of a function f, for example $f(x, ITE(x \circ y \geq 2, x, y))$, we will now redefine the formulation as a program $P \in L$. The grammar $L \in G_L$ can be explained within the context of a `SKETCH`, which is a grammar that allows for 'unknown' information. Informally, a candidate expression or program is a sketch if it contains holes for parameters or symbols that need to be filled.

As outlined in [27], A sketch $S \in L$ defines a set of candidate programs $\bar{S}$, which is the set of all possible programs produced by replacing the holes $H \in P$ with concrete expressions. We can define the synthesis problem therefore in terms of fillings holes. Given a sketch S, the program synthesis task is to find a completion $\vec{h}$ for the holes H in S such that $\forall x.\phi(x, [S[H := \vec{h}]](x))$ is valid modulo T where $[S]$ is the set of semantics for the sketch S.

Example 3.2.4. Sketch Example Suppose we take a syntax restricted grammar like we defined in Example 3.2.3. We can now define a grammar with holes denoted ??. These

holes can either be variables or connectives:

$$\text{exp} := \text{x} \mid ?? \mid \text{CONSTANT} \mid ?? \circ \text{exp}.$$

An example sketch might therefore be $S = ?? \circ x$. We can have operators as holes:

$$\text{exp} := \text{x} \mid \text{y} \mid \text{CONSTANT} \mid \text{exp}??\text{exp}.$$

A valid sketch would be $x??y$.

3.2.2 Optimal Syntax Guided Synthesis

A unique solution to a given synthesis problem is rare, even where there are syntactic constraints. Our simple L in Example 3.2.3 has several 'correct' instantiations, i.e. $f(x, ITE(x \circ y \geq 2, x, y))$ or $f(ITE(x \circ y \geq 4, x, y), ITE(x \circ y \geq 2, x, y))$, and in this non-optimal example, the synthesizer is free to return any one of them. But for many applications, some solutions are more desirable than others due to requirements such as memory, execution time or program size. In this instance, the synthesis is no longer a search task, but an optimisation one. Following [27], we define the optimal syntax-guided synthesis problem as additional constraints on SyGuS. The optimal program synthesis problem is the task of searching a space of candidate programs L for a lowest-cost candidate P that satisfies the given specification ϕ. The search is performed with respect to a cost function $k : G_L \to \mathbb{R}$, which assigns a numeric cost to each program $P \in G_L$ where G_L in this case is the universe of possible programs that exist within a programming language. It is important to ensure that the gradient of the cost function ensures that the solution returned is globally optimal as discussed in [27].

Definition 5. Optimal SyGuS [27] Let G_L be a programming language, and T a decidable theory. Given a specification formula $\phi(x, [G_L](x))$ in T, a cost function k, and a search space $L \subseteq G_L$ of candidate programs, the optimal SyGuS is to find a program $P \in L$ such that the formula $\forall x.\phi(x, [P](x))$ is valid modulo T, and $k(P)$ is minimal among all such programs. Note that when the cost function k is constant, optimal synthesis reduces to SyGuS.

Example 3.2.5. Cost Function Consider our theory with signature $T_{\mathcal{R}\circ} = \{0, 1, +, -, \leq, ITE, \circ\}$ and consider the following grammar:

$$\text{exp} := \text{x} \mid \text{y} \mid \text{CONSTANT} \mid \text{exp} \circ \text{exp}.$$

We define a cost function $k : S \to \mathbb{R}$ as follows:

$$k(x) = 2,\ k(y) = 3,\ k(\text{ CONSTANT }) = 1,\ k(\text{exp } \circ \text{ exp}) = 5,\ k((exp \circ exp) \circ exp) = 50.$$

Such a cost function would heavily penalise the use of nested $\circ$ expressions. An example would be a program $P = x \circ y$, which would have a cost of 10 (5 for the use of the $\circ$ operator plus 5 for use of the variables).

We now introduce a new abstraction for specifying and solving optimal synthesis problems called `metaSketches`. `metaSketches` generalize sketches as defined in Definition 4. As set out in [27], a `metaSketch` generally consists of three components: (1) a space of candidate programs, represented as a countable, ordered set of finite sketches; (2) a cost function from programs to numeric cost values, and (3) a gradient function from each cost value c to a set of sketches that may contain a program with a lower cost than c. The use of holes in a `metaSketch`, results in fine-grained control over the shape of the candidate space and leads to effective search methods for solving optimal synthesis problems. A `metaSketch` provides a method for assigning a cost function to programs and a gradient for searches towards lower-cost regions of a candidate space.

Definition 6. `metaSketches` [27] A `metaSketch` is a tuple $m = \langle S, k, g \rangle$ where:

- The space $S \subseteq L$ is a countable set of sketches in L, equipped with a total ordering relation $\preceq$.
- The cost function $k : G_L \to \mathbb{R}$ assigns a cost to each program in the language G_L.
- The gradient function $g : \mathbb{R} \to 2^S$ returns an overapproximation of the set of sketches in S that contain programs with lower cost than a given value $c \in \mathbb{R}$ where:

$$g(c) \supseteq \{s \in S | \exists \vec{h}. k(S[H := \vec{h}]) < c\}, \tag{3.2}$$

  where $\vec{h}$ is an assigment for all holes H that exist within a program.

Given a specification ϕ, a `metaSketch` $m = \langle S, k, g \rangle$ defines an instance of the optimal program synthesis problem in which the search space G_L is the union $\cup_{S \in S} \bar{S}$ of the search spaces of each sketch in the set S, and the cost function is given by k.

Example 3.2.6. MetaSketch Example Consider our grammar and cost function in Example 3.2.5. The set of possible sketches S would consist of all possible instantiations of the grammar with holes (i.e. $?? \circ ??$, $?? \circ y$, $??\ ??\ x$). We can limit the number of possible sketches by disallowing the $\circ$ operator in the list of possible holes. The cost of an assignment $\vec{h}$ would be interpreted by the synthesizer. The gradient for the cost would be in the direction of minimising the number of $\circ$ operators and use of the variable y.

3.3 SMT-ODE Solving

In syntax-guided synthesis, the synthesis problem typically reduces to a Satisfiability Modulo Theories (SMT) problem [23], which is expressed in a specified logic with respect to some background theory. The first-order background theory places restrictions upon the semantics of the program or sketch, as well as the language for specifying correctness properties.

Since we focus on continuous rather than discrete intepretations of CRNs, our approach requires expressing and solving SMT problems over the reals and containing non-linear ODEs. To this purpose, we resort to the framework of *satisfiability modulo ODEs* [133, 159, 33], which provides complete decision procedures for this theory by exploiting a relaxed notion of satisfiability. We stress that this framework allows for a continuous encoding of the CRNs semantics, thus avoiding introducing discrete approximations of its dynamics. Below, we give a brief account of the theory and an SMT problem.

Definition 7. Hybrid Automata Although this thesis does not examine an equivalence between our CRN encodings with hybrid automata, many of the components are necessary for understanding our SMT-ODE encoding of CRNs. The definition also explains the structure and terms use within an SMT-ODE program. The link between CRNs and hybrid models has been discussed extensively in [160]. A hybrid automaton $\mathcal{H}$ consists of a set of discrete states $Q = (q_1, .., q_n)$ (something referred to in the rest of the thesis as *modes*), a vector $\vec{x} = (x_1, .., x_n)^T$ of continuous variables representing the physical state of the system and jumps (or transitions) which is a multi-set $J \subseteq (Q \times Q) \cup Q$. A combined state of $\mathcal{H}$ is then a pair $(q, \vec{x})$ of a mode $q \in Q$ and a vector $\vec{x}$ of real numbers representing the physical states. Each mode q has an invariant condition, denoted by the predicate $\text{inv}_q(\vec{x})$ which defines the set of all possible values of $\vec{x}$ in mode q. Similarly, a set of initial states is expressed by using predicates $\text{init}_q(\vec{x})$. The continuous dynamics of a hybrid automaton is specified by a flow condition of the form $\frac{d\vec{x}}{dt} = \text{flow}_q(\vec{x})$ for mode q, expressing a system of time invariant ODEs for the variable $\vec{x}$. A discrete transition between two modes q and q' is specified by a jump condition of the form $\text{jump}_{q,a,q'}(\vec{x}, \vec{x}')$ which can also be identified with action $a \in A$ given a set of actions A.

Example 3.3.1. Protein Production / Degradation Model
The levels of proteins within cells are determined not only by rates of protein production, but also by rates of degradation. The degradation rate of proteins within cells varies enormously from seconds to several days and differential rates of protein degradation are an important aspect of cell regulation [161]. For an example of a hybrid system model, we use

a simple protein production/degradation model which are popular models in mathematical analysis [162].

Consider a problem where we have two types of protein, and we wish to model the following system: "a protein P is produced greater than its degradation rate until it has the same number of molecules as a second protein G, given that initially G has a greater number of molecules than P".

We wish to encode the problem such that if the value of $P < G$ then the production of P is greater than the degradation. If the value of $P = G$ then the production is equal to the degradation. The dynamics of our proteins are given by the following ODEs:

$$\frac{dP}{dt} = ((G \times pro) - (2 \times P^2 \times deg), \frac{dG}{dt} = 0\ . \tag{3.3}$$

where the equations $\frac{dP}{dt}, \frac{dG}{dt}$ represent the change in P, G with respect to time, and pro and deg are the rates of production/degradation for the first protein P. If we consider our problem has two modes, the first being when $P < G$ and the second being when $P = G$ then we can describe the jump condition between the first mode and the second as:

$$\text{jump}_{q_1,a,q_2}((pro, deg, P, G, t) = (pro', deg', P', G', t') \wedge P = G). \tag{3.4}$$

where our action a in this case is the condition $P = G$, and the continuous variables $\vec{v}$ keep the same value from one mode to the other. We can also place a further restriction that $P = G$ is an invariant within the second mode.

Definition 8. SAT Modulo ODE formula [163] We adopt a definition for this from the thesis of Eggers [163] (Chapter 1), the creator of the `iSAT`-ODE tool. A SAT Modulo ODE (SMT-ODE) formula is a quantifier-free Boolean combination of arithmetic constraints over a background theory of real, integer, and Boolean variables with bounded domains, ODE constraints over real variables, and flow invariants. In addition to this we place the following constraints upon possible formulations:

- arithmetic constraints over variables x, y, and z are of the form $x \sim \circ(y, z)$ or $x \sim \circ(y)$, where $\sim$ is a relational operator from $\{<, \leq, =, \geq, >\}$, and $\circ$ is a total unary or binary operator from $\{+, -, \times, \div, \sin, \cos, \text{pow}_{\mathbb{N}}, \exp, \min, \max\}$, with $\text{pow}_{\mathbb{N}}$ denoting a power with constant positive integer exponent and exp denoting y^z. The other operators have their standard meanings,
- simple bounds are of the form $x \sim c$ with $\sim$ as a relational operator from the above, x a variable, and $c \in \mathbb{Q}$ a constant;

- ODE constraints are given by $\dot{x}_i = dx_i/dt = f(x_1, \ldots, x_n)$ with all occurring variables x_i themselves again being defined by ODE constraints and f being a function composed of $\{+, -, \times, \div, pow_{\mathbb{N}}, exp, ln, \sqrt[\mathbb{N}]{}, sin, cos\}$. Additionally, ODE constraints have to occur under positive polarity, and
- flow invariant constraints are of the form $x(t) \leq c$ or $x(t) \geq c$ with x being an ODE-defined variable and $c \in \mathbb{Q}$ being a constant. Note that these constraints do NOT apply to more general constraints which can still be expressed as a fragment of MTL.

A SAT modulo ODE formula is specified by a variable declaration, which introduces the domain for each variable, and by an initial, transition, and a target predicate over the declared variables. The actual formula is then obtained by a k-fold instantiation of the variables and unwinding of the transition predicate in bounded model checking algorithms introduced in [164]. Flow invariants must only occur within the transition predicate and only under an even number of negations to avoid explosion of formulas under De Morgan's Law. Note that the variable t represents the global time and Δt denotes the time-step within a transition.

Definition 9. Satisfiability of SMT-ODE formulae [163] A SAT modulo ODE formula ϕ over variables $\vec{x}$ (the vector of physical states of the system) is satisfiable if and only if there exists a point in the domain of $\vec{x}$ which satisfies ϕ. For a valuation (assigned variables) e to satisfy ϕ, there must be a combination of atoms in ϕ that is satisfied by e, such that the Boolean structure of ϕ is satisfied. An arithmetic atom is satisfied by e if the evaluation of the left and right-hand-side expressions under e satisfies the constraint's relational operator.

Definition 10. δ-satisfiability [133] Given an SMT formula ϕ, and any positive rational number δ, the δ-SMT problem asks for one of the following decisions:

$$\texttt{unsat} : \phi \text{ is unsatisfiable.}$$
$$\delta - \texttt{sat} : \text{ The } \delta - \text{weakening of } \phi \text{ is satisfiable.}$$

Intuitively, a formula is δ-satisfiable if it is satisfiable under some δ-relaxation, that is, a variation of its numerical terms bounded by δ. Clearly, if a formula is satisfiable, then it is also δ-satisfiable and δ would be zero. It has been noted that this relaxation is a small price to pay in order to guarantee decidability of an SMT problem. This is because, in general, real arithmetic with non-linear functions is undecidable [133]. Algorithms for solving δ-satisfiable SMT problems are called δ-complete decision procedures, and determine whether a formula is unsatisfiable or δ-satisfiable. Importantly, unsatisfiability results are unsatisfiable for the original problem.

3.3.1 Tool Use and Behaviour

Throughout Chapters 6 and 7 we use the `iSAT`[1] and `dReach`[2] tools that supports arithmetic constraint systems involving non-linear arithmetic and ODEs. The constraints solved are a combinations of Boolean variables, arithmetic constraints over real, integer, and Boolean valued variables with bounded domains, and ODE constraints over real variables plus flow invariants. The precise SMT-ODE formulation is given in Definition 8.

Due to undecidability of the fragment of arithmetic addressed, iSAT(ODE) and dReach are sound, yet quantifiably incomplete [133]. The unsatisfiability check is based on a combination of interval constraint propagation (ICP) for arithmetic constraints, safe numeric integration of ODEs, and conflict-driven clause learning (CDCL) for manipulating the Boolean structure of the formula. This procedure investigates "boxes", i.e. Cartesian products of intervals, in the solution space until it either finds a proof of unsatisfiability based on a set of boxes covering the original domain or finds some hull-consistent box [165], called a *candidate solution box*, with edges smaller than a user-specified width $\delta > 0$. While the interval-based unsatisfiability proof implies unsatisfiability over the reals, thus rendering the procedure sound, the report of a candidate solution box only guarantees that a slight relaxation of the original problem is satisfiable (as we underline in the previous Definition 10). Within this relaxation, all original constraints are first rewritten to equi-satisfiable inequational form $t \sim 0$, with $\sim \in \{>, \geq\}$, and then relaxed to the strictly weaker constraint $t \sim -\delta$. In that sense, `iSAT` and `dReach` provide reliable verdicts on either unsatisfiability of the original problem or satisfiability of its aforementioned δ-relaxation [166], and do in principle always terminate with one of these two verdicts. Note that this is only when considering the abstract algorithms using unbounded precision rather than the safe rounding employed in their floating-point based actual implementations. For more information on on command-line parameters and tool use please refer to Appendix B.

3.3.2 Encoding Hybrid Automata in `iSAT/dReach`

In this section we describe how to encode our hybrid automaton, outlined in Example 3.3.1, as an `iSAT/dReach` problem. For an additional encoding example in `iSAT` please refer to Appendix A. In general, `iSAT` programs are broken up into the four sections outlined in Definition 8: `DECL` denotes the declaration of variables, `INIT` describes the initial conditions of the system, `TRANS` describes the transition of the system and `TARGET` describes the post conditions. `dReach` programs are defined by a series of modes which have sections: `invt` i.e. invariant conditions on the mode, `flow` which describes the flow of the mode, `jump`

[1] http://www.avacs.org/tools/isatode/
[2] http://dreal.github.io/dReach/

which describe the jumping conditions for a mode. In addition to this there is also a section `init` which describes the initial conditions and `goal` which describes the post conditions.

Variable Declarations For each state variable, we can assign either a Boolean, integer, or real valued interval with the given variable name. Variable declarations are within the `DECL` section of an `iSAT` program and at the start of a `dReach` program, within the `init` section. We can also declare any constants. In our example we declare a constant `Cmax` which represents the maximum number of protein molecules that can exist in the system. `pro` and `deg` refer to the rates of the protein `P` with `G` being the other referenced protein. `max_time` and `time` refers to t and Δt within Definition 8. In `dReach` the declaration of modes, Δt and the variable type is implicit.

```
//iSAT
define Cmax = 2;
float [0, 100] time;
float [0, 100] max_time;
boole mode_1;
boole mode_2;
float [0, 2] pro;
float [0, 2] deg;
float [0, Cmax] P;
float [0, Cmax] G;
```

```
#dReach
#DEFINE Cmax = 2;
[0, 100] time;
[0, 1] x;
[0, 2] pro;
[0, 2] deg;
[0, Cmax] P;
[0, Cmax] G;
```

Flow Declarations ODEs are defined in an intuitive fashion, where the variables' behaviour is defined for each mode (be it different or the same). In `iSAT` they are declared in the `TRANS` section. In `dReach` they are declared within each individual mode. It is important to note that constants that are reals also have to have a defining ODE behaviour. `SF` is a constant produced by our tool `CRNSynth` We use the flow defined by the ODEs in our protein production degradation model in Example 3.3.1:

```
//iSAT
mode_1 -> (d.P / d.time =
   SF*(G*pro - 2*P^2*deg);
mode_1 ->(d.G / d.time =0);
mode_1 ->(d.pro / d.time =0);
mode_1 -> (d.deg / d.time =0);
mode_2 -> (d.P / d.time =
    SF*(G*pro - 2*P^2*deg);
     ...
```

```
#dReach
{mode 1;
d/dt[P]  =
    SF*(G*pro - 2*P^2*deg);
d/dt[deg] = 0;
d/dt[pro] = 0;
d/dt[G]  = 0; };
{mode_2;
...}
```

Initial Conditions Initial conditions are implied to be the starting condition of variables at $t = 0$ and in $\texttt{mode}_0$. They are defined in the `INIT` section of `iSAT` and in the `init` section

of dReach. Again the $time$ variable is implicit within dReach. We explicitly state that the number of molecules of our protein P is less than G:

```
//iSAT
time = 0;
P < G;
```

```
#dReach
(P < G);
```

Invariant, Jump and Goal Conditions Given our problem specification, our encoded MTL specification will look like the following $\phi = ((P < G)_{t=0} \wedge (P < G)\ U_{t<T}\ \Box_{t\leq T}\ (P = G)$, where T is the maximum time allowed. We can encode such a specification within two modes. In the first mode we specify that initially $P < G$ and we place an invariant upon this mode that $P < G$. The jump condition will be as defined in Example 3.3.1, where at some point before the maximum time and the condition $P = G$ is met the program will jump from the first mode to the second. Then we specify that $P = G$ for the invariant of the second mode and that the goal is $P = G$ and time $t = T$ (so $P = G$ for all measured time, in this case 100 seconds). This is encoded as the modes $mode_1$ and $mode_2$ within iSAT and dReach. Invariants are explicitly defined in dReach under invt whilst being implicitly defined in iSAT under implication i.e. mode_1 => $(P < G)$. For any goal conditions dReach has a goal keyword and in iSAT a similar TARGET keyword exists.

3.3.3 Results

We can run the above encoding of the protein production/degradation model specified in Example 3.3.1 with either iSAT or dReach, using the command-line interface described in Section 3.3.1. In Figure 3.1 we can see an example solver output file and the plotting of the ODEs, given the substitution of the values given by iSAT to accompany it. We have also included an example solver output in Appendix B for a more verbose example.

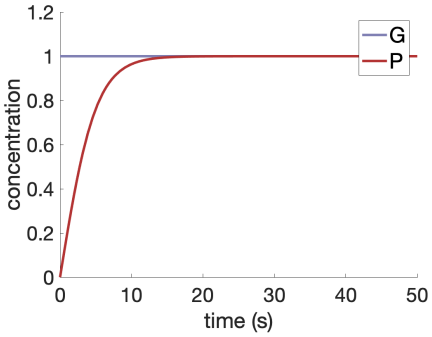
3.4 Proportional Integral Derivative Control

3.4.1 Feedback Control

Feedback control systems regulate the behaviour of dynamical systems, otherwise referred to as controlled systems. This aim, referred to as regulation, is a result of the manipulation of a system variable known as the actuated variable, which has an overall effect on the system dynamics. With the aim of achieving a particular behaviour, the actuated variable is manipulated, which, in turn, is compared against the system output. The aim of a control system is to have the regulated variable reach and remain at a desired level or set point which is known as steady-state tracking.

```
CANDIDATE SOLUTION:
    mode_1 (boolean):
        @0:  [1,1] (point interval)
        @1:  [0,0] (point interval)

    mode_2 (boolean):
        @0:  [0,0] (point interval)
        @1:  [1,1] (point interval)

    G (float):
        @0:  [1,1] (width: 1.73472348e-18)
        @1:  [1,1] (width: 0.00527359375)

    delta_time (float):
        @0:  [18.723465, 18.73045] (width: 0.0078125)
        @1:  [100,100] (width: 0.0087890625)

    P (float):
        @0:  [0,0] (width: 1.73472348e-18)
        @1:  [1,1] (width: 0.0078125)

    pro (float):
        @0:  [0.2,0.2] (width: 1.73472348e-18)
        @1:  [0.2,0.2] (width: 1.73472348e-18)

    deg (float):
        @0:  [0.1,0.1] (width: 1.73472348e-18)
        @1:  [0.1,0.1] (width: 1.73472348e-18)

Statistics.
Number of variables                  : 214
    Boolean variables                : 121
    Integer variables                : 65
    Real variables                   : 131
Number of complex bounds             : 174
.....
Time solver                          : 14.09 / 14.09 sec
```

(a)



(b)

Figure 3.1: In (a) the file output of our formulation for a simple protein production/degradation model using the tool `iSAT`. In (b) we can visualize this output by substituting the values synthesized into our ODE system and integrating over it with MATLAB. We can observe that the concentration of P rises and matches the value G which we can verify is correct according to the MTL specification provided in Section 3.3.2.

The input to the control system is a reading of a measured variable over time defined as the difference between the steady-state goal and the regulated variable. The measured variable acts as a signal within the control system, in addition to the set point goal, to generate a correction signal that is used to guide the controlled system, referred to as the plant. There are many internal working components of common control systems, referred to as a strategy. For example, an integral feedback-based strategy calculates the integral over time of the difference between the set point and the regulated variable which is then added to the overall control signal.

3.4.2 Proportional-Integral-Derivative (PID) Control

Proportional-Integral-Derivative (PID) control, a specific feedback control, has been universally accepted in both industry and academia. PID controllers are popular for two main reasons: firstly their robustness in terms of their use under various environmental conditions and secondly their simplicity, which allows for easy construction and operation. PID controllers are formed of three components, proportional, integral and derivative, which can be varied to achieve an optimal response.

Definition 11. PID Control [167] More formally, the definition of the PID controller is:

$$u(t) = ke(t) + k_i \int_0^t e(\tau)d\tau + k_d \frac{de}{dt}, \tag{3.5}$$

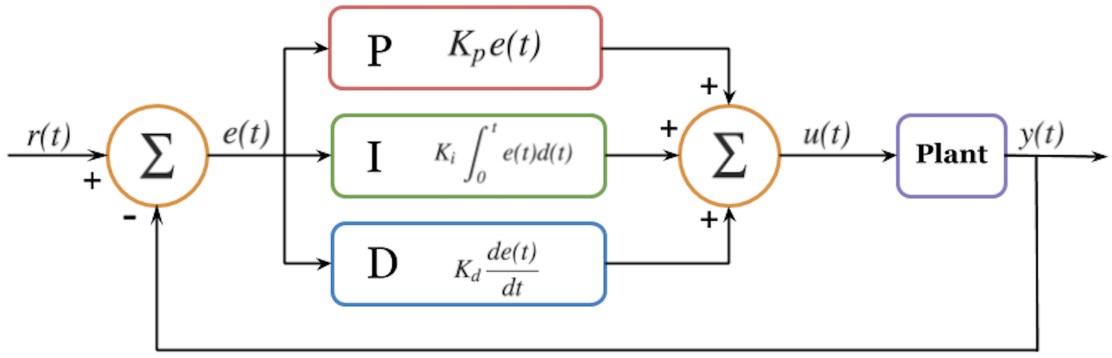


Figure 3.2: A block diagram of a PID controller in a feedback loop. $r(t)$ is the desired process value or set point, $e(t)$ is the control signal and $y(t)$ is the measured process value. The P,I and D blocks have their associated transform on the control signal detailed within them. We include a generic plant process which is controlled by the variable $u(t)$ which is a linear combination of the P, I and D components.

where u is the control signal, e is the control error ($e = r - y$). The control signal is thus a sum of three components: the P-component, which is proportional to the error, the I-component, which is proportional to the integral of the error, and the D-component which is proportional to the derivative of the error. The controller parameters are proportional gain K_p, integral gain K_i and derivative gain K_d. As a function of time:

$$u(t) = k(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}), \tag{3.6}$$

where T_i is called integral time and T_d derivative time. Intuitively, the P-component acts on the present value of the error, the I-component acts on an average of past errors and the D-component, using linear extrapolation, acts as a predictor for future errors. A block diagram for a complete PID control loop is given in Figure 3.2, where we can see how the control error ($e = r - y$) is minimized by the combination of the PID blocks.

3.4.3 Proportional Component

The proportional component is solely dependent upon on the the error term, defined as the difference between the set point and the process variable. The proportional gain K_p determines the ratio between itself and the error signal as the combined control signal. As an example, a proportional gain of 6 and an error signal of 10 would equal a combined control signal of value 60. Increasing K_p will generally result in an increase in speed of the control system response. However, increasing K_p beyond a certain point will cause the plant to oscillate. If K_p becomes even larger, the oscillations will also increase and the system will become unstable, never tracking a set-point.

The proportional controller is most commonly used in first-order plant processes to stabilize an unstable system. Proportional control is used to reduce the steady-state error of a system by driving the output of the plant to a proportional value of the reference signal. The steady-state error is inversely proportional to an increase in K_p up to a point. However, despite this, the use of proportional control alone can never entirely remove the steady state error. We can see an example of a proportional controller in Figure 3.3, where we observe, against a constant-valued reference signal, that the proportional controller stabilizes the output of the plant but fails to stabilize at the correct value.

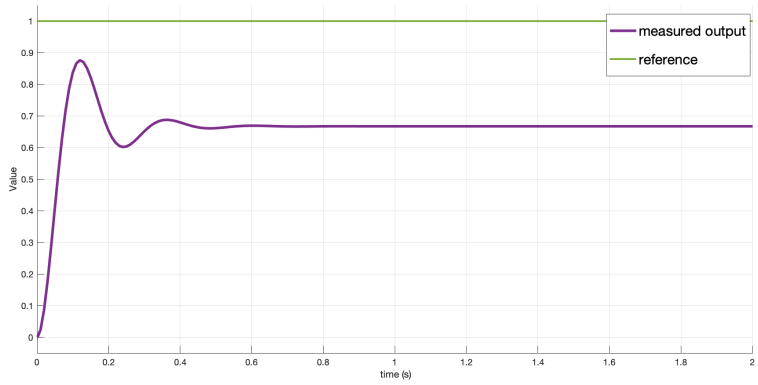


Figure 3.3: A proportional controller in a closed loop-system with a measured output and reference signal (of value 1) plotted. The plant is a simple linear system $\{\dot{x} = Ax + Bu, y = Cx + Du\}$ where u is the control signal and y is the output. We initialize the controller with $K_p = 2.675$. It can be observed that the proportional controller never quite stabilizes at the value of the reference signal.

3.4.4 Integral Component

The integral component is the result of the summation of the error over time. Given a small enough error, the integral component will slowly increase. So long as the error is greater than zero the integral response will continually increase over time and so will drive the error to zero.

The Proportional-Integral (PI) controller is used to eliminate the steady state error by combining the result of the proportional, integral and error terms. The trade-off is that the integral component has a negative impact on the response time and stability i.e. the number of oscillations. Therefore, its recommended use is within systems where speed of the controller is not essential. Oscillations result from the fact that the PI-controller has no means by which to predict future events and therefore cannot control the speed at which it rises to a set-point and hence overshoots. In Figure 3.4 we can observe a PI controller against a constant-valued reference signal.

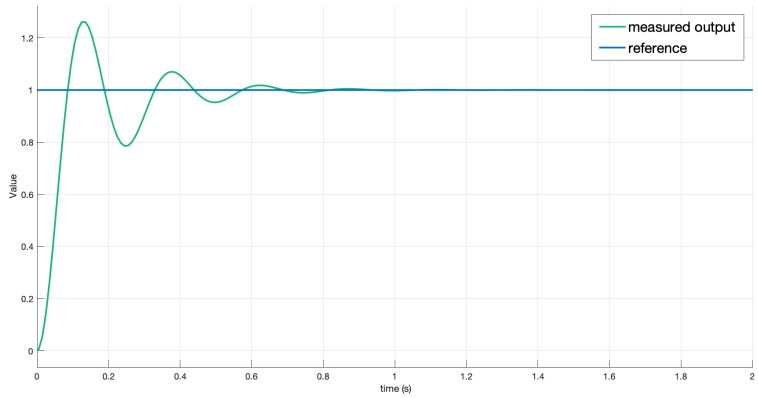


Figure 3.4: A PI controller in a closed loop system with a measured output and reference signal plotted. The plant is the same as Figure 3.3 and parameters are initialized $K_p = 2.675, K_I = 1.069$. We can observe that the PI controller drives the output towards the value of the reference signal. However, the system takes longer to stabilize.

3.4.5 Derivative Component

If the process variable rises too rapidly, as mentioned previously, the derivative component controls this rise and hence the derivative is proportional to the process variable's rate of change. As T_d increases, the reaction speed of the overall controller to the error term will also increase. The downside to this is that the derivative component is very susceptible to noise and hence most real-world control systems use a very small T_d as a proportion of the control signal. If the signal is too noisy this can lead to instability.

The PID controller is seen to have all of the advantages stated in that it has fast response, zero error, dampened oscillations and higher overall stability. Both dampened oscillations and a lack of overshoot can be attributed to the D-component within PID. We can observe a PID controller under the same conditions as the previous two controllers in Figure 3.5. We can observe that it stabilizes faster than the PI controller in Figure 3.4.

3.4.6 Tuning

Tuning is defined as the optimisation over K_p, K_i and K_d terms in order to minimise steady-state error. One such approach is the Ziegler-Nichols [168] method. This method, based upon trial and error, first sets all variables to 0. K_p is then increased until the output oscillates. As we mentioned above, when proportional gain increases, the response time of the control system increases. There is a point where K_p is sufficiently fast but also unstable. The integral term makes the system stable but also increases overshoot which will always appear if the response is sufficiently fast. The integral term is increased to the point the steady-state error as $t \to \infty$ approaches zero. The derivative term K_d is then increased in

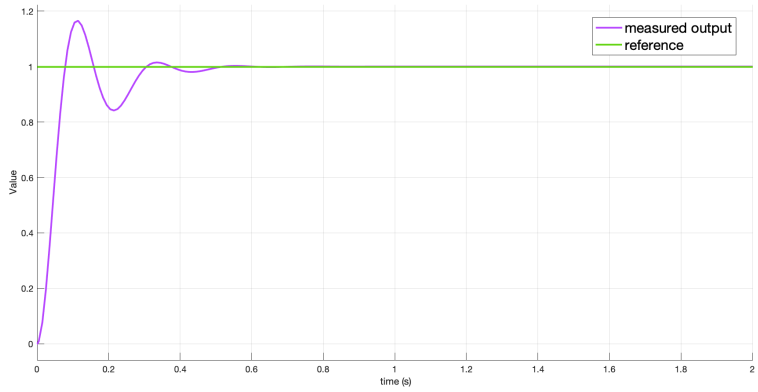


Figure 3.5: A PID controller in a closed-loop system with a measured output and reference signal plotted. We have the same linear plant as in Figure 3.3 and initialize parameters $K_p = 2.675, K_i = 1.069, K_d = 0.285$. It can be observed that the output of the plant correctly stabilizes on the reference signal and stabilizes faster than in the case of the PI controller shown in Figure 3.4 due to derivative component dampening the effect of the overshoot.

order to dampen the overshoot. The derivative term is increased until the point it is likely the system is not overly susceptible to noise.

CHAPTER 4

Chemical Reaction Networks

Contents

Chemical Reaction Networks (CRNs) are a formal language used primarily (although not exclusively) to describe molecular interaction. Like any formal language, it has both a syntax and many semantic interpretations. The literature review (Chapter 2) discusses the application of CRNs, which range from therapeutic devices to macromolecular structures. This chapter explores the syntax of CRNs in Section 4.1, ranging from the standard definitions to our own diagrammatic notation. Section 4.2 describes common semantic interpretations for CRNs including deterministic interpretations, i.e. *Mass Action*, *Hill* kinetics and *Michaelis-Menten* (MM) kinetics, and stochastic interpretations, i.e. the Chemical Master Equation (CME) and an approximation of this, the Linear Noise Approximation (LNA). CRNs are used extensively throughout this thesis to model several molecular systems known from systems biology. We provide working examples throughout which are used in the main body Chapter 5, Chapter 6, and Chapter 7. The diagrammatic notation for CRNs introduced in this chapter was developed in [15, 14] and based on [85] and Synthetic Biology Graphical Notation - Activity Flow (SBGN-AF) [169].

4.1 Syntax

Definition 12. Chemical Reaction Network A Chemical Reaction Network $\mathcal{C} = (\Lambda, \mathcal{R})$ is a pair of finite sets, where Λ is an ordered set of species, $|\Lambda|$ denotes its size, and $\mathcal{R}$ is an ordered set of reactions. Species in Λ interact according to the reactions in $\mathcal{R}$. A reaction

$\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^{|\Lambda|}$ is the *reactant complex*, $p_\tau \in \mathbb{N}^{|\Lambda|}$ is the *product complex* and $k_\tau \in \mathbb{R}_{>0}$ is the coefficient associated with the rate of the reaction. Complexes r_τ and p_τ represent the stoichiometry of reactants and products. We denote the i-th component of complex r_τ by $r_{\tau,i}$; the zero complex is denoted by $\varnothing$. Given a CRN with species set $\Lambda = \{A, B, C\}$, a reaction $([1,1,0]^T, [0,0,2]^T, k_1)$ will be denoted by $A + B \xrightarrow{k_1} 2C$. The *state change* associated to τ is defined by $\upsilon_\tau = p_\tau - r_\tau$. For example, the state change of the reaction above is $[-1,-1,2]^T$. A *chemical reaction system* (CRS) $C = (\Lambda, \mathcal{R}, x_0)$ is a tuple where $(\Lambda, \mathcal{R})$ is a CRN and $x_0 \in \mathbb{N}^{|\Lambda|}$ represents its initial condition. Since a CRS is a CRN with an initial condition, we sometimes use these terms interchangeably.

Additional Notation and Assumptions A uni-molecular reaction refers to a reaction with 0 or 1 reactants. A bi-molecular reaction refers to a reaction with strictly 2 reactants. A catalytic reaction refers to reactions of the form $A \xrightarrow{k} A + B$ or $A + C \xrightarrow{k} A + B$ where the species A in both cases acts as a catalyst. An auto-catalytic reaction is of the form $A + B \xrightarrow{k} 2A$ or $A + B \xrightarrow{k} 2A + B$, where A acts as both a catalyst and an additional product of the reaction. $\varnothing$ represents a null species which can be replaced by any equivalent species or process.

We assume that the system is well stirred [14], that is, the probability of the next reaction occurring between two molecules is independent of the location of those molecules, at fixed volume V and temperature. Under these assumptions a *configuration* or *state* of the system $x \in \mathbb{N}^{|\Lambda|}$ is given by the number of molecules of each species. Given a configuration x, we define $z = \frac{x}{N}$, where $N = V \cdot N_A$ is the volumetric factor, V is the volume and N_A Avogadro's number. Then, for $s \in \Lambda$, x_s represents the number of molecules of s in the configuration x and $\frac{x_s}{N} = z_s$ is the concentration of s in the same configuration.

Example 4.1.1. Bell Shape CRN Consider the CRN $\mathcal{C} = (\{A, B, K\}, \mathcal{R})$ where $\mathcal{R} = \{([1,0,1]^T, [0,0,2]^T, 56)\ ,\ ([0,1,1]^T, [0,2,0]^T, 43)\}$. Then $\mathcal{C}$ can be expressed by the following two bi-molecular auto-catalytic reactions:

$$A + K \xrightarrow{56} 2K$$
$$K + B \xrightarrow{43} 2B$$

Diagrammatic Language

Definition 13. Graphical Notation of CRNs A CRN can be represented as a labelled directed bipartite graph according to the usual Petri net representation with species and reaction nodes [85]. A reaction node is labelled with a rate coefficient. There is an edge from

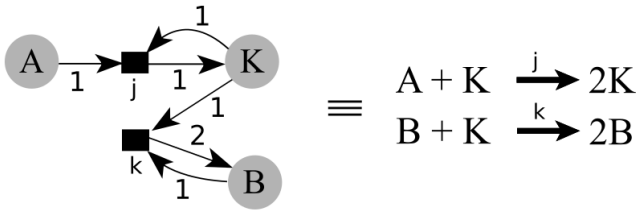


Figure 4.1: An equivalence between the reaction network $A + K \xrightarrow{j} 2K$, $K + B \xrightarrow{k} 2B$ and an equivalent Petri net diagram. In the Petri net the black boxes represent the reactions and the associated reaction rates k, j, whereas the transition labels represent the stoichiometry and the circular nodes represent the species.

a species node to a reaction node if the species is in the reactant complex, with label equal to its multiplicity; similarly, an edge from a reaction node to a species node indicates the presence of a species in the product complex. For example, in Figure 4.1 we visualise a Petri net representation for the reactions $A + K \xrightarrow{j} 2K$, $K + B \xrightarrow{k} 2B$ where circles represent species and black boxes represent reactions.

Unfortunately, this type of representation can become verbose and defeats the point of using a diagrammatic language, which is to simplify and help the reader to visualise a CRN. To visualise a CRN $C = (\Lambda, \mathcal{R})$, following [85], we represent C as a place-transition (P/T) Petri net where source/target functions are defined over arcs and vertices (or synonymously, transitions and places within a Petri net). Each arc is either a pointed arrow (↑) or a rounded arrow (⥉), with the source represented by the flat edge and the target represented by the arrow head. A ↑ represents a non-catalytic reaction and ⥉ represents a catalytic reaction. The Petri net equivalences that define these arrows are given in Figure 4.2, which is a novel contribution to diagrammatic CRN notation. The language itself however, is similar to SBGN-AF [169], with the ⥉ and □ notation denoting 'influence' found within SBGN–AF replaced with ⥉ denoting catalysis. Through these equivalences we simplify the diagrammatic notation making it easier to infer the meaning of a given CRN.

Example 4.1.2. Diagrammatic Language We provide two examples of our diagrammatic language in order to demonstrate why it is easier to use than simply listing reactions. Our first example using the diagrammatic notation below demonstrates most types of reaction. The CRN and its equivalent diagrammatic form is presented in Figure 4.3, where the rates $s, r, t, u, v \in \mathbb{R}^+$.

For the second example we reference back to Example 4.1.1. We present our CRN from Figure 4.1, referencing the previous diagrammatic example in Figure 4.3 to include the use of auto-catalytic reactions in 4.4. We abstract the rates to $j, k \in \mathbb{R}^+$.

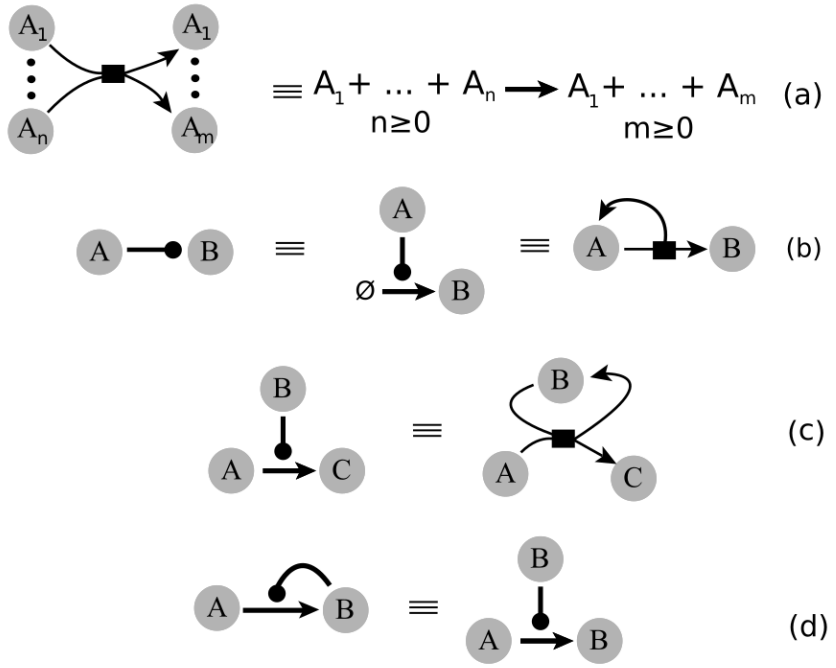


Figure 4.2: CRN diagrammatic notation and Petri net equivalences. In (a) we define multiple source/target functions as an equivalent multi-reactant and multi-product reaction. We note that in order to capture both catalytic and non-catalytic reactions a source can lead to itself. We abuse notation here in order to simplify the diagram as technically a species should only be represented once. In (b) we define a catalytic uni-molecular reaction as an equivalent Petri net. In (c) we define a catalytic bi-molecular reaction as an equivalent Petri net. This represents the reactions $A + B \rightarrow B + C$, meaning that B is the catalytic enabler for the reaction $A \rightarrow C$. This represents the reaction $A \rightarrow A + B$. In (d) we define an autocatalytic reaction of the form $A + B \rightarrow 2B$.

Definition 14. Dual-Rail Encoding In some cases a dual-rail encoding of a signal is required in order to express negative values due to the fact a CRN species is strictly non-negative. In order to handle this the so-called dual rail encoding [154] is used, by which a signal is decomposed into a "positive" and "negative" species component whilst preserving the laws of most CRN semantics that each individual species concentrations cannot be negative. Specifically, for a signal A, the two distinct component species by A^+ and A^- are denoted, representing the positive and negative signals, respectively. For further reference and examples this was used extensively in papers [15, 14, 30] and is also used in Chapter 7.

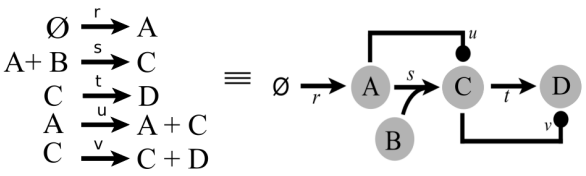


Figure 4.3: An example CRN demonstrating the various features of our diagrammatic language. We can visualise the connectivity of the CRN more intuitively in the diagrammatic version.

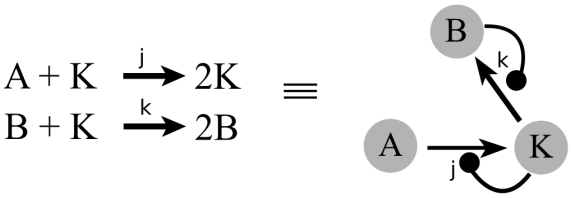


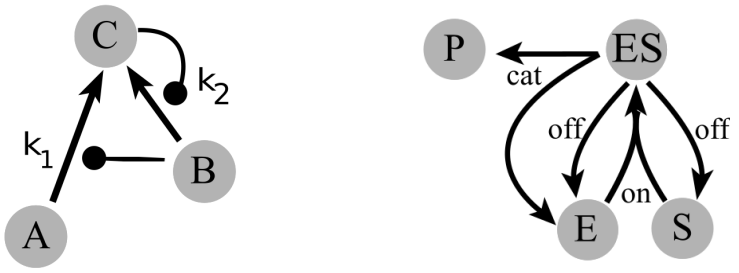
Figure 4.4: An example showing the Bellshape CRN of Example 4.1.1 in our diagrammatic language using auto-catalytic reactions.

4.2 Semantics

As a CRN is a formal language, we can imbue that language with meaning, known as the semantics. There are many different semantics for CRNs but they broadly fit into two categories: deterministic and stochastic. The deterministic interpretation takes every reaction within a CRN and interprets it as a system of differential equations, an example of which is mass action kinetics. Stochastic semantics considers the state of each species within a system as a non-negative integer and state transitions as reactions which have a non-zero probability of occuring. We also discuss a novel semantics, the LNA, given in [57]. All of these semantics can be viewed as a hierarchy in which deterministic semantics can be viewed as the simplest continuous abstraction, farthest from the physical system and stochastic semantics can be seen as the closest representation of the physical system. The LNA, like the Langevin semantics, can be seen as intermediates between the two. Under all semantics referenced here there is discussion on reachability, correctness and equivalence [85, 57, 61].

Definition 15. Deterministic Semantics

The deterministic semantics is used primarily under assumptions of high molecular counts and describe the time evolution of a CRN via a set of ODEs. Deterministic CRNs correspond to rate equations derived from the deterministic interpretation of mass action kinetics - an ODE interpretation derived from observations of molecules in solution. Let $\mathcal{C} = (\Lambda, R)$ be a



(a) (b)

Figure 4.5: In (a) a CRN with reactions $A + B \xrightarrow{k_1} B + C$ and $B + C \xrightarrow{k_2} C + C$, used as the CRN for demonstrating deterministic, stochastic and LNA examples. In (b) a CRN with reactions $E + S \xrightarrow{\text{on}} ES$, $ES \xrightarrow{\text{off}} E + S$ and $ES \xrightarrow{\text{cat}} E + P$ which is used as an example of Michaelis-Menten kinetics. Note that the reaction $ES \xrightarrow{\text{off}} E + S$ is expressed with two separate arrows with the same rate label for compactness of the diagram.

CRN. The deterministic semantics models the concentration of the species in Λ over time as a set of autonomous polynomial first order differential equations (ODEs):

$$\frac{\mathrm{d}\Phi(t)}{\mathrm{d}t} = F(\Phi(t)), \tag{4.1}$$

where $F(\Phi(t)) = \sum_{\tau=(r_\tau,p_\tau,k_\tau)\in R} \upsilon_r \alpha_{c,\tau}(\Phi(t))$ and $\alpha_{c,\tau}(\Phi(t)) = k_\tau \prod_{i=1}^{|\Lambda|} \Phi_i(t)^{r_{\tau,i}}$. $\Phi : \mathbb{R}_{\geq 0} \to \mathbb{R}^{|\Lambda|}$ describes the concentration of the species over time, therefore $\Phi(t) \in \mathbb{R}^{|\Lambda|}$ is the vector of the species concentrations at time t. Assuming $t_0 = 0$, the initial condition is $\Phi(0) = z(x_0)$, where x_0 is the initial configuration of the system. $F(\Phi(t))$ is Lipschitz continuous, so Φ exists and is unique [63].

Example 4.2.1. Consider the CRN $\mathcal{C} = (\{A, B, C\}, R)$, where R is given by the reactions in Figure 4.5a, with initial state x_0 such that $x_0(A) = 1$, $x_0(B) = 0.6$ and $x_0(C) = 0.2$, for a system with $N = 20$.

Then, the deterministic semantics of $\mathcal{C}$ is given by the following set of ODEs:

$$\begin{aligned}
\frac{dA}{dt} &= -k_1 AB, \\
\frac{dB}{dt} &= -k_2 BC, \\
\frac{dC}{dt} &= k_1 AB + k_2 BC.
\end{aligned}$$

Under the initial conditions $k_1 = 2.0, k_2 = 2.0, x_0(A) = 1.0, x_0(B) = 0.6$ and $x_0(C) = 0.2$ we can integrate over these ODEs to demonstrate a simulation of the solution seen in Figure 4.6.

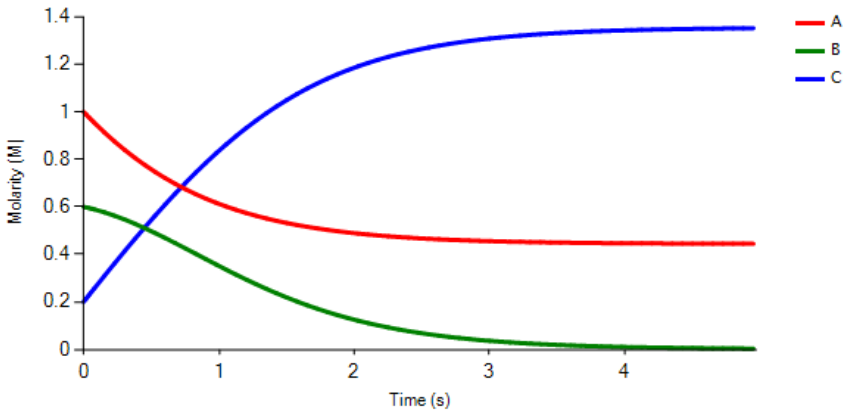


Figure 4.6: Deterministic interpretation of two catalytic reactions: $A + B \xrightarrow{k_1} B + C$ $B + C \xrightarrow{k_2} C + C$ with initial conditions, $k_1 = 2.0, k_2 = 2.0, x_0(A) = 1.0$, $x_0(B) = 0.6$ and $x_0(C) = 0.2$, the ODE representation of which is given in Example 4.2.1.

Definition 16. Stochastic Semantics The stochastic semantics are used in systems with low molecular count, as noise tends to be greater in these systems since individual reactions have greater effect on the overall system. The stochastic semantics is represented through a Continuous Time Markov Chain (CTMC), whose transient evolution can be given via the Chemical Master Equation (CME) [56].

The propensity rate α_τ of a reaction τ is a function of the current configuration of the system x, the vector molecular counts for each species, such that $\alpha_\tau(x)dt$ is the probability that a reaction event occurs in the next infinitesimal interval dt. We assume mass action kinetics, therefore:

$$\alpha_\tau(x) = k_\tau \frac{\prod_{i=1}^{|\Lambda|} r_{i,\tau}!}{N^{|r_\tau|-1}} \prod_{i=1}^{|\Lambda|} \binom{x(\lambda_i)}{r_{i,\tau}},$$

where $\lambda_i \in \Lambda$, $r_{i,\tau}$ is the i-th component of the vector r_τ and $|r_\tau| = \sum_{i=1}^{|\Lambda|} r_{i,\tau}$ [170]. Assuming fixed volume, N, the volumetric factor is a constant factor and therefore, for the sake of a simpler notation, N is considered embedded inside the coefficient k for any reaction.

Under the assumption above as well as the well stirred assumption and constant temperature, CRNs are well represented by a stochastic process whose transient analysis can be performed via the Chemical Master Equation (CME) [56], defined below.

Given a CRN $\mathcal{C} = (\Lambda, R)$ and the volumetric factor N, we define a time-homogeneous CTMC $(X^C(t), t \in \mathbb{R}_{\geq 0})$ with state space $S \subseteq \mathbb{N}^{|\Lambda|}$. Given $x_0 \in S$, the initial configuration of the system, then $P(X^C(0) = x_0) = 1$. The transition rate from state x_i to state x_j is defined as $r(x_i, x_j) = \sum_{\{\tau \in R | x_j = x_i + v_\tau\}} \alpha_\tau(x_i)$.

$X^C(t)$ describes the stochastic evolution of the molecular populations of each species in C at time t. Often, when the CRN is evident from the context we write $X(t)$ instead of

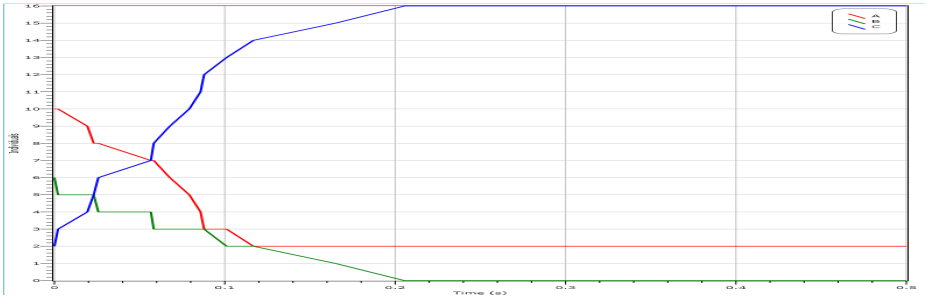


Figure 4.7: The stochastic intepretation of the CRN with reactions $A + B \xrightarrow{2} B + C, B + C \xrightarrow{2} C + C$, simulated using a CTMC with probabilities corresponding to the CME. Species A is in red, B in green and C in blue. Note that the tool we used does not support enlarging of the axes labels. The horizontal axis shows time and the vertical axis shows concentration. We can see clearly the discrete jumps for each species over time.

$X^C(t)$. For $x \in S$, we define $P^{(t)}(x) = P(X(t) = x | X(0) = x_0)$, where x_0 is the initial configuration. The CME describes the time evolution of X as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(P^{(t)}(x)\right) = \sum_{\tau \in R} \{N\alpha_{c,\tau}(x - \upsilon_\tau)P^{(t)}(x - \upsilon_\tau) - N\alpha_{c,\tau}(x)P^{(t)}(x)\}. \tag{4.2}$$

The CME can be equivalently defined in terms of the infinitesimal generator matrix Q : $|S| \times |S| \to \mathbb{R}_{\geq 0}$ as

$$\frac{\mathrm{d}}{\mathrm{d}t}P^{(t)}(x) = QP^t(x),$$

where Q is defined as $Q(x_i, x_j) = r(x_i, x_j)$ for $i \neq j$ and $Q(x_i, x_i) = -\sum_{j=1 \wedge j \neq i}^{|S|} r(x_i, x_j)$. Note that $|S|$ can be countably infinite. This representation admits computing an approximation of the CME using, for example, the sliding window method [171].

Example 4.2.2. Stochastic Semantics Example We provide an example using the stochastic semantics seen in Figure 4.7 simulated on a CTMC using the tool Visual GEC. We use the same CRN, as the deterministic semantics example i.e. a CRN with species A, B, C and reactions $A + B \xrightarrow{k_1} B + C, B + C \xrightarrow{k_2} C + C$, where $k_1 = 2, k_2 = 2$. However we change the initial conditions to $x_0(A) = 10$, $x_0(B) = 6$ and $x_0(C) = 2$ due to the fact that the stochastic interpretation operates under discrete molecular counts and therefore there are only discrete jumps in molecular count.

Definition 17. Linear Noise Approximation Semantics

The stochastic semantics of a CRN (or CRS) is generally given in terms of a CTMC described above. However, this semantics is generally not scalable for most of the practical applications because of exponential blow-up in state space [172]. As a consequence, often a continuous-state process is used [85], describing the deterministic evolution of the mean concentration values of species. However, this does not take into account the noise intrinsic

in molecular interactions. Because of this we introduce a third semantics which tries to capture the intrinsic noise within chemical solutions whilst still remaining scalable [173]. The LNA is able to capture the stochastic behaviour of a CRN, characterised in terms of a Gaussian process Y. Being a Gaussian process, Y is much easier to analyse, since, for any $t \in \mathbb{R}_{\geq 0}$, $Y(t)$ is fully determined by its expectation and covariances. These can be computed by solving a set of ODEs, and thus, without requiring state space exploration. As a consequence, the LNA semantics of CRNs we consider, though stochastic, is fully determined by the solution of a set of ODEs.

The LNA approximates the CTMC as a continuous-state Gaussian process that describe the time evolution of expectation and variance of the species. Given a CRN $\mathcal{C} = (\Lambda, \mathcal{R})$ with initial condition $x_0 \in \mathbb{N}^{|\Lambda|}$ and in a system of size N, we define the stochastic process

$$Y = N \cdot \Phi + \sqrt{N} \cdot Z,$$

where Φ is the deterministic process described in Definition 15, and Z is a zero-mean Gaussian process because we assume the initial condition is a fixed value, and Z has an associated covariance $C[Z(t)]$ described by the solution of the following ODEs with initial condition $C[Z(0)] = 0$:

$$\frac{\mathrm{d}C[Z(t)]}{\mathrm{d}t} = F'(\Phi(t), C[Z(t)]) = J_F(\Phi(t))C[Z(t)] + C[Z(t)]J_F^T(\Phi(t)) + W(\Phi(t)), \quad (4.3)$$

where $J_F(\Phi(t))$ is the Jacobian of $F(\Phi(t))$, $J_F^T(\Phi(t))$ its transpose version, and $W(\Phi(t)) = \sum_{\tau \in \mathcal{R}} \upsilon_\tau {\upsilon_\tau}^T k_\tau \prod_{S \in \Lambda} (\Phi_S)^{r_{S,\tau}}(t)$. Y is a Gaussian process with expectation $E[Y(t)] = N\Phi(t)$ and covariance matrix $C[Y(t)] = NC[Z(t)]$. As a consequence, for any $t \in \mathbb{R}_{\geq 0}$, the distribution of $Y(t)$ is fully determined by its expectation and covariance. These are computed by solving the ODEs in Equation 4.1 in Definition 15 and the equation for variance above, and thus avoiding the state space exploration. We denote by $[\![\mathcal{C}]\!]_N = (E[Y], C[Y])$ the solution of these equations for CRS $\mathcal{C}$ in a system of size N, henceforth called the *LNA model*.

Expected value and covariance matrix of $Y(t)$ are completely characterized by $\Phi(t)$ and $C[Z(t)]$ as

$$E[Y(t)] = N\Phi(t); \quad C[Y(t)] = N^{\frac{1}{2}} C[Z(t)] N^{\frac{1}{2}} = NC[Z(t)]. \quad (4.4)$$

By using the LNA we can consider stochastic properties of CRNs whilst maintaining a scalability which is comparable to the deterministic model [57]. In fact, the number of ODEs required for the LNA representation, is quadratic in the number of species and independent of the molecular counts.

Example 4.2.3. LNA Example We use the same example as the deterministic semantics, i.e. a CRN with species A, B, C and reactions $A + B \xrightarrow{k_1} B + C, B + C \xrightarrow{k_2} C + C$. We demonstrate the variance and covariance matrix produced by the LNA equations in Definition 17 in list form in order to remove the repeated symmetric parts of the matrix. We also omit the rate equations for A, B, C as they are expressed in Example 4.2.1. In the following the variables $varA$, etc. represent the variance of the species A, and the covariance, e.g. $covCB$ represents the covariance of C and B.

$$
\begin{aligned}
\frac{covCB}{dt} &= B \times covBA \times k_1 + B \times covCB \times k_2 - B \times k_2 \times varC, \\
\frac{covCA}{dt} &= -A \times covCB \times k_1 - B \times covCA \times k_1 + B \times covCA \times k_2 + \\
&\quad B \times k_1 \times varA + covBA \times (A \times k_1 + C \times k_2), \\
\frac{covBA}{dt} &= -A \times k_1 \times varB - B \times covBA \times k_1 - B \times covCA \times k_2 - C \times covBA \times k_2, \\
\frac{varA}{dt} &= -2 \times A \times covBA \times k_1 - 2 \times B \times k_1 \times varA, \\
\frac{varB}{dt} &= -2 \times B \times covCB \times k_2 - 2 \times C \times k_2 \times varB, \\
\frac{varC}{dt} &= 2 \times B \times covCA \times k_1 + 2 \times B \times k_2 \times varC + 2 \times covCB \times (A \times k_1 + C \times k_2).
\end{aligned}
$$

We can simulate the mean and variance for each of these species. Given the same initial conditions as seen in the deterministic example $x_0(A) = 1.0$, $x_0(B) = 0.6$ and $x_0(C) = 0.2$ and $k_1 = 2, k_2 = 2$, we can visualise the LNA in Figure 4.8, in which we can see the variance through the shaded regions and the mean as thick lines. The mean was previously represented in our deterministic rate equations simulated in Figure 4.6.

Definition 18. Hill and Michaelis-Menten Approximations

So far we have considered rate equations with a rate given by a single number, e.g. $A + B \xrightarrow{2} B + C$ has a rate of 2. However, we can also represent the rate as a function dependent upon the concentration of variables. We consider two such functions which can be seen as approximations of larger mass action systems. The first are the Hill equations, which refer to two closely related equations that reflect the binding of ligands to macromolecules, as a function of the ligand concentration. They are as follows:

$$
k_\tau = \begin{cases} hill(X, K_M, n) = \frac{[X]^n}{K_M^n + [X]^n}, & \text{if positive Hill reaction} \\ hill^-(X, K_M, n) = \frac{K_M^n}{K_M^n + [X]^n}, & \text{if negative Hill reaction} \end{cases} \tag{4.5}
$$

where $[X]$ is the concentration of a species X, K_M is a domain specific constant and n is the Hill coefficient that is usually decided dependent upon how expressive the approximation

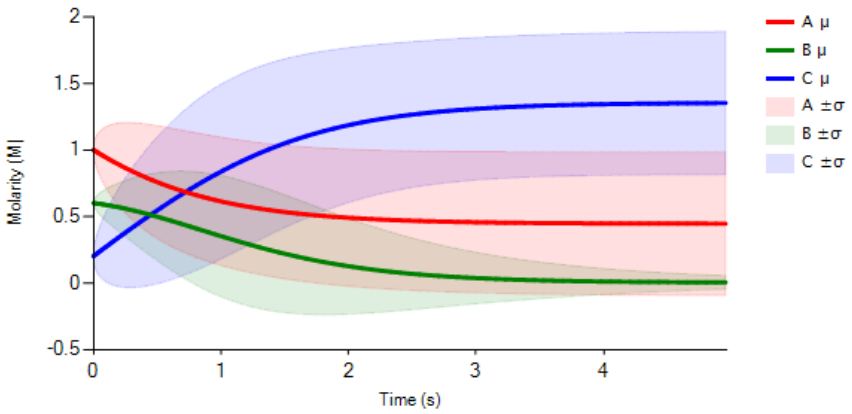


Figure 4.8: Similarly to the deterministic example in Figure 4.6, we demonstrate a reaction network $A + B \xrightarrow{2} B + C, B + C \xrightarrow{2} C + C$ but this time under the LNA semantics. Note that the shaded regions represent the variance (σ) denoted symbolically by the equations in Example 4.2.3. The mean is given by the thick lines. We can see that, because the LNA is a Gaussian process, the variance regions are symmetric.

of the system needs to be. The other kinetics that are commonly used are the Michaelis-Menten kinetics, one of the best known models for enzyme kinetics. The rate equation is as follows:

$$k_\tau = \frac{V_{\max}[S]}{K_M + [S]}. \tag{4.6}$$

The Michaelis constant K_M is the concentration $[S]$ at which the reaction rate is at half-maximum and $V_{max}[S]$ refers to the maximum concentration of species S in the system.

Example 4.2.4. Michaelis-Menten Example We provide an example of the Michaelis-Menten kinetics, in which an enzyme (E) and a substrate (S) is bound reversibly to a complex (ES) in which sometimes a protein (P) is produced. This is dependent upon three rates: 'on' which describes the enzyme-to-substrate binding rate, 'off' which describes the enzyme-to-substrate unbinding rate, and 'cat' which describes the protein production rate. The example is described by the CRN provided in Figure 4.5b or the equivalent Michaelis-Menten approximation:

$$E + S \xrightarrow{MM(S)} E + P$$

where $MM(S) = \text{cat} * \max(E) * S/((\text{off} + \text{cat})/\text{on}) + S$ (note $\max(E)$ is the largest concentration of the species E). Note that S is the only variable in this function as everything else reduces to a constant. With initial conditions $\{E : 1, S : 10, ES : 0, P : 0\}$ we get the plot in Figure 4.9.

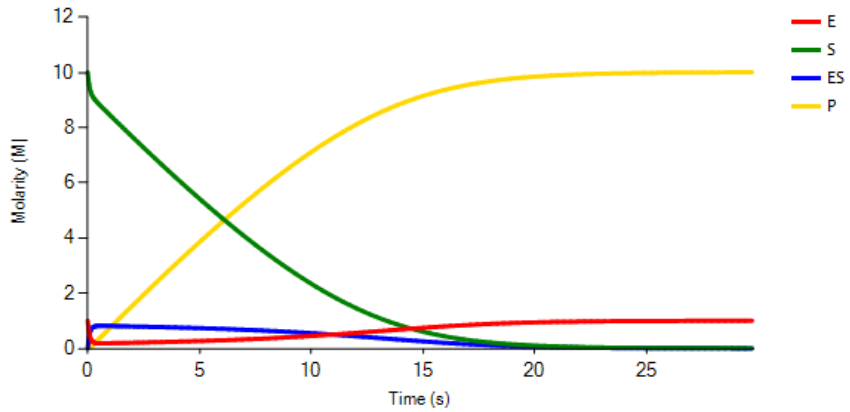


Figure 4.9: An example of the Michaelis Menten kinetics on a CRN $E + S \overset{MM(S)}{\rightarrow} E + P$ where $MM(S)$ = cat $*$ max$(E) * S/((\text{off} + \text{cat})/\text{on}) + S$ and initial conditions are $\{E : 1, S : 10, ES : 0, P : 0\}$. We observe a production of the protein P.

Simulating Semantics

Our own tool `CRNSketch` provided the symbolic equations for the examples given in this Chapter which we include a screenshot of in Figure 4.10. We use the tool *Kaemika*[1] by Prof. Luca Cardelli, which provides formatting tools for simulation plots. *Kaemika* has an easy to use language, expanding upon Microsoft's *Visual GEC* tool. Visual GEC provides a programming language, LBS, for designing and simulating any given CRN. Of the plots in this Chapter, the deterministic, the LNA and the Michaelis-Menten simulations were provided using *Kaemika*. However, for the simulation of the CME we used *Visual GEC*, as this feature is not available in *Kaemika* at the present time.

[1]Currently a Beta version is available at `https://sameapk.com/kaemika/`.

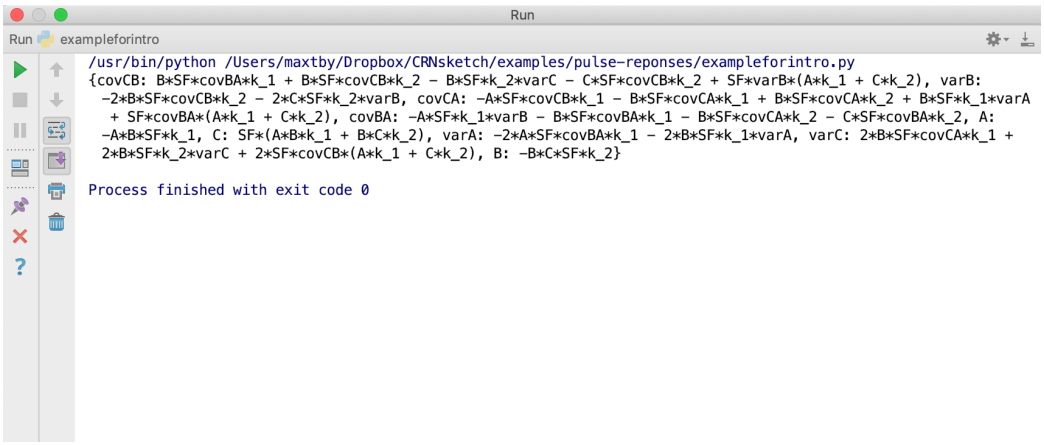


Figure 4.10: A demonstration of the output of the LNA ODEs produced by our tool `CRNSketch` based upon the CRN in Example 4.2.3. The variable 'SF' is a constant that can be ignored.

CHAPTER 5

Sketching with CRNs

Contents

In this chapter we examine the theory for Optimal Syntax-Guided Synthesis of stochastic CRNs aimed at applications in synthetic and systems biology, for example, design automation for molecular devices, or the construction of predictive molecular models. Given a syntactic template (a SKETCH, see Definition 4 in Chapter 3) our goal is to define structural constraints on CRN components and topology, a formal specification of the required behaviour, and a cost function defined over the CRN syntax, in order to simultaneously meet constraints, satisfy the specification and minimise a cost function.

Section 5.1 proposes a sketching language for CRNs that can adequately and concisely capture the syntactic constraints, while at the same time allowing under-specification of the network. Section 5.2 discusses a specification language for our synthesis procedure. Section 5.3 fits this sketching language to the `metasketch` framework outlined in Definition 6 in Chapter 3 that exploits the concept where a set of sketches is ordered according to a cost function. To ensure computational feasibility of the synthesis process, instead of relying on the discrete-state Continuous Time Markov Chain (CTMC) semantics, which is not scalable, the synthesis process employs the use of the Linear Noise Approximation (LNA) seen in

Definition 4.2.3 in Chapter 4. Therefore, Section 5.4 provides a symbolic parametric representation of the LNA and discusses how it fits within our synthesis framework. In Section 5.5 we provide novel refinement-based algorithms for the optimal synthesis of CRNs, which are employed extensively in Chapter 6. The results presented in this chapter were published in [13].

5.1 Sketching Language for Chemical Reaction Networks

CRN sketches are defined in a similar fashion to CRNs defined in Chapter 4 which we will refer to as concrete CRNs. The main difference is that species, stoichiometric constants and reaction rates are specified as unknown variables. The use of variables considerably increases the expressiveness of the language, allowing the modeller to specify additional constraints over them. Constraints facilitate the representation of key background knowledge of the underlying network, e.g. that a reaction is faster than another, or that it consumes more molecules than it produces.

Another important feature is that reactants and products of a reaction are lifted to choices of species (and corresponding stoichiometry). In this way, the modeller can explicitly incorporate in the reaction a set of admissible alternatives, letting the synthesiser resolve the choice.

Further, a sketch distinguishes between optional and mandatory reactions and optional and mandatory species. These are used to express that some elements of the network might be present and that, on the other hand, other elements must be present. Our sketching language is well suited for synthesis of biological networks: it allows expressing key domain knowledge about the network, and, at the same time, it allows for network underspecification (holes, choices and variables). This is crucial for biological systems, where, due to inherent stochasticity or noisy measurements, the knowledge of the molecular interactions is often partial.

Definition 19. Sketching language for CRNs

A CRN sketch is a tuple $\mathcal{S} = (\Lambda, \mathcal{R}, \mathrm{Var}, \mathrm{Dec}, \mathrm{Ini}, \mathrm{Con})$, where:

- $\Lambda = \Lambda_m \cup \Lambda_o$ is a finite set of species, where Λ_m and Λ_o are sets of mandatory and optional species, respectively.
- $\mathrm{Var} = \mathrm{Var}_\Lambda \cup \mathrm{Var}_c \cup \mathrm{Var}_r$ is a finite set of variable names, where Var_Λ, Var_c and Var_r are sets of species, coefficient and rate variables, respectively.

- Dec is a finite set of variable declarations. Declarations bind variable names to their respective domains of evaluation and are of the form $x : D$, where $x \in \text{Var}$ and D is the domain of x. Three types of declaration are supported:
  - Species, where $x \in \text{Var}_\Lambda$ and $D \subseteq \Lambda$ is a finite non-empty set of species.
  - Stoichiometric coefficients, where $x \in \text{Var}_c$ and $D \subseteq \mathbb{N}$ is a finite non-empty set of non-negative integers.
  - Rate parameters, where $x \in \text{Var}_r$ and $D \subseteq \mathbb{R}_{\geq 0}$ is a bounded set of non-negative reals.
- Ini is the set of initial states, that is, a predicate on variables $\{\lambda_0\}_{\lambda \in \Lambda}$ describing the initial number of molecules for each species.
- Con is a finite set of additional constraints, specified as quantifier-free formulas over Var.
- $\mathcal{R} = \mathcal{R}_m \cup \mathcal{R}_o$ is a finite set of reactions, where $\mathcal{R}_m$ and $\mathcal{R}_o$ are sets of mandatory and optional reactions, respectively. As for a concrete CRN, each $\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where in this case $k_\tau \in \text{Var}_r$ is a rate variable; the reaction complex r_τ and the product complex p_τ are sets of reactants and products, respectively. A reactant $R \in r_\tau$ (product $P \in p_\tau$) is a finite choice of species and coefficients, specified as a (non-empty) set $R = \{c_i\lambda_i\}_{i=1,\ldots,|R|}$, where $c_i \in \text{Var}_c$ and $\lambda_i \in \text{Var}_\Lambda$. We denote with f_{r_τ} the uninterpreted choice function for the reactants of τ, that is, a function $f_{r_\tau} : r_\tau \to \text{Var}_c \times \text{Var}_\Lambda$ such that $f_{r_\tau}(R) \in R$ for each $R \in r_\tau$. The choice function for products, f_{p_τ}, is defined equivalently. As an example, reaction $\tau = (\{\{c_1\lambda_1, c_2\lambda_2\}, \{c_3\lambda_3\}\}, \{\{c_4\lambda_4, c_5\lambda_5\}\}, k)$ is preferably written as $\{c_1\lambda_1, c_2\lambda_2\} + c_3\lambda_3 \xrightarrow{k} \{c_4\lambda_4, c_5\lambda_5\}$, using the shortcut $c_3\lambda_3$ to indicate the single-option choice $\{c_3\lambda_3\}$. A possible concrete choice function for the reactants of τ is the function $\overline{f_{r_\tau}} = \{\{c_1\lambda_1, c_2\lambda_2\} \mapsto c_1\lambda_1, \{c_3\lambda_3\} \mapsto c_3\lambda_3\}$ that chooses option $c_1\lambda_1$ as first reactant.

The following example illustrates the proposed sketching language and the optimal solution obtained using our synthesis algorithm introduced in Section 5.5.1.

Example 5.1.1. Consider a sketch with mandatory species $\Lambda_m = \{A, B\}$, optional species $\Lambda_o = \{C\}$ and variable declaration Dec = $\{\lambda_1, \lambda_2 : \Lambda_m, \lambda_3 : \Lambda_m \cup \Lambda_o, k : [1, 10], c_1, \ldots, c_3 : [0, 1]\}$, where $c_1, \ldots c_3 \in \text{Var}_c$ are stoichiometric variables; $\lambda_1, \ldots \lambda_3 \in \text{Var}_\Lambda$ are species variables and $k \in \text{Var}_r$ is a rate variable. Declarations indicate that: λ_1 and λ_2 range over

mandatory species; λ_3 can be any species; k ranges from 1 to 10; and $c_1, \ldots, c_3$ can be either 0 or 1. The following

$$\tau_1 = \{c_1\lambda_1\} + \{c_2\lambda_2\} \xrightarrow{k} \{c_3\lambda_3\},$$

represents a reaction with rate k involving as reactants c_1 molecules of λ_1 and c_2 molecules of λ_2, and producing c_3 molecules of λ_3. Note that mandatory species must occur in any concrete instantiation of the sketch. Therefore, possible instantiations are: $A + B \xrightarrow{1} \varnothing$ or $A \xrightarrow{9.9} B$, but not $A \xrightarrow{5} C$ since the latter does not include the mandatory species B. Valid instantiations of a CRN sketch will be formally characterised in Definition 20.

To express that the total number of molecules consumed by τ_1 is larger than those produced, we can include the additional constraint:

$$\text{Con} = \{c_1 + c_2 > c_3\}$$

, which would now rule out reaction $A \xrightarrow{9.9} B$ as a solution of the sketch. If we knew that reaction τ_1 cannot produce species A, we could either add the constraint $\lambda_3 \neq A$ in Con, or, more efficiently, restrict the range of λ_3 through the declaration $\lambda_3 : \{B, C\}$. Further, consider the reaction:

$$\tau_2 = \{c_1\lambda_1, c_2\lambda_2\} \xrightarrow{k} \{c_3\lambda_3\}.$$

Although almost identical to τ_1, τ_2 describes instead a unimolecular reaction, where the reactant is given by the choice between $c_1\lambda_1$ and $c_2\lambda_2$.

Holes Unknown information about the network also can be expressed using holes, i.e. portions of the model left "unfilled" and resolved by the synthesiser. Holes, denoted with ?, are implicitly encoded through sketch variables. To correctly interpret holes, we assume default domains, $D_r \subseteq \mathbb{R}$ bounded and $D_c \subseteq \mathbb{N}$ finite, for rate and coefficient variables, respectively. We also support the implicit declaration of variables. Starting from a general CRN sketch $\mathcal{S} = \{\Lambda, \mathcal{R}, \text{Var}, \text{Dec}, \text{Ini}, \text{Con}\}$, we show how these additional terms can be expressed in an extended sketch $\mathcal{S}' = \{\Lambda, \mathcal{R}, \text{Var}', \text{Dec}', \text{Ini}, \text{Con}\}$. We further assume a maximum stoichiometric constant $c^{\top} \in \mathbb{N}$ and a maximum reaction rate constant $r^{\mathbb{N}} \in \mathbb{R}^{+}$.

- **Implicit species variable declaration**: for $D \subseteq \Lambda$,

  $$\{\ldots, cD, \ldots\} \equiv \{\ldots, c\lambda', \ldots\},$$

  where $\lambda' \notin \text{Var}_\Lambda$, $\text{Var}'_\Lambda = \text{Var}_\Lambda \cup \{\lambda'\}$, and $\text{Dec}' = \text{Dec} \cup \{\lambda' : D\}$.

- **Implicit coefficient variable declaration**: for $D \subseteq \mathbb{N}$ finite,

  $$\{\ldots, D\lambda, \ldots\} \equiv \{\ldots, c'\lambda, \ldots\},$$

  where $c' \notin \text{Var}_c$, $\text{Var}'_c = \text{Var}_c \cup \{c'\}$, and $\text{Dec}' = \text{Dec} \cup \{c' : D\}$.

- **Implicit rate variable declaration**: for $D \subseteq \mathbb{R}_{\geq 0}$ bounded,

$$r_\tau \xrightarrow{D} p_\tau \equiv r_\tau \xrightarrow{k} p_\tau,$$

where $k \notin \text{Var}_r$, $\text{Var}_r' = \text{Var}_r \cup \{k\}$, and $\text{Dec}' = \text{Dec} \cup \{k : D\}$.

- **Species, coefficient and rate holes**:

$$\{\ldots, c?, \ldots\} \equiv \{\ldots, c\Lambda, \ldots\},$$
$$\{\ldots, ?\lambda, \ldots\} \equiv \{\ldots, [0, c^\top]\lambda, \ldots\}, \text{ and}$$
$$r_\tau \xrightarrow{?} p_\tau \equiv r_\tau \xrightarrow{(0, r^\top]} p_\tau$$

- **Reactant and product holes**:

$$? + r_\tau \xrightarrow{k} p_\tau \equiv \{c'\lambda'\} + r_\tau \xrightarrow{k} p_\tau \text{ and}$$
$$r_\tau \xrightarrow{r} ? + p_\tau \equiv r_\tau \xrightarrow{k} \{c'\lambda'\} + p_\tau,$$

where $c' \notin \text{Var}_c$, $\lambda' \notin \text{Var}_\Lambda$, $\text{Var}_c' = \text{Var}_c \cup \{c'\}$, $\text{Var}_\Lambda' = \text{Var}_\Lambda \cup \{\lambda'\}$, and $\text{Dec}' = \text{Dec} \cup \{c' : [0, c^\top], \lambda' : \Lambda\}$.

The following examples introduce the notion of implicit variable declaration and holes which are resolved by the synthesiser:

Example 5.1.2. Consider the CRN sketch of Example 5.1.1. Then,

$$\{c_1\lambda_1, [0,2]\lambda_2\} + \{c_2\{A, B\}\} \xrightarrow{[3,4]} \{c_3\lambda_3\},$$

corresponds to reaction

$$\{c_1\lambda_1, c'\lambda_2\} + \{c_2\lambda'\} \xrightarrow{k'} \{c_3\lambda_3\},$$

where c', λ', k' are fresh variables declared as $c' : [0, 2]$, $\lambda' : \{A, B\}$, $k' : [3, 4]$.

Example 5.1.3. Consider the CRN sketch of Example 5.1.1. Let $c^\top = 2$ and $r^\top = 10$. Then reaction

$$\{c_1\lambda_1, ?\lambda_2\} + ? \xrightarrow{?} \{c_3?\},$$

corresponds to reaction

$$\{c_1\lambda_1, c'\lambda_2\} + \{c''\lambda''\} \xrightarrow{k'} \{c_3\lambda'\},$$

where $c', c'', \lambda', \lambda'', k'$ are fresh variables declared as $c', c'' : [0, c^\top]$, $\lambda', \lambda'' : \Lambda = \{A, B, C\}$, $k' : [0, r^\top]$.

In addition, to simplify notation, reactants and products containing a single option $\{c\lambda\}$ as $c\lambda$ shall be rewritten. Similarly, inline declarations of variables ranging over domains with a single element, can be replaced by the same element. For instance, reaction:

$$\{c_1\lambda_1\} + \{\{1\}\{A\}\} \xrightarrow{[10,10]} \{\{2\}\{C\}, c_3\lambda_3\},$$

can be rewritten as

$$c_1\lambda_1 + A \xrightarrow{10} \{2C, c_3\lambda_3\}.$$

We now characterise when a concrete network is a valid instantiation of a sketch.

Definition 20. Sketch Instantiation A CRS, see Chapter 4 Definition 12, $\mathcal{C} = (\Lambda_\mathcal{C}, \mathcal{R}_\mathcal{C}, x_0)$ is a valid instantiation of a sketch $\mathcal{S} = (\Lambda, \mathcal{R}, \text{Var}, \text{Dec}, \text{Ini}, \text{Con})$ if: $\text{Ini}(x_0)$ holds; there exists an interpretation I of the variables in Var and choice functions, such that:

1. all additional constraints are satisfied: $I \vDash \bigwedge_{\phi \in \text{Con}} \phi$,
2. for each $\tau \in \mathcal{R}_m$ there is $\tau' \in \mathcal{R}_\mathcal{C}$ that *realises* τ, i.e., τ' is obtained from τ by replacing variables and choice functions with their interpretation. When τ' realises sketch reaction τ, its reactants $r_{\tau'}$ is a set of the form $\{c_R\lambda_R\}_{R \in r_\tau}$, i.e. containing a concrete reactant for each choice R. Then, this is readily encoded in the reactant vector form $r_{\tau'} \in \mathbb{N}^{|\Lambda|}$ as per CRN definition 4.2.3). Similar reasoning applies for products $p_{\tau'}$, and
3. for each $\tau' \in \mathcal{R}_\mathcal{C}$ there is $\tau \in \mathcal{R}$ such that τ' realises τ;

and the following conditions hold:

4. for each $\tau' = (r_{\tau'}, p_{\tau'}, k_{\tau'}) \in \mathcal{R}_\mathcal{C}$: $k_{\tau'} > 0$ and $r_{\tau'} + p_{\tau'} > 0$ and,
5. $\Lambda_m \subseteq \Lambda_\mathcal{C}$ and $\Lambda_\mathcal{C} \subseteq \Lambda_m \cup \Lambda_o$ and,
6. for each species $A \in \Lambda_\mathcal{C}$ there is $r \in \mathcal{R}_\mathcal{C}$ such that A appears in r as reactant or product.

Such an interpretation is called *consistent* for $\mathcal{S}$. For sketch $\mathcal{S}$ and consistent interpretation I, we denote with $I(\mathcal{S})$ the *instantiation of $\mathcal{S}$ through I*. We denote with $L(\mathcal{S})$ the set of valid instantiations of $\mathcal{S}$.

Condition 4. states that there are no void reactions, i.e. having null rate ($k_{\tau'} = 0$), or having no reactants and products ($r_{\tau'} + p_{\tau'} = 0$). Further, condition 6. ensures that the concrete network contains only species occurring in some reactions.

Example 5.1.4. Bell shape generator We define a sketch $\mathcal{S}$ for the bell-shape CRN of Example 4.1.1, inspired by the solution presented in [174]. For a given species K, our goal is to synthesize a CRN such that the evolution of K, namely the expected number of molecules of K, has a bell-shaped profile during a given time interval, i.e. during an initial interval the population K increases, then reaches the maximum, and finally decreases, eventually dropping to 0. The sketch for the bellshape generator is defined as follows:

$$\Lambda_m = \{K\}, \Lambda_o = \{A, B\}, \mathcal{R}_m = \{\tau_1, \tau_2\},$$
$$\mathcal{R}_o = \{\tau_3\}, \text{Dec} = \{c_1, \ldots, c_4 : [0, 2],$$
$$k_1, k_2, k_3 : [0, 0.1], \lambda_1, \lambda_2 : \{A, B\}\},$$
$$\text{Con} = \{\lambda_1 \neq \lambda_2, c_1 < c_2, c_3 > c_4\},$$
$$\text{Ini} = \{K_0 = 1 \wedge A_0 \in [0, 100] \wedge B_0 \in [0, 100]\}$$
$$\tau_1 = \lambda_1 + c_1 K \xrightarrow{k_1} c_2 K$$
$$\tau_2 = \{0, 1\}\lambda_2 + c_3 K \xrightarrow{k_2} ?\lambda_2 + c_4 K$$
$$\tau_3 = \varnothing \xrightarrow{k_3} \{\lambda_2, [1, 2]K\}$$

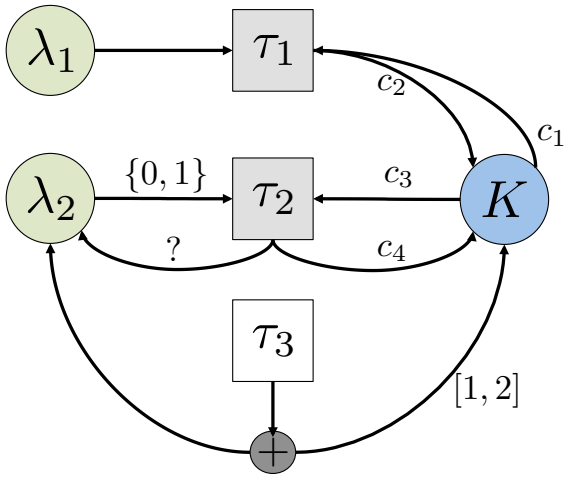


Figure 5.1: Graphical representation of the CRN sketch of Example 5.1.4. This graphical representation of sketches is used extensively within our tool `CRNSketch`. Boxes represent reactions (grey: mandatory, white: optional), while circles represent species (blue: concrete species, green: species variables). Arrows connect species and reactions, their direction indicates whether a species is consumed or produced, and are labelled with the corresponding stoichiometry. Reactants or products expressed as a choice are denoted with the $\oplus$ symbol, which is connected to the possible options by outgoing arrows.

This sketch reflects our prior knowledge about the control mechanism of the production/degradation of K. It captures that the solution has to have a reaction generating K

(τ_1) and a reaction where K is consumed (τ_2). We also know that τ_1 requires a species, represented by variable λ_1, that is consumed by τ_1, and thus τ_1 will be blocked after the initial population of the species is consumed. An additional species, λ_2, different from λ_1, may be required. However, the sketch does not specify its role exactly: reaction τ_2 consumes either none or one molecule of λ_2 and produces an unknown number of λ_2 molecules, as indicated by the hole $?$. There is also an optional reaction, τ_3, that does not have any reactants and produces either 1 molecule of λ_2 or between 1 and 2 molecules of K. The sketch further defines the mandatory and optional sets of species, the domains of the variables, and the initial populations of species. We assume the default domain $D_c = [0, 2]$, meaning that the hole $?$ can take values from 0 to 2. Note that many sketch variables are implicitly declared, e.g. term $[1, 2]K$ corresponds to $c'\lambda'$ with fresh variables $c' : [1, 2]$ and $\lambda' : \{K\}$. The sketch is summarised in a graphical representation seen in Figure 5.1 in order to provide clarity on the structure.

Variant We further specify a variant of sketch $\mathcal{S}$, $\mathcal{S}'$, where we integrate additional domain knowledge about the relative speed of reactions. Specifically, we want to synthesise networks where τ_2 is ten times faster than τ_1 and τ_1 is ten times faster than τ_3. To do so, we update the constraints of $\mathcal{S}'$ as follows:

$$\text{Con}' = \text{Con} \cup \{k_2 \geq 10 \cdot k_1, k_1 \geq 10 \cdot k_3\}.$$

Example 5.1.5. A CRS $\mathcal{C}_1 = \{\{A, K\}, \{\tau_1', \tau_2'\}, x_0\}$ where

$$\begin{aligned} x_0 &= (K_0 = 1 \wedge A_0 = 100) \\ \tau_1' &= A + K \xrightarrow{0.1} 2K \\ \tau_2' &= K \xrightarrow{0.1} \varnothing \end{aligned}$$

is a valid instantiation of $\mathcal{S}$, where reactions τ_1' and τ_2' realise respectively reaction sketches τ_1 and τ_2, but not of $\mathcal{S}'$, since reaction rates do not satisfy the additional constraints. On the other hand, a CRS $\mathcal{C}_2 = \{\{A, B, K\}, \{\tau_1', \tau_2', \tau_3'\}, x_0\}$ where

$$\begin{aligned} x_0 &= (K_0 = 1 \wedge A_0 = 100 \wedge B_0 = 1) \\ \tau_1' &= A + K \xrightarrow{0.01} 2K \\ \tau_2' &= B + K \xrightarrow{0.1} 2B \\ \tau_3' &= \varnothing \xrightarrow{0.001} K \end{aligned}$$

is a valid instantiation of both $\mathcal{S}$ and $\mathcal{S}'$, where τ_1', τ_2' and τ_3' realise respectively τ_1, τ_2 and τ_3. The corresponding consistent interpretation is $I = \{\lambda_1 \mapsto A, c_1 \mapsto 1, k_1 \mapsto 0.01, c_2 \mapsto$

$2, c_1' \mapsto 1, \lambda_2 \mapsto B, c_3 \mapsto 1, k_2 \mapsto 0.1, H \mapsto 2, c_4 \mapsto 0, k_3 \mapsto 0.001, f_{p_{\tau_3}} \mapsto \{\{\lambda_2, [1,2]K\} \mapsto [1,2]K\}, c_2' \mapsto 1\}$, where c_i' is the i-th implicit stoichiometric variable and H is the only hole. The interpretation of $f_{p_{\tau_3}}$ indicates that the choice $\{\lambda_2, [1,2]K\}$ is resolved as $[1,2]K$.

5.2 Specification Language

We are interested in verifying whether the behaviour or time evolution of a CRN satisfies constraints expressed as a logic. For this purpose, our specification language supports constraints on the expected number and variance of molecules, and, importantly, about rates of change in their behaviour over time. This allows us, for instance, to synthesise a network where a given species shows a bell-shape profile (as in Example 5.1.4), or has variance greater than its expectation (considered in Chapter 7 Section 6.3.1). As a preamble for the specification language, the logical framework over which properties and CRN sketches are interpreted and evaluated, is introduced.

Specification for CRNs

Within the synthesis problem, instantiations of a parametric CRN are compared against a specification. This specification is a broad restriction of the well established logic Metric Temporal Logic (MTL) which is defined in Chapter 3 Definition 1. The class of properties our method supports are formulas describing a dynamical profile composed as a finite sequence of phases. Each phase i is characterised by an arithmetic predicate $\mathsf{pre\text{-}post}_i$, describing the system state at its start and end points (including arithmetic relations between these two), as well as by flow invariants (formula inv_i) pertaining to the trajectory observed during the phase. Formally, a specification φ comprising $M \geq 1$ phases is defined by

$$\varphi = \bigwedge_{i=1}^{M} \mathsf{inv}_i \wedge \mathsf{pre\text{-}post}_i. \tag{5.1}$$

Note that entry as well as target conditions of phases can be expressed within $\mathsf{pre\text{-}post}_i$.

CRS correctness. For a CRS $\mathcal{C}$ (recall from Chapter 4 Definition 12), Volume N, and property φ, we are interested in checking whether $\mathcal{C}$ is *correct* with respect to φ, written $[\![\mathcal{C}]\!]_N \vDash \varphi$ (where $[\![\mathcal{C}]\!]$ is the semantics), i.e., whether $\mathcal{C}$ at Volume N exhibits the dynamic behavior required by φ. Since $[\![\mathcal{C}]\!]_N$ is a set of ODEs, this corresponds to checking whether $\hat{\varphi} \wedge \varphi_{[\![\mathcal{C}]\!]_N}$ is satisfiable, where $\varphi_{[\![\mathcal{C}]\!]_N}$ is an SMT formula encoding the set of ODEs given by $[\![\mathcal{C}]\!]_N$ and their higher-order derivatives, by means of the corresponding ODE constraints, and $\hat{\varphi}$ is the usual bounded model checking unrolling of the step relation $\bigwedge_{i=1}^{M}(phase = i \implies \mathsf{inv}_i \wedge \mathsf{pre\text{-}post}_i) \wedge phase' = phase + 1$, encoding the phase sequencing and the pertinent

phase constraints, together with the BMC target $phase = M$ enforcing all phases to be traversed. For more information on BMC encodings of ODEs see Chapter 2 of the thesis of Eggers [134]. As this satisfiability problem is undecidable in general, we relax it to checking whether $\hat{\varphi} \wedge \varphi_{[\![\mathcal{C}]\!]_N}$ is δ-satisfiable in the sense of admitting a candidate solution box of width δ. In that case, we write $[\![\mathcal{C}]\!]_N \vDash^{\delta} \varphi$.

Example 5.2.1 (Specification for the bell-shape generator)**.** This example demonstrates that we can reason over complex temporal specifications including, for instance, a relevant fragment of bounded MTL, defined in Chapter 3 Definition 1. The required bell-shaped profile for Example 5.1.4 can be formalized using a 2-phase specification, composed of phase $1 : (\mathsf{inv}_1, \mathsf{pre\text{-}post}_1)$ and phase $2 : (\mathsf{inv}_2, \mathsf{pre\text{-}post}_2)$, as follows:

$$\mathsf{inv}_1 \equiv E^{(1)}[K] \geq 0, \;\; \mathsf{pre\text{-}post}_1 \equiv E^{(1)}[K]' = 0 \;\wedge\; E[K]' > 30,$$
$$\mathsf{inv}_2 \equiv E^{(1)}[K] \leq 0, \;\; \mathsf{pre\text{-}post}_2 \equiv E[K]' \leq 1 \;\wedge\; T' = 1,$$

where $E[K]$ is the expected value of species K and $E^{(1)}[K]$ its first derivative. T is the global time. Primed notation ($E[K]', E^{(1)}[K]', T'$) indicates the variable value at the end of the respective phase. Constraints inv_1 and inv_2 require, respectively, that $E[K]$ is not decreasing in the first phase, and not increasing in the second (and last) phase. $\mathsf{pre\text{-}post}_1$ states that, at the end of phase 1, $E[K]$ is a local optimum ($E^{(1)}[K]' = 0$), and has an expected number of molecules greater than 30. $\mathsf{pre\text{-}post}_2$ states that, at the final phase, the expected number of molecules of K is at most 1 and that the final time is 1.

5.3 Cost Functions and `MetaSketching`

Previously we defined the Optimal SyGuS problem in Chapter 3 Definition 5 which is used to cost and evaluate different instantiations of the same sketch. We would like to examine the Optimal SyGuS problem for CRNs and to this end we first need to introduce the class of cost functions considered. A cost function G for a sketch $\mathcal{S}$ has signature $G : L(\mathcal{S}) \to \mathbb{N}$ and maps valid instantiations of $\mathcal{S}$ to a natural cost. A variety of interesting cost functions fit this description, and, depending on the particular application, the modeller can choose the most appropriate one. A special case is, for instance, the overall number of species and reactions, a measure of CRN complexity used in e.g. CRN comparison and reduction [175, 176]. It is important to note that the cost function is restricted to the natural numbers and not to the reals. This is necessary for completeness of the optimal synthesis algorithms defined later in the chapter. Importantly, cost functions are defined over the structure of the concrete instantiation, rather than its dynamics. As we shall see, this considerably simplifies the

optimisation task, since it leads to a finite set of admissible costs. In the rest of this chapter and indeed the next, we consider the following cost function, which captures the structural complexity of the CRN and the cost of physically implementing it using DNA [12, 7].

Definition 21. Cost function For a sketch $\mathcal{S} = (\Lambda, \mathcal{R}, \mathsf{Var}, \mathsf{Dec}, \mathsf{Ini}, \mathsf{Con})$, we consider the cost function $G_{\mathcal{S}} : L(\mathcal{S}) \to \mathbb{N}$ that, for any CRS instantiation $\mathcal{C} = (\Lambda, \mathcal{R}) \in L(\mathcal{S})$, is defined as:

$$G_{\mathcal{S}}(\mathcal{C}) = 3 \cdot (|\Lambda \cap \Lambda_o|) + \sum_{\tau \in \mathcal{R}_{\mathcal{C}}} \sum_{S \in \Lambda} 6 \cdot r_{S,\tau} + 5 \cdot p_{S,\tau}$$

, where $r_{S,\tau}$ and $p_{S,\tau}$ is the stoichiometry of species S as reactants and products of τ. The coefficients in this cost function are derived from the number of bases needed to implement them in a 2-domain strand displacement system [7].

This cost function penalizes the presence of optional species (Λ_o) and the number of reactants and products in each reaction. It does not explicitly include a penalty for optional reactions, but this is accounted for through an increased total number of reactants and products. We stress that different cost functions can be used, possibly conditioned also on the values of reaction rates.

Definition 22. Optimal synthesis of CRNs Given a sketch S, cost function $G_{\mathcal{S}}$, property φ and precision δ, the problem of optimal synthesis is to find CRS $\mathcal{C}^* \in L(\mathcal{S})$, if it exists, such that $[\![\mathcal{C}^*]\!]_N \models \varphi$ and for each CRS $\mathcal{C} \in L(\mathcal{S})$ such that $G_{\mathcal{S}}(\mathcal{C}) < G_{\mathcal{S}}(\mathcal{C}^*)$ it holds that $[\![\mathcal{C}]\!]_N \not\models \varphi$.

An important characteristic of the sketching language and the cost function is that, because the cost function only includes discrete parameters, for each sketch $\mathcal{S}$ the set $\{G_{\mathcal{S}}(\mathcal{C}) \mid \mathcal{C} \in L(\mathcal{S})\}$ is finite. This follows from the fact that $\mathcal{S}$ restricts the maximal number of species and reactions as well as the maximal number of reactants and products for each reaction. Therefore, we can define for each sketch $\mathcal{S}$ the minimal cost $\mu_{\mathcal{S}}$ and the maximal cost $\nu_{\mathcal{S}}$.

Example 5.3.1. Through substitution it can be verified that the cost of the CRS $\mathcal{C}$ of Chapter 4 Example 4.1.1, is a valid instantiation of the bell-shape generator sketch $\mathcal{S}$ in Example 5.1.4, is $G_{\mathcal{S}}(\mathcal{C}) = 3 \cdot 2 + 6 \cdot 4 + 5 \cdot 5 = 55$, and that minimal and maximal costs of sketch $\mathcal{S}$ are, respectively, $\mu_{\mathcal{S}} = 3 \cdot 1 + 6 \cdot 2 + 5 \cdot 2 = 25$ and $\nu_{\mathcal{S}} = 3 \cdot 2 + 6 \cdot 5 + 5 \cdot 7 = 71$.

We define now a meta-sketch abstraction for our sketching language that allows us to formulate an efficient optimal synthesis algorithm.

Definition 23 (`Metasketch` for CRNs)**.** Given a sketch $\mathcal{S}$ and a cost function $G_{\mathcal{S}}$, we define the `metasketch` $\mathcal{M}_{\mathcal{S}}$ as $\mathcal{M}_{\mathcal{S}} = \{\mathcal{S}(i) \mid \mu_{\mathcal{S}} \leq i \leq \nu_{\mathcal{S}}\}$, where $\mathcal{S}(i)$ is a sketch whose instantiations have cost smaller than i, i.e. $L(\mathcal{S}(i)) = \{C \in L(\mathcal{S}) \mid G_{\mathcal{S}}(C) < i\}$.

A `metasketch` $\mathcal{M}_{\mathcal{S}}$ establishes a hierarchy over the sketch $\mathcal{S}$ in the form of an ordered set of sketches $\mathcal{S}(i)$. The ordering reflects the size of the search space for each $\mathcal{S}(i)$, as well as the cost of implementations in a 2-domain DNA strand displacement system [7], of the CRNs described by $\mathcal{S}(i)$. In contrast to the meta-sketch abstraction defined in Chapter 3 Definition 6, the ordering is given by the cost function and thus it can be directly used to guide the search towards the optimal solution.

5.4 Symbolic Encoding

Given a sketch of CRN $\mathcal{S} = (\Lambda, \mathcal{R}, \mathsf{Var}, \mathsf{Dec}, \mathsf{Ini}, \mathsf{Con})$, we show that the semantics of $L(\mathcal{S})$, set of possible instantiation of $\mathcal{S}$, can be described symbolically by a set of ODEs, plus additional constraints. These equations depend on the sketch variables and on the choice functions of each reaction, and describe the time evolution of expected value and variance of the species.

Encoding of ODE functions. For $S \in \Lambda$, we define the indicator function $\mathcal{I}_S(\lambda) = \begin{cases} 1, & \text{if } \lambda = S \\ 0, & \text{otherwise} \end{cases}$. We recall that given $\tau \in \mathcal{R}$, r_τ and p_τ are the reactants and products of τ, respectively. For instance, for $\tau : c_1x_1 \xrightarrow{k} \{c_1x_2, c_2x_2\} + c_2x_1$, we have that $p_\tau = \{P_1, P_2\}$ and $r_\tau = \{R_1\}$, where $P_1 = \{c_1x_2, c_2x_2\}$, $P_2 = \{c_2x_1\}$ and $R_1 = \{c_1x_1\}$. For $S \in \Lambda$ and $\tau \in \mathcal{R}$, we define the following constants

$$r_{S,\tau} = \sum_{\substack{R \in r_\tau \\ (c,\lambda) = f_{r_\tau}(R)}} c \cdot \mathcal{I}_S(\lambda), \tag{5.2}$$

$$p_{S,\tau} = \sum_{\substack{P \in p_\tau \\ (c,\lambda) = f_{p_\tau}(P)}} c \cdot \mathcal{I}_S(\lambda), \tag{5.3}$$

$$v_{S,\tau} = p_{S,\tau} - r_{S,\tau}, \tag{5.4}$$

that are equivalent to the corresponding coefficients for concrete CRNs, but conditioned on the values of the species variables.

Example 5.4.1. Consider the following sketch composed by the only reaction $\tau : \{c_1\lambda_1, c_2\lambda_2\} + c_2\lambda_2 \to^k c_3\lambda_3$, where $\lambda_1 : \{A, B\}, \lambda_2 : \{B\}, \lambda_3 : \{A, B\}$. Call $R_1 = \{c_1\lambda_1, c_2\lambda_2\}$ and let

$(c_{ch}, \lambda_{ch}) = f_{r_\tau}(R_1)$ be the reactant picked from R_1 by the choice function f_{r_τ}. Then, we have

$$
\begin{aligned}
r_{A,\tau} &= c_{ch} \cdot \mathcal{I}_A(\lambda_{ch}) + c_2 \cdot \mathcal{I}_A(\lambda_2) \\
r_{B,\tau} &= c_{ch} \cdot \mathcal{I}_B(\lambda_{ch}) \\
p_{A,\tau} &= c_3 \cdot \mathcal{I}_A(\lambda_3) \\
p_{B,\tau} &= c_3 \cdot \mathcal{I}_B(\lambda_3).
\end{aligned}
$$

In this way we can substitute the indicator function for all $r_{S,t}, p_{S,t}$, such that expectation and covariances are defined as in the concrete case in Chapter 4 Definition 17. The practical implementations of this are discussed in the next chapter. We restrict to the set of possible valid instantiations of the symbolic encoding by imposing the following constraints.

Consistency Constraints. With the following, we restrict the possible interpretations of sketch variables in order to have consistent interpretations, i.e. those yielding only valid instantiations of a sketch (see Definition 20). Such constraints are given by the formula:

$$
\mathsf{consist} = \mathsf{Ini}(x_0) \wedge \bigwedge_{\phi \in \mathsf{Con}} \phi \wedge \bigwedge_{\tau \in \mathcal{R}_m} \neg \mathsf{void}(\tau) \wedge \bigwedge_{S \in \Lambda_m} \mathsf{used}(S). \tag{5.5}
$$

The formula states that initial state and additional constraints have to be met, that all mandatory reactions must not be void and that all mandatory species must appear in some (non-void) reactions. From condition 2. of Definition 20, recall that a reaction is void when it has a null rate or has empty reactants and products:

$$
\mathsf{void}(\tau) = (k_\tau = 0) \vee \sum_{S \in \Lambda} (r_{S,\tau} + p_{S,\tau}) = 0, \tag{5.6}
$$

where $r_{S,\tau}$ and $p_{S,\tau}$ are as per Equations 5.2 and 5.3. For $S \in \Lambda$, predicate $\mathsf{used}(S)$ is defined by:

$$
\mathsf{used}(S) = \bigvee_{\tau \in \mathcal{R}} \left(\neg \mathsf{void}(\tau) \wedge (r_{S,\tau} + p_{S,\tau}) > 0 \right). \tag{5.7}
$$

Importantly, through the above encoding, valid instantiations can be obtained as an evaluation from an SMT solver. The following proposition guarantees the correctness of the symbolic encoding with respect to the semantics given for concrete CRNs in Definition 17.

Sketch Correctness. Given an interpretation I consistent for $\mathcal{S}$, call Φ_I and $C[Z]_I$, the concrete functions obtained from Φ and $C[Z]$ by substituting variables and functions with their assignments in I. The symbolic encoding ensures that the LNA model $[\![I(\mathcal{S})]\!]_N$ of CRS $I(\mathcal{S})$ (i.e. the instantiation of $\mathcal{S}$ through I, see Definition 20) is equivalent to $(\Phi_I, C[Z]_I)$.

With reference to our synthesis problem, this implies that the synthesis of a CRS $\mathcal{C}^*$ that satisfies a correctness specification φ from a sketch $\mathcal{S}$ corresponds to finding a consistent interpretation for $\mathcal{S}$ that satisfies φ. Similarly to the case for concrete CRSs, this corresponds to checking if $\hat{\varphi} \wedge \mathsf{consist} \wedge \varphi_{[\![\mathcal{S}]\!]_N}$ is δ-satisfiable for some precision δ, where $\hat{\varphi}$ is the BMC encoding of ϕ (see Section 5.2) and $\varphi_{[\![\mathcal{S}]\!]_N}$ is the SMT encoding of the symbolic ODEs given by $[\![\mathcal{S}]\!]_N$ and the corresponding derivatives.

Cost Constraints. For a sketch $\mathcal{S}$ and cost $i \in \mathbb{N}$, the following predicate encodes the cost function of Definition 21 in order to restrict $\mathcal{S}$ into $\mathcal{S}(i)$, i.e. the sketch whose instantiations have cost smaller than i:

$$\mathsf{Con}_G(i) \equiv \left(3 \cdot \sum_{S \in \Lambda_o} \mathcal{I}(\mathsf{used}(S)) + \sum_{\tau \in \mathcal{R}} \mathcal{I}(\neg \mathsf{void}(\tau)) \cdot \sum_{S \in \Lambda} \left(6 \cdot r_{S,\tau} + 5 \cdot p_{S,\tau} \right) \right) < i,$$

where $\mathcal{I}$ is the indicator function, and used and void are predicates defined above.

5.5 Synthesis with δ-decidability over reals

In this section, we present two algorithms for the optimal synthesis of CRNs. The algorithm exploit the meta-sketch abstraction providing a cost-based ordering over the set of sketches. The algorithms builds on δ-complete decision procedure for solving SMT with ODEs. We first present a baseline algorithm that performs expensive monolithic calls of the decision procedure and then its improvement using iterative refinement of the real-valued variables and precision. Our experiments demonstrate that this improvement leads to a significant acceleration of the synthesis process.

5.5.1 Synthesis algorithm

In Algorithm 1, we present a scheme for solving the optimal synthesis problem for CRNs. It builds on the meta-sketch abstraction described in Definition 6, which enables effective pruning of the search space through cost constraints, and the SMT-based encoding of Section 5.4, which allows for the automated derivation of meta-sketch instantiations (i.e. CRNs) that satisfy the specification and the cost constraints.

This scheme repeatedly invokes the SMT solver (δ-solver) on the sketch encoding, and at each call the cost constraints are updated towards the optimal cost. We consider three approaches: 1) *top-down*: starting from the maximal cost $\nu_{\mathcal{S}}$, it solves meta-sketches with decreasing cost until no solution exists (UNSAT); 2) *bottom-up:* from the minimal cost $\mu_{\mathcal{S}}$, it increases the cost until a solution is found (SAT); 3) *binary search:* it bounds the

Algorithm 1 Generalised synthesis scheme

Require: Meta-sketch $\mathcal{M}_S$, property φ, precision δ and initial precision δ_{init}
Ensure: C^* is a solution of Definition 22 if $\exists\ C \in L(\mathcal{M}_S^\omega) : C \vDash^\delta \varphi$, otherwise C^* = null
1: $i^\top \leftarrow \nu_S$; $i^\bot \leftarrow \mu_S$; $i \leftarrow g(i^\bot, i^\top)$; $C^* \leftarrow$ null
2: **repeat**
3: $SAT_1 \leftarrow \delta$-solver$(\mathcal{S}(i), \varphi, \delta_{init})$; $SAT_2 \leftarrow$ false
4: **if** SAT_1 **then**
5: $(M, SAT_2) \leftarrow \delta$-solver$(\mathcal{S}(i), \varphi, \delta)$
6: **if** SAT_2 **then** C^* = getSoln$(\mathcal{S}(i), M)$
7: **else** $\delta_{init} = (\delta_{init} - \delta)/2$
8: **end if**
9: $(i^\bot, i^\top) \leftarrow f(i, i^\bot, i^\top, SAT_2, \mathsf{G}_\mathcal{S}(C^*))$; $i \leftarrow g(i^\bot, i^\top)$
10: **until** $i^\bot \leq i^\top$
11: **return** C^*

upper estimate on the optimal solution using a SAT witness and the lower estimate with an UNSAT witness.

We further improve the algorithm by exploiting the fact that UNSAT witnesses can also be obtained at a lower precision δ_{init} ($\delta_{init} \gg \delta$), which consistently improves performance. Indeed, UNSAT outcomes are precise and thus valid for any precision. Note that the top-down strategy does not benefit from this speed-up since it only generates SAT witnesses.

At every iteration, variable i maintains the current cost. The solver is firstly called using the rough precision δ_{init} (line 3). If the solver returns SAT (potential false positive), we refine our query using the required precision δ (line 5). If this query is in turn satisfiable, then the solver also returns a candidate solution box M, where all discrete variables are instantiated to a single value and an interval smaller than δ is assigned to each real-valued variable. Function getSoln computes the actual sketch instantiation C^* as the centre point of M that δ-satisfies φ. The cost of C^* provides the upper bound on the optimal solution. If either query returns UNSAT, the current cost i provides the lower bound on the optimal solution. The second query being UNSAT implies that the rough precision δ_{init} produced a false positive, and thus it is refined for the next iteration (line 7).

The actual search strategy used in Algorithm 1 is given by the functions f controlling how the upper ($i^\top$) and lower ($i^\bot$) bounds on the cost are updated and by g determining the next cost to explore. Note that such bounds ensure the termination of the algorithm (line 9). In the bottom-up approach, f "terminates" the search (i.e. causes $i^\bot > i^\top$) if SAT_2 is true (i.e. when the first SAT witness is obtained), otherwise f sets $(i^\bot, i^\top) \leftarrow (i + 1, i^\top)$ and g sets $i \leftarrow i^\bot$. In the top-down case, f terminates the search if SAT_2 is false (i.e. at the first UNSAT witness), otherwise it sets $(i^\bot, i^\top) \leftarrow (i^\bot, \mathsf{G}_\mathcal{S}(C^*) - 1)$ and $i \leftarrow i^\top$, where $\mathsf{G}_\mathcal{S}(C^*)$ is the cost of CRN C^*. Binary search is obtained with f that updates $(i^\bot, i^\top)$ to $(i^\bot, \mathsf{G}_\mathcal{S}(C^*) - 1)$ if SAT_2 = true, to $(i + 1, i^\top)$ otherwise, and with g that updates i to $i^\bot + \lfloor (i^\top - i^\bot)/2 \rfloor$.

5.5.2 Refinement algorithm

The key idea of the δ-IterSolver is illustrated in Algorithm 2. It also starts with the entire domain of the sketch $\mathcal{S}$, and can be seen as a variant of the first algorithm. However, it distinguishes the domain of discrete variables D^D and the domain of real-valued variables D^R. Instead of starting with the required precision δ the iterative algorithm starts with a much lower precision δ' (i.e. $\delta' \gg \delta$) obtained using the function getFirstDelta. The algorithm iteratively refines the domain of the real-valued variables D^R and increases the precision δ' in a depth-first-search fashion until a solution with the required precision δ is found or until the unsatisfiability is proved over the entire domain. This is similar to Counter-Example Guided Synthesis (CEGIS) [177, 178] in which counter-examples, in this case UNSAT solutions, restrict and guide the search.

In every step (each iteration of the main loop) the algorithm examines the domain D^D and a sub-domain of D^R using the precision δ'. If δ-solver returns $(M^D, M^R, true)$ (i.e. SAT) and $\delta' \leq \delta$ the algorithm terminates and returns CRN C^* obtained from M^D and M^R. Otherwise, a refinement step is performed. Sub-domain $D^R \setminus M^R$ and the precision δ' is pushed in the stack $\mathcal{D}$ to store a backtracking point in the search. In the next iteration of the loop the new domain M^R of the real-valued variables is explored using the precision $\delta'/2$ (the update is done at line 7).

If δ-solver proves unsatisfiability of the sketch over D^R (line 8), the current branch of the search is pruned and the algorithm backtracks to the previous refinement, i.e. it pops from the stack $\mathcal{D}$ the domain representing the complement of the sub-domain M^R from the previous refinement and the corresponding δ'.

The refinement-based algorithm can be extended in the following way. If the δ-solver returns SAT, but $\delta' > \delta$ the algorithm can try to prove the precise satisfiability over the domains M^D and M^R by negating the property φ. If it succeeds (i.e. the δ-solver returns UNSAT), it is guaranteed that C^* (obtained from M^D and M^R) satisfies φ. However, from experimental observation it has been concluded that this extension slows down the synthesis, since the precise satisfiability has not been proved in a single case and the extra calls to the δ-solver bring significant overhead and so we leave this as an extension in case the reader wishes to apply these algorithms to a different solver.

Proposition 1. Correctness and termination Both the base-line algorithm (Algorithm 1, and the refinement-based algorithm (Algorithm 1 + Algorithm 2) terminate and solve the optimal synthesis problem as stated in Definition 22.

Proof. For the base-line algorithm using the monolithic calls to the δ-solver), the proposition directly follows from the correctness and completeness of the underlying δ-solver and from

Algorithm 2 δ-IterSolver – refinement-based algorithm

Require: Sketch $\mathcal{S}$, property φ, and precision δ
Ensure: returns (C, true) such that $C \models^{\delta} \varphi$ if $\exists\ \mathcal{C} \in L(\mathcal{S}) : C \models^{\delta} \varphi$, otherwise returns $(-, \mathsf{false})$

1: $(D^D, D^R) \leftarrow \mathsf{Dom}(\mathcal{S})$; $\mathcal{D} \leftarrow \varnothing$; $\delta' \leftarrow \mathsf{getFirstDelta}(\delta)$;
2: **while** true **do**
3: $(M^D, M^R, SAT) \leftarrow \delta\text{-}\mathsf{solver}(\mathcal{S}, D^D, D^R, \varphi, \delta')$
4: **if** SAT = true **then**
5: **if** $\delta' \leq \delta$ **then return** $(\mathsf{getSoln}(\mathcal{S}, M^D, M^R), \mathsf{true})$
6: $\mathcal{D}.\mathsf{push}(D^R \setminus M^R, \delta')$
7: $D^R \leftarrow M^R$; $\delta' \leftarrow \delta'/2$
8: **else**
9: **if** $\mathcal{D}.\mathsf{isEmpty}()$ = true **then return** $(-, \mathsf{false})$
10: $(D^D, \delta') \leftarrow \mathcal{D}.\mathsf{pop}$
11: **end if**
12: **end while**

the definition of the cost-based ordering of the sketches $\mathcal{S}(i)$. Costs are discrete, ordered and finite given the bounds in line 9 Algorithm 1.

The termination of the δ-IterSolver follows from the fact that only a finite number of push operations are done and during each iteration of the main loop either a $push$ or pop operation is performed, or the return is called. Each branch of the search becomes successful after a finite number of refinements (δ' becomes smaller then δ) or is pruned, since the unsatisfiability for the corresponding sub-domain is proved.

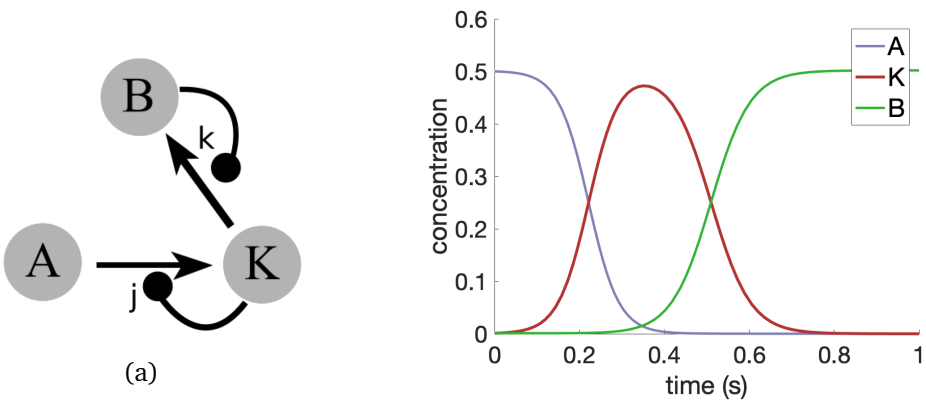
Assuming the correctness of the δ-solver, if δ-IterSolver returns (C^*, true), the CRN C^* δ-satisfies the property φ. Otherwise, the δ-IterSolver pruned all branches and backtracked to the initial node representing the entire domain of the real-valued variables. In this situation, the unsatisfiability for the whole sketch is proved and the δ-IterSolver correctly returns $(-, \mathsf{false})$.

□

Example 5.5.1. Synthesis of the Bellshape Encoding Given our sketch encoding of the bellshape profile defined in Example 5.1.4 and the cost function defined in Definition 21, we can make monolithic calls to our SMT-ODE solver of choice (see Section 3.3.1 in Chapter 3) using the above algorithms. Figure 5.2(a) demonstrates the optimal CRN computed by both algorithms by minimising the cost over possible sketches and additionally (b) represents the bell-shape profile produced by substituting the values produced by the solver into our symbolic encoding and integrating over them as per Chapter 3 Example 3.3.2.

5.6 Conclusion

This chapter presents each component that makes up a `metasketch` abstraction for CRNs. Introduced is a novel method for SMT-based optimal synthesis of CRNs with a deterministic and stochastic interpretation from MTL specifications combined with sketches. By means of the LNA, defined is the semantics of a sketch in terms of a set of parametric ODEs that are quadratic in the number of species, which allows us to reason over stochastic aspects not possible with the deterministic ODE-based semantics. Correctness is explored, specifically, what it means for a CRN to satisfy a given specification. Lastly presented, are synthesis algorithms given our SMT encoding. Presented is a biologically relevent running example [174] of a bellshape profile generator. The next chapter explores the practical implementation of this theory.



(a) (b)

Figure 5.2: In (a) we see the graphical representation of the concrete CRN instantiation in the graphical notation defined in Chapter 4 Definition 12. In (b), we observe the integration over the solution for our ODE synthesis procedures producing the bellshape profile (species K) synthesized by our algorithm.

CHAPTER 6

CRN Synthesis

Contents

The previous chapter introduced a framework for CRN synthesis with a sketching language, and encoded the optimal syntax-guided synthesis problem as an SMT-ODE problem. This chapter presents our experimental results and demonstrates the use of our tool `CRNSketch`, which generalizes the SMT-ODE problem outlined in the previous chapter. The structure of the tool will be described, and the general flow of data outlined. New features of the tool will also be introduced, specifically input-ODEs and custom kinetics. The optimal synthesis algorithms presented in the last chapter will be evaluated, and their shortcomings discussed along with the cost of synthesizing accurate and precise parameters. In Section 6.1 a general overview of the tool is given and its features summarized. A comparison with the original implementation in [13] is provided. In Section 6.2 our optimal synthesis algorithms are evaluated, and limitations of the optimal synthesis method are considered. Finally, Section 6.3 evaluates the performance of our method upon systems that have been motivated by real-world applications. An example generated file is included in Appendix A, and a solver output file included in Appendix B. The work is based upon a later version of the tool presented initially in [31], which was a collaboration with James Scott-Brown, and expands upon the experimental work outlined in [13]. My contribution is the work presented; the backend and experimentation in its entirety. The GUI and front-end is James Scott-Brown's contribution. The SMT-ODE solvers used by the tool are referenced and explained in Chapter 3 Section 3.3.1.

6.1 Implementation

In our initial work all case studies were individually constructed - including the ODE encodings, which is an error prone process. Subsequent advancements in automation of the CRN synthesis problem culminated in the tool `CRNSketch`, which overcame this problem and also added some new features which are discussed in detail in this section.

6.1.1 Tool Overview

The `CRNSketch` tool was designed to be usable not only by computer scientists, but also by biochemists and other biologists. It is composed of two modules, providing an interface designed for the needs of each community. The first is a Python library (`CRNSynthesis`[1]) which, given a parametric CRN sketch from Chapter 5 Definition 19, allows programmatic generation of a set of ODEs encoding the dynamics of the expectation and variance in concentration for each species based on the LNA semantics, and the generation of a SAT-ODE problem from a CRN sketch and MTL specification, which we will refer to as the backend. The second is a graphical user interface (GUI) that supports the design and novel representations for both CRN sketches and temporal logic specifications[2]. The GUI enables broader appeal at the cost of some flexibility with regards to specifications. In this section an in-depth overview of the backend contribution is presented, followed by summaries of the tool features for the frontend. The backend of `CRNSketch` is presented as object-oriented Python 2.7. A `Docker` container is provided for the backend via `GitHub` which, as a launcher for platform independent software, will automatically install any third-party dependencies and also install the SMT-ODE solvers used.

The core of the backend of `CRNSketch` is divided into several files `symbolicEncoding.py`, `solverParser.py` and `solverCaller.py`. Although some static functions are grouped within these files, the tool itself is object-oriented, with all files mentioned containing multiple objects, allowing for the use of this code outside of the scope of the tool. The flow of the tool is captured in Figure 6.1. The user provides an input CRN, a specification, and any additional solver constraints or information. This is converted into an equivalent CRN object and a specification object. These are then passed to the file *symbolicEncoding.py*, which in turn converts the CRN into an equivalent ODE representation. This ODE representation is then passed into the file *solverParser.py* which produces `iSAT` and `dReach` files. This is then used in the file `solverCaller.py` which utilizes our synthesis algorithm. These files are discussed in greater detail as follows.

[1] `https://github.com/max1s/CRNSynthesis/`

[2] `https://github.com/jamesscottbrown/crn-designer/`

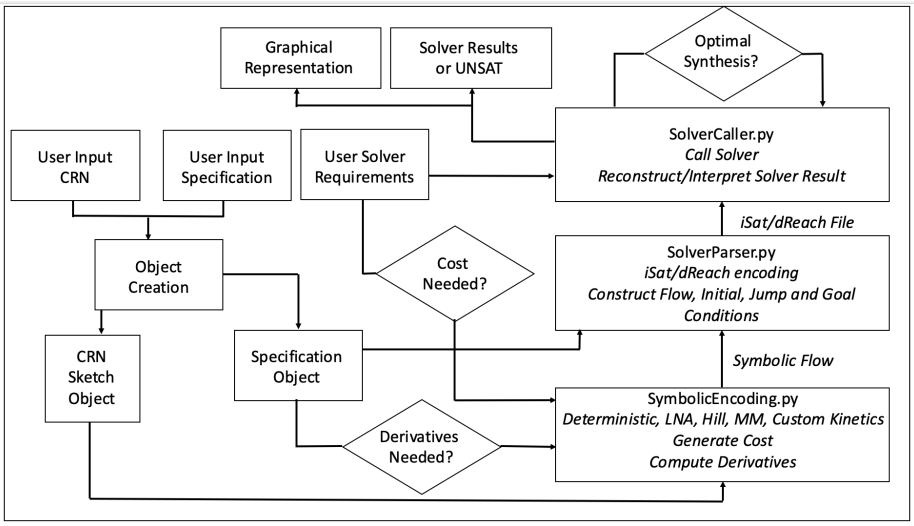


Figure 6.1: An overview of the flow of the `CRNSketch` tool. The user inputs a CRN (either diagrammatic or via the Python interface), a specification, and any other constraints upon the solver. The input goes through various transformations, through its symbolic ODE representation to an `iSAT` or `dReach` file. The solver is called multiple times according to the optimal synthesis algorithm, and both a plot and results file are given as output.

The file `symbolicEncoding.py` takes a CRN as input and converts it into a symbolic ODE representation. The file provides several classes and methods for calculating the cost of a CRN given a cost function and additionally provides methods for calculating a symbolic representation of the derivative of a species within a CRN, if the specification requires. The function *flow()* takes the CRN object and unpacks it, assigning unique tokens to all of the variables, before calculating the appropriate symbolic deterministic encoding. In the case of the deterministic, Michaelis-Menten, and positive and negative Hill equations, the conversion from the CRN object to these equations is explicitly defined. However, there is also support for custom kinetics which treats the rate parameter as a user-defined function, given as a string before it is interpreted symbolically. The function *LNAflow()* takes these ODEs and calculates the LNA based upon this. The LNA can be produced from any set of deterministic rate equations produced by the previous function, although we cannot say whether this is meaningful beyond its use in mass-action, Hill and Michaelis-Menten kinetics. Once this is complete, all necessary derivatives are calculated through the *derivative()* function, which takes the flow as input. As a result, the file returns a flow dictionary which has species, their covariances, and their derivatives coupled with the appropriate ODE.

The file `solverParser.py` takes the flow dictionary produced by the symbolic encoding and, through a series of syntactic rules, transforms this flow into an equivalent `iSAT` or `dReach` problem encoding. The file is divided into four classes representing the different

sections discussed in Chapter 3 Section 3.3.2: *Declaration*, which deals with the declaration of variables and their associated type, as well as the parameter range, *Initial* which deals with initial conditions upon species and conditions from the specification, *Transition* which deals with translating the flow dictionary into the equivalent representation for each of the solvers and *Post* which deals with any goals or post conditions on the program. The *Transition* class is instantiated multiple times dependent on the number of modes with each jump condition, specified in the specification, linking to the next object. Within each of these four classes are two functions which deal with the different syntactic construction for each solver.

`solverCaller.py` is used to call either the `iSAT` or `dReach` solver, employ the algorithms demonstrated in Chapter 5 Section 5.5.1, and to display results to the user. All functions in this file are duplicated with each being slightly different dependent on whether the solver in question is `iSAT` or `dReach`. The function *callSolver()* takes the name of the solver file produced by `solverParser.py`, solver parameters such as the δ-value which is specified by the user, and executes the calling of the solver on the command-line. The function *optimalSynthesisWithSpecifiedCost()* takes a maximum and minimum cost, employs either the top-down, bottom-up or binary search strategies outlined in Chapter 5 Section 5.5.1. With each iteration, the variable $COST$ is edited within the `iSAT` or `dReach` file to reflect the cost of the optimal synthesis. The function *getCRNValues()* takes the results files from the solver and either interprets it as an UNSAT result, or uses regular expressions to extract the values for the parameters and create a dictionary of parameter-value pairs. The function *simulate_solutions()* takes the dictionary of parameter values for each parameter variable, takes the average value, substitutes the parameters into the initial symbolic ODEs, and integrates over them in order to produce an ODE plot. It can be specified within this function whether to simulate variance and plot times for mode switches as shaded regions or not.

The CRN object is a collection or, more accurately, the top of a hierarchy of objects that is outlined in Figure 6.2. Each component of a reaction has a separate class that has minimum and maximum parameter values. In the case of stoichiometry this is constrained to integers, whereas for lambda species this is constrained to a subset of species present within the reaction network. Stoichiometry is combined with species to produce a *Term* object that is used as reactants and products in the *Reaction* class. Reactions can either be mandatory or optional as defined in Chapter 5 Definition 19. The transformation of the specification into an object happens in `solverparser`, and is an object that contains pre-conditions, invariants, and post conditions for various modes.

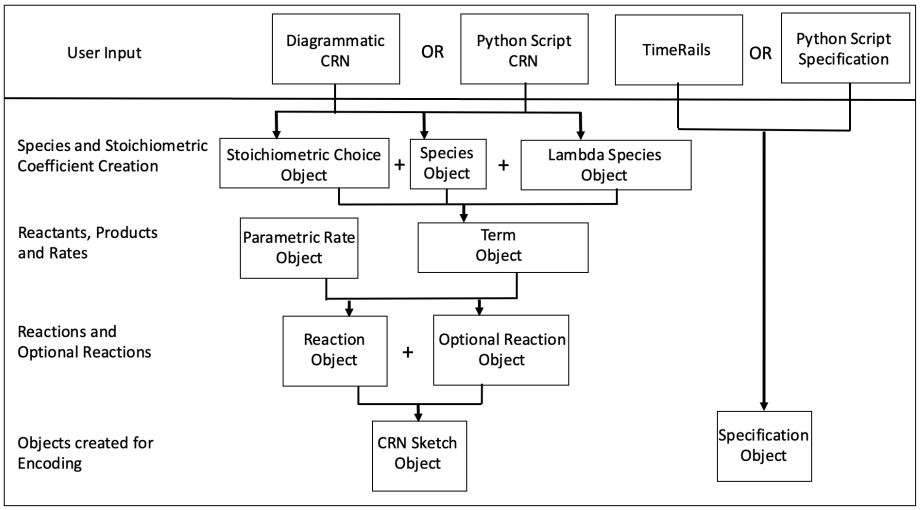


Figure 6.2: The object hierarchy of the CRN object in which each individual component of the CRN sketch is instantiated with a minimum and maximum parameter value, before being combined as input to the next level. Each object has a unique ID and symbolic representation. Included in addition to the input CRN, is the other user input, the specification, which is transformed into a specification Object.

6.1.2 ODE Encoding Outline

In this section, an outline of the `flow` algorithm responsible for the encoding of ODEs in Algorithm 3 is given. Firstly, all of the propensities for all reactions are collected and placed into a vector called `propensities` (α in Chapter 4 Definition 15). The net reaction change is then calculated and placed into the vector `stoichiometry_change` (see net change in CRNs Chapter 4 Definition 12). The transpose of `stoichiometry_change` is then multiplied by `propensities` to give the deterministic semantics representation of the ODE (`dSpeciesdt`), which is in turn zipped into a dictionary a along with a corresponding species e.g. $\{A : -k_1AB\}$. If the LNA is required, the Jacobian of the deterministic ODEs — the symbolic covariance matrix of the vector of species and G as given in Chapter 4 Definition 17 — is calculated. The naming convention for the covariance matrix is `varA` for the variance of a species `A`, and `covAB` for the covariance of a species `A` and `B`. This is added as the covariance to the dictionary a. The derivatives are subsequently calculated for each species where the derivative is needed in the specification. These are calculated by applying the chain rule symbolically to the ODEs. The derivatives are then appended to the dictionary a before a is returned.

Algorithm 3 Overview of Flow Algorithm for ODE Encoding of a CRN Sketch

procedure CALCULATEFLOW(crn,isLNA, derivatives)
    propensities ← [reaction.get_propensity() for reaction in crn.reactions]
    stoichiometry_change ← parametricNetReactionChange()
    dSpeciesdt ← stoichiometry_change^T * propensities
    a ← newDictionary(crn.species, dSpeciesdt)
    if isLNA **then**
        J ← dSpeciesdt.jacobian(crn.species)
        C ← generateCovarianceMatrix(crn.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]
        end for
    end if
    derivative_expressions ← compute_derivatives(derivatives, a)
    a.append({derivatives, derivative_expressions})
    Return a
end procedure

Representing CRNs

Within Python Within the CRNSketch tool, we can define a CRN declaratively, using the objects outlined in Figure 6.2. An example of this is given in Figure 6.3. Species are declared with a name and an optional initial value. We can also define lambda choices between species and, in general, all features of our sketching language from Chapter 5, Definition 19. Reactions have reactants, products and a parameterized rate. The CRN is returned with sets of mandatory and optional reactions, and input species.

```python
def form_crn():
    A = Species('A', initial_value=0.1)
    B = Species('B', initial_value=0.1)
    lam1 = LambdaChoice([A, B], 1)

    r1 = Reaction([], [(A, 2)], RateConstant('k_1', 0, 1))
    r2 = Reaction([(A, 1)], [], RateConstant('k_2', 0, 1))

    return CRNSketch([r1, r2], [], [])
```

Figure 6.3: Demonstration of an example user input in our Python package for the Poisson example outlined in Section 6.3.1. We define two species A and B. `lam1` represents a lambda choice between A and B. Two reactions are given, where A is a product in the first reaction and a reactant in the second. We also define two rate constants k_1 and k_2 in each respective reaction.

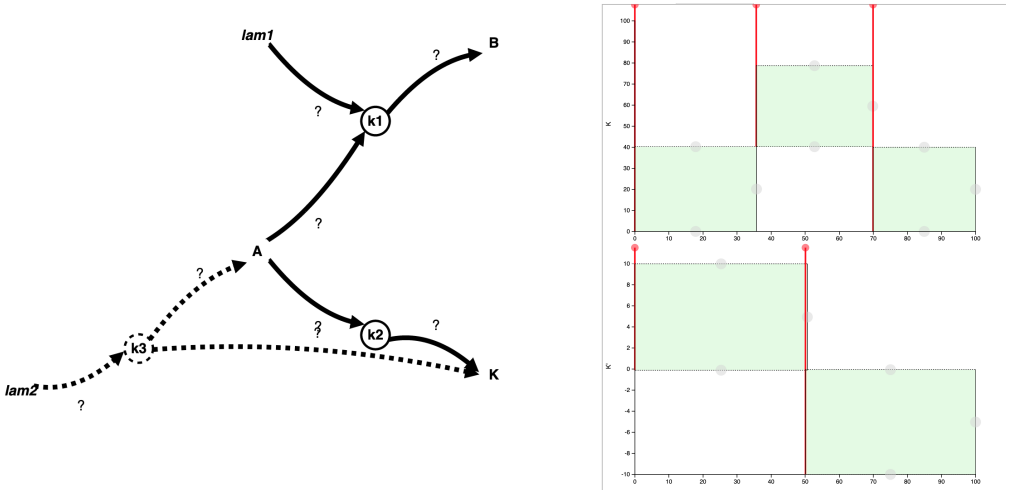
Visually Within the `CRNSketch` tool a CRN sketch is represented visually (see Figure 6.4a) as a directed bi-partite graph in which nodes represent both reactions (with a circular border) and species (with no border). Edges link reactants to reactions, and reactions to products, and are labelled by the corresponding stoichiometric coefficient (which may be expressed as 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. These pairs are represented by a dummy node labelled 'or' with a separate link from (to) each alternative reactant (product) labelled with the corresponding stoichiometry. A species can be designated as an 'input', in which case its expected concentration is given as a function of time by a sum of user-selected (bell-shaped, linear, or sigmoidal) terms, rather than by mass-action kinetics. Such a species can appear as a reactant, in which case it influences the rate of the corresponding reaction, but is effectively not consumed. The user can automatically switch between two alternative representations: in one, each species is represented by exactly one node, and in the other, species are duplicated as necessary to allow each reaction to be represented by an isolated subgraph. Figure 6.4a demonstrates a GUI representation of a subset of the bellshape sketch discussed in Chapter 5 Section 5.1.4. Within this example we demonstrate how reactions, optional reactions, species and choices are represented graphically within the `CRNSketch` tool.

Representing Specifications

Within Python Within the tool, specifications are stored within the specification Object. Specifications are declared as a set of modes each with a pre-condition, an invariant, and a post-condition as per Chapter 5 Section 5.2 (modes are also defined for Hybrid Automata in Chapter 3 Definition 7). Within each of these sections we use a subset of MTL logic which is defined in Chapter 3 Definition 1. This subset is syntactically similar but we have to express the 'until' operator through a series of 'and' constraints due the SMT-solver syntax. More formally we can define the grammar that represents specifications as the following recursive grammar:

$$\begin{aligned} \langle \textit{specification} \rangle \quad &::= \langle \textit{mode} \rangle \\ &\mid \langle \textit{mode} \rangle \text{ Until } \langle \textit{mode} \rangle \end{aligned}$$



(a) (b)

Figure 6.4: (a) provides a bellshape CRN sketch using the tool GUI. Each reaction is represented by a circle with a reaction rate inside. Each species connects through a reaction. The dotted line represents an optional reaction, the question marks represent a choice variable and we have included `lam1` and `lam2` to represent a species choice where `lam1` is a choice between A or B and `lam2` is a choice between B or K. The reactions represented are $\lambda_1 + A \xrightarrow{k_1} B, 2A \xrightarrow{k_2} K$ and $\lambda_2 \xrightarrow{k_3} A + K$. (b) provides an example specification using timerails for a bellshape profile. The coloured regions represent the mode switches. The upper plot represents the desired concentration of the species K over time (with K on the y-axis and time on the x-axis). At first the value of K is below a threshold, before rising above a threshold and then returning lower again. The lower plot represents the desired behaviour for the derivative of the species K over time (dK/dt on the y-axis and time on the x-axis). At first the derivative must be positive and then after some time negative.

$$
\begin{aligned}
\langle \textit{mode} \rangle &::= \langle \textit{term} \rangle \\
&\mid \langle \textit{term} \rangle \text{ or } \langle \textit{term} \rangle \\
&\mid \langle \textit{term} \rangle \text{ and } \langle \textit{term} \rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle \textit{term} \rangle &::= \langle \textit{quantity} \rangle \leq \langle \textit{number} \rangle \\
&\mid \langle \textit{quantity} \rangle \geq \langle \textit{number} \rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle \textit{quantity} \rangle &::= \langle \textit{variable} \rangle \\
&\mid \text{derivative of } \langle \textit{variable} \rangle
\end{aligned}
$$

where *variable* is either the expected concentration of a species, or the variance in concentration of a species. An example from within our tool is the bellshape specification outlined in Chapter 5 Example 5.2.1, which in our tool would be represented by the following:

```
specification_bellshape = [('K < 0.1', 'dK_dt >= 0',
   '(and (K > 0.4) (dK_dt = 0) )'), ('','dK_dt <= 0', 'K < 0.2')]
```

where there are two modes each with a pre-condition, an invariant, and a post condition. Here for example, the pre-condition on mode 1 states that a species $K < 0.1$, the invariant states that the derivative of K must be greater than or equal to zero, and the post condition (which would be the jump condition between modes) states that the value of $K > 0.4$ and the derivative is zero.

Visual In the GUI, a diagrammatic representation is employed that is a restriction of the `TimeRails` system [31] to the fragment of logic that `CRNSketch` 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 rectangle, indicating that the value of the corresponding term should lie in the corresponding range. Importantly, the precise time range during which the condition must hold is not directly specified, but it is possible to link rectangles in order to express requirements about their ordering. 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. In a subplot this is indicated by horizontal alignment of the two rectangles, which are linked by a vertical 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 in a subplot.

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 different constraints on the state. An example of this is visible in Figure 6.4b where a `TimeRails` specification of the bellshape encoding was created.

The graphical representation has limitations, for instance it does not support the 'or' operator within a specification as multiple `TimeRails` plots cannot be created for the same species. The Python interface however, is more flexible, and allows a *term* to be a more complicated inequality involving algebraic expressions including one or more *quantity*. The tool allows for reasoning both over the deterministic expectation of a species but also reasoning over variance (as in Example 6.3.1).

6.1.3 Cost Function

The user is able to employ our previously defined cost function related to the complexity of implementation of the CRN in DNA (see Chapter 5 Definition 21). However, the user can opt for a custom cost function to guide the optimal synthesis algorithm. This cost function is simplified due to the fact the provided cost function includes references to the total number of reactions and species which have to be calculated by the tool. Currently the cost function is entered in the GUI through a 'constraints box'. In the Python code the user can include this in one of the function calls. In our optimal synthesis algorithm, at each call to the solver, the variable `COST` is replaced by a cost function. An example of a user defined cost function would be:

```
custom_cost = 3 * A  + 2 * r1
```

where `A` is a species and `r1` refers to the first reaction. This cost function would penalize the number of uses of the species `A`, and the use of reaction `r1` (which would make sense if `r1` is optional). Figure 6.5 demonstrates the lines in which the `COST` variable is replaced by the insertion of the function `get_cost`, which by default is the cost function that penalizes CRN complexity.

```
s += " @1 (and\n"
s += "\t // cost condition\n"
s += "\t( (or ( (%s) <= MAX_COST) (NO_COST_LIMIT = 1)))\n\n" % self.crn.get_cost()

def get_cost(self):
"""
Construct a SymPy expression for the topology cost in terms of the decision variables.

The cost is defined as:
 3*(total number of species that participate in a reaction) +
  6*(sum of stoichiometric coefficients of all reactants) +
  5*(sum of stoichiometric coefficients of all products) +

:return:
"""
```

Figure 6.5: In the first 3 lines, the cost constraint is outlined in the `dReach` encoding in which a string $\%s$ is replaced by the output of the function `get_cost`. The documentation summary for `get_cost` is shown in grey, and by default, implements the cost function outlined in Chapter 5, Definition 21.

Handling Inputs

A user is able to designate a species as 'input species', whose concentration follows a predetermined trajectory over time, rather than being determined by the kinetics of reactions. In the GUI this trajectory can be a sum of linear, sigmoidal (Hill) and bellshaped (Gaussian) terms. In the Python package these ODEs can take any closed form. An example of an input species in the Python package would be:

```
rate =  sympify("-1*10^20*(1/(t))^10")
input1 = InputSpecies("Input1",rate, initial_value=0)
```

where the `rate` is the ODE (as a function of time `t`), `Input1` is the input species, and an optional `initial_value` of 0 is provided. The function `sympify` takes a string and transforms it to its symbolic Python representation. Using the GUI, the user can manipulate Gaussian, linear and sigmoidal terms graphically to produce complex input functions as is demonstrated in Figure 6.6, which demonstrate the sum of two sigmoidal inputs created and manipulated by the user.

6.1.4 Encoding of the derivative

A difference in performance between the synthesis of the bellshape sketch presented in Chapter 5 Example 5.1.4 (see [13]) and the synthesis presented in the tool is the method by which the derivative was computed. The original paper relied on domain specific knowledge of the starting trajectory of the species K. With this knowledge two ODEs are defined K and Kd, where Kd was a perturbation of K such that initially K was 0.1 and Kd was 0.12. The derivative was then calculated via a check $K - Kd$ for the mode jump, essentially performing a Euler approximation of the derivative. The ODEs were then defined as:

```
(d.K/ d.time = SF*(c7*choice1*k3*SF + k1*(A*(1- lam1) +
B*lam1)*(c2 - (1-c1c))*(c1c + K*(1-c1c)) - k2*((1-c3c)*K^2 + c3c*K)*(c5c +
(1-c5c)*(A*(1- lam2) + B*lam2))*(c3c - c4 + 2*(1-c3c))));


(d.Kd / d.time = SF*(c7*choice1*k3*SF + k1*(Ad*(1- lam1) +
```

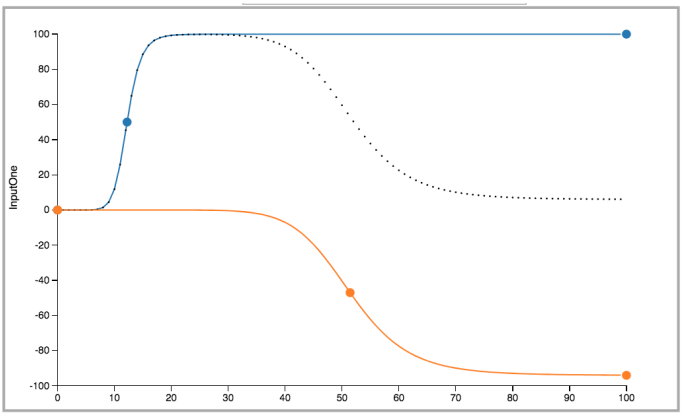


Figure 6.6: Demonstration of the possibility for a user to create custom input functions for CRNs. The user specifies and can edit a series of Gaussian or sigmoidal input functions which are summed to give an output function (seen in the dotted output line). In this case, two sigmoidal inputs are summed which produces a bellshape-like output profile.

No. of Choice Parameters	Perturbation Method	General Derivative Method
2	3m43s	4m12s
4	4m11s	6m12s
6	6m14s	11m28s
8	10m54s	17m13s

Table 6.1: A table comparing methods for computing the derivative using the *Perturbation Method*, where the derivative of our bellshape model is computed using a perturbation of the trajectory of K, is compared with the *General Derivative Method*, where a symbolic differentiation of K is calculated as a result of the chain rule. *No. of Choice Parameters* refers to the number of open discrete parameters within the bellshape sketch, with the continuous rate parameters k_1, k_2 and k_3 already open. From a quick observation we can see that the generalized derivative method is slower.

```
Bd*lam1)*(c2 - (1-c1c))*(c1c + Kd*(1-c1c)) - k2*((1-c3c)*Kd^2 + c3c*Kd)*(c5c
+ (1-c5c)*(Ad*(1- lam2) + Bd*lam2))*(c3c - c4 + 2*(1-c3c))));
```

where `Ad`,`Bd` and `Kd` are the perturbations of `A`,`B`,`K`, and `lam1`, `lam2`, `c1c`, ..., `c5c` were the choice encodings.

In the tool the derivative was calculated symbolically as a result of the chain rule. For example the derivative of K in the bellshape model is as follows:

```
(dK./ d.time = -SF*(B*K*k_2 + k_1*(c1_0 + c1_1*(A*lamA1 + B*lamB1) +
c1_2*(A*lamA1 + B*lamB1)^2)*(K^2*c2_2 + K*c2_1 + c2_0)*(c2_1 + 2*c2_2 -
c3_1 - 2*c3_2) - k_3*tc_1_1*(c7_1 + 2*c7_2)));
```

While this meant the derivatives could be generalized, and have n-th order derivatives, it also meant our synthesis method was slower. Table 6.1 presents timings on a single synthesis of our bellshape sketch in which, starting with a model with parametric rate parameters only, choice parameters (`c1`,`lam1` etc) are added. Comparatively, the derivative encoding presented in the tool is generally slower due to an increase in the complexity of the ODEs involved. It is important to note however, that this allows for reasoning over the derivatives without prior knowledge on initial conditions needed in the Euler approximation method.

It is important to note that the derivative ODEs are only computed if the user specifies the use of a derivative in the specification, which our tool is able to recognize.

6.1.5 Interpreting Results from the Solver

The `iSAT` and `dReach` tools produce results files that contain a list of the parameterized variables with boxes of width $< \delta$ at the start and end of the synthesis and at mode jumps.

To see an example of a results file, please refer to Appendix B. An example of the result for the derivative of K in our bellshape example is as follows:

```
dK_dt (float):
 @0:  [0.14999999,0.15000001] (width: 1.94289029e-16)
 @1:  [0,0.00100001) (width: 0.001)
 @2:  [2.49962499,2.50875196) (width: 0.00912695313)
```

The initial values are taken and the average of the box (the upper value − the lower value $/2$) is then substituted as the initial value into the symbolic ODEs. Originally, the value was taken and substituted into the CRN directly using the tool `VisualGEC` which plotted the simulation of the CRN. However, in the `CRNSketch` tool values were instead substituted into the symbolic ODEs directly and used the supplied Python ODE solver in order to plot these. The latter tool also provides the user with the functionality to plot the time of the mode changes.

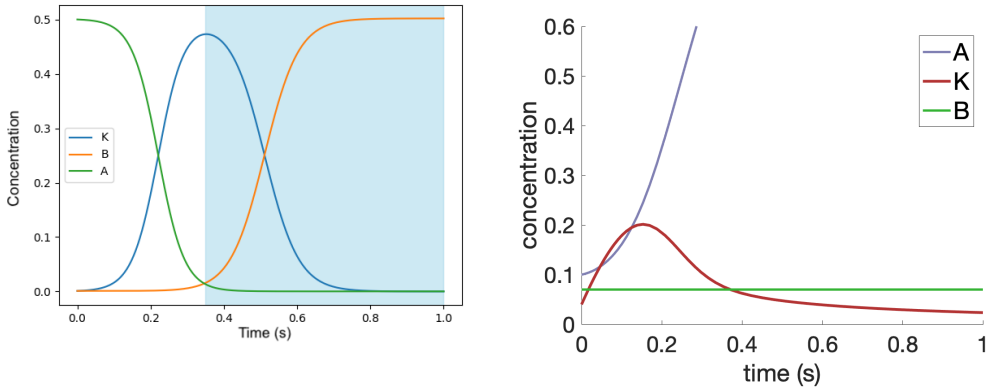
6.2 Optimal Synthesis Algorithm Evaluation

All experimental data used in the production of the summary tables presented in this section are available at the repository referenced at the beginning of the chapter. However, let us move on to evaluating the Optimal Synthesis algorithms described in Chapter 5 Section 5.5.1 on the bellshape sketch provided in Chapter 5 Example 5.1.4. Given this sketch and specification we evaluate our bellshape encoding based upon the cost function:

```
6*(c1 + cc + c3 + 2*(1-c3)) + 5*(c2 + c6 + c4 ) <= $COST$
```

where the variable `COST` is replaced by the given cost in the current iteration of the algorithm within the file *SolverCaller.py*, and the cost function is the instantiation of the default cost function provided in Chapter 5 Example 3.2.5. Firstly, visualizations of an individual synthesis of the sketch in Figure 6.7 are given. In (a) the result of the optimal synthesis with an individual cost of 25 (i.e. the choices made by the synthesizer must be less than 25 when evaluated by the cost function) is visible. In (b) a result of the optimal synthesis without a cost function can be seen, that contains a CRN with the inclusion of the optional reaction - the third reaction - which would be heavily penalized with the cost function.

Following on from this initial experiment examining individual synthesis, the scalability of the solver with respect to precision δ can be evaluated. This is done by increasing the size of the discrete search space, achieved by increasing the size of the choice variables of species and coefficient variables of the sketch. That is, starting with a concrete instantiation, the



(a) (b)

Figure 6.7: Two bellshape profiles synthesized by our tool `CRNSketch`. (a) shows the output of our tool's synthesis of the bellshape with a maximum cost of 25. The coloured regions represent the mode switches. The first bellshape has initial rates $k_1 = 56, k_2 = 43$, initial conditions $A = 0.5, B = 0.001, K = 0.001$ and reactions $\{A + K \xrightarrow{k_1} 2K, K + B \xrightarrow{k_2} 2B\}$. Note the mode switch occurs when the derivative of K is 0. The second bellshape (b) has initial rates $k_1 = 52, k_2 = 92.6, k_3 = 2.0$, initial conditions $A = 0.1, B = 0.07, K = 0.04$ and reactions $\{A + K \xrightarrow{k_1} 2K, A + K \xrightarrow{k_2} 2A, \varnothing \xrightarrow{k_3} K\}$ and was synthesized with no cost function associated. Observe that, with the inclusion of the reaction $\varnothing \xrightarrow{k_3} K$, which would be heavily penalised by the cost function due to the instantiation of an additional optional reaction, produces an alternative profile.

parameter choices are slowly opened up. Ccost constraints are excluded, since these reduce the size of the search space by removing possible discrete instantiations of the sketch. The runtimes, provided in Figure 6.8, correspond to a single call to iSAT with different δ values, leading to SAT outcomes in all cases. The time-out was set at 4 hours. On the basis of these results, users of the tool are recommended to not use a precision greater than $\delta = 10^{-}3$ when using the tool for models with large parameter space. Note that the size of the continuous state space, given by the domains of rate variables, does not impose such a large performance degradation, as shown in Table 6.9. Furthermore in several application domains discussed [57, 29, 179], the precision of rates and molecular counts do not exceed $\delta = 10^{-}3$ and therefore this is sufficient for the application domain specified in the literature review.

Let us next discuss, given calls to the solver and the evaluation of solver times, how the generalized synthesis algorithm, Algorithm 1 and the variation Algorithm 2, are used for optimal synthesis. To summarize, the first algorithm explores search strategies at a set precision with a variable cost function and the second is a variation on this. The variant algorithm also has a variable cost function, but at a given cost, attempts to perform a depth

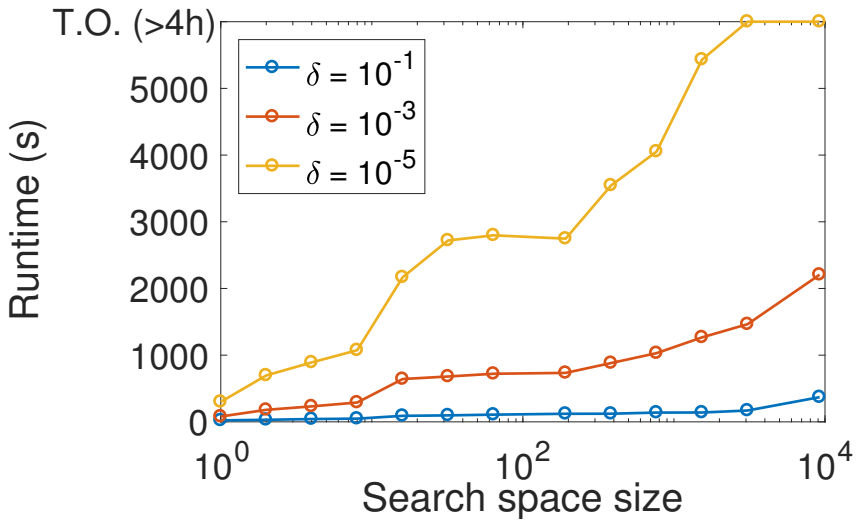


Figure 6.8: Performance evaluation of the iSAT solver with the bellshape generator model. Three studies with different precisions δ against increasing discrete search space size are given. Our time-out time was at 4 hours. The x-axis is logarithmic and the search space size is calculated by taking the cartesian product of open parameters. e.g. $[0, 2] * [0, 4] = 8$.

first search by first searching a larger parameter space at lower precision before then looking at smaller subsets of that space at higher precision. If at higher precision the result is UNSAT then the negation of that space is added to the list of parameter regions to explore. Table 6.2 compares the runtimes for both of these algorithms against the strategies discussed in Chapter 5 Section 5.5.1. The bottom-up strategy, which attempts to find the minimal cost by starting with UNSAT cases, starts with a cost of 22, the minimum cost of any configuration of discrete parameters. The top-down strategy starts with a cost of 37, which covers all possible sketches regardless of specification. The binary search starts with a minimum of 22 and a maximum of 37. The second column demonstrates the number of solver calls with UNSAT/SAT outcomes. The third and fourth column compare the runtimes for the main synthesis algorithm and its variant. All calls to the solver were made at a precision of $\delta = 10^{-3}$. Importantly, the average runtime for a single call to iSAT is significantly improved when cost constraints are used, since these reduce the discrete search space (between 216s and 802s with cost constraints, 1267s without). Moreover, results clearly indicate that UNSAT cases are considerably faster to solve, because inconsistent cost constraints typically lead to trivial UNSAT instances. This favours the bottom-up approach over the top-down. In this example, the bottom-up approach also outperforms binary search, though the opposite is expected to be true for synthesis problems with wider spectra of costs. As expected, a speed-up can be observed when using a lower precision for UNSAT witnesses, except for with the top-down approach. It can be observed therefore, that the cumulative timings of

Search Strategy	UNSAT/SAT calls	Total Time Algorithm 1 (s)	Total Time Algorithm 2 (s)
bottom-up	7/1	2671	1732
top-down	1/6	4863	5612
binary-search	2/4	3440	3121

Table 6.2: A table comparing the different search strategies outlined in Chapter 5 Section 5.5.1 against the synthesis algorithm outlined in Chapter 5 Algorithm 1 and its variant Algorithm 2.

Algorithm 2 on average do not produce substantially better results than Algorithm 1.

6.3 Tool Evaluation

6.3.1 Poisson Process

So far we have concentrated on the deterministic semantics of a CRN. However, we will now look to show that we are able to synthesize CRNs that, according to a specification that reasons over variance, demonstrates that our approach is able to synthesize a CRN that behaves as a stochastic process. This is shown with a super Poisson process which is identified as a process which has variance greater than its expectation. The behaviour on the interval $[0, 1]$ is formalized using a 1-mode specification:

```
invt: varA >= A;
post: time = 1;
```

where $varA$ is the variable that encodes variance as per Chapter 4 Definition 17. The following sketch is considered, where both reactions are mandatory, reflecting the knowledge that A is both produced and degraded.

$$\Lambda = \{A, B\}, \Lambda_o = \{B\}, \lambda_1, \lambda_2 : \Lambda,$$
$$\mathcal{R}_m = \{\tau_1, \tau_2\}, A_0 = B_0 = 0,$$
$$k_1, k_2 : [0, 100], c_1, c_2, c_3 : [0, 2]$$
$$\tau_1 :\rightarrow^{k_1} c_1 A + c_2\lambda_1; \quad \tau_2 : A \rightarrow^{k_2} c_3\lambda_2;$$

Using precision $\delta = 10^{-3}$, we obtained the optimal solution $\{\xrightarrow{23} 2A, A \xrightarrow{94}\}$ (cost 16) in 4s. Notably, the synthesis without cost constraints took 19s. Moreover, the ability to reason over the variance allows the solver to discard solution $\{\rightarrow A, A \rightarrow\}$ (implementation of a Poisson process [180]), which would have led to a variance equal to expectation. The Table (left) in Table 6.9 demonstrates the scalability of our approach with respect to the size of the continuous parameter space. Note that, in comparison to the increase in continuous space,

Rate interval	Time (s)
$[0, 1]$	4
$[0, 10]$	18
$[0, 100]$	31

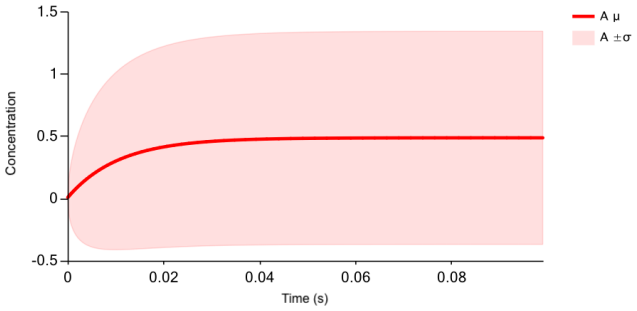


Figure 6.9: In (a), consider the impact of increasing the size of the continuous rate parameter space for CRN sketches. Using our Poisson process continuous rate parameters and record the resulting time are synthesized. In (b), the expectation and variance of the Poisson process over time is plotted. The mean for the Poisson process is depicted by the thick line and the variance indicated by the shaded region produced by our tool. The variance is larger than the expectation throughout the time evolution of the CRN plot. Note that the variance of LNA will always be non-negative, and the fact that here it is displayed to the contrary is an artifact of the tool. The CRN is represented is $\{\xrightarrow{23} 2A, A \xrightarrow{94}\}$.

the time for completion of the synthesis call remains fast. Observe in Table 6.9 (right) the synthesized process with the mean and variance plotted by our tool. Observe also that the process has variance that is larger than the expectation over time.

6.3.2 Phosphorelay Network

Up until this point, we have looked only at problems where both discrete and continuous parameters are considered. However, given the promising result for the synthesis of rate parameters shown in Figure 6.9 let us then examine systems with large open parameter spaces by now considering synthesis of rates. A rate synthesis problem (i.e. all discrete parameters are instantiated) is presented for a three-layer phosphorelay network [181]. In this network, each layer Li ($i = 1, 2, 3$) can be found in phosphorylated form, i.e. are attached to a phosphorelated group which we denote Lip. Additionally, there is a ligand B, acting as an input for the network. The authors of [181] were interested in finding rates such that the time dynamics of $L3p$ shows ultra-sensitivity which, in terms of continuous behaviour, is a sigmoid shape of the time evolution of $L3p$, i.e. the second derivative of $L3p$ is equal to zero at some point. They obtained the resulting phosphorelay by manually varying the parameters until the right profile was discovered. This section will demonstrate that, in our approach, these parameters can automatically be synthesized, thus reducing

```
def synthesize_with_dreal(crn):
    derivatives = [{"variable": 'L3p', "order": 1, "is_variance": False, "name": "L3p_dot"},
                   {"variable": 'L3p', "order": 2, "is_variance": False, "name": "L3p_dot_dot"}]

    specification_dreal = [('', '(and (L3p_dot >= 0)(L3p_dot_dot >= 0))', '(and (L3p_dot_dot = 0)(B > 5))'),
                           ('', '(and (L3p_dot >= 0)(L3p_dot_dot <= 0))', '')]
```

Figure 6.10: Specification for our phosphorelay network in which a species $L3p$ is specified to produce a sigmoidal response where the second derivative of $L3p$ is 0 at some point. First, the derivatives of $L3p$ that are required for the specification are defined before specifying that the first and second derivative of $L3p$ must be greater than 0, that the second derivative must be 0 in transition to the the second mode, and that the second derivative must be less than 0 throughout the second mode.

the time required for the task. Consider then the following sketch:

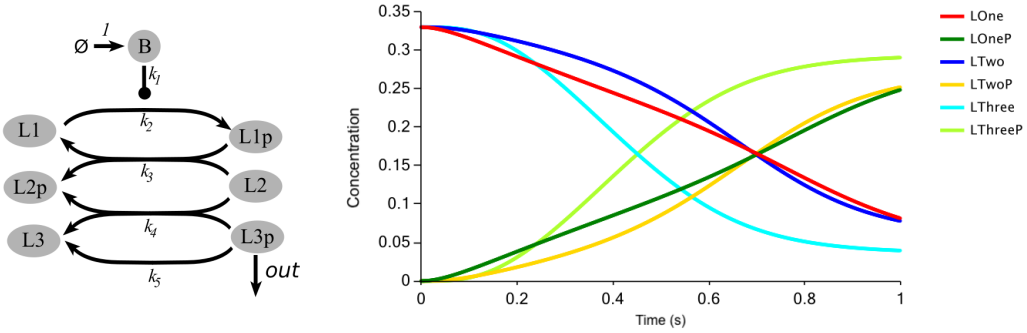
$$
\begin{aligned}
&L1 + B \xrightarrow{k_1} B + L1p \\
&L2 + L1p \xrightarrow{k_2} L1 + L2p \\
&L2p + L3 \xrightarrow{k_3} L2 + L3p \\
&L3p \xrightarrow{k_4} L3; \quad \varnothing \xrightarrow{1} B \\
&k_1, \ldots, k_4 : (0, 100] \\
&Li_0 = 330, Lip_0 = B_0 = 0
\end{aligned}
$$

where k_1, k_2, k_3 and k_4 are the parameters we wish to synthesize. As there are no discrete parameters to synthesize, a diagrammatic version of this is provided in Figure 6.11(a).

We formalize the required behaviour using a 2 mode specification. In particular, we consider a time interval $[0, 1]$ during which $L3p$ never decreases ($E^{(1)}[L3p] \geq 0$), requiring also that an inflection point in the second derivative occurs in the transition between the two phases. This specification is demonstrated in Figure 6.10.

In Figure 6.11(b) the CRN with synthesized rates at precision $\delta = 10^{-3}$ can be observed. Observe also the sigmoid profile obtained (`LThreeP`), illustrated in light green. Further consideration is given to a more complex variant of the problem, where the specification is extended to require that the variance of $L3p$ at its inflection point (the point where the variance is known to reach its maximum [181]) is limited by a threshold. This extension led to an encoding with 37 symbolic ODEs, compared to the 9 ODEs (7 species plus two ODEs for the derivatives of $L3p$) needed for the previous specification. Figure 6.12(a) demonstrates the runtimes of the synthesis process for both variants of the model and different precision δ. The results demonstrate that neither increasing the number of ODEs nor improving the precision leads to exponential slowdown of the synthesis process, thus indicating our approach is scalable. See also Figure 6.12(b), which demonstrates the results of LNA synthesis

of the phosphorelay network and shows the variant of the model producing a variance for $L3p$ at a reduced threshold.



(a) (b)

Figure 6.11: Observe in (a) the diagrammatic notation for the phosphorelay network with the parameterized rates given as k_1,k_2,k_3,k_4 and k_5. In (b), observe the synthesis of the deterministic version of the phosphorelay network. $L3P$ is given in light green and has an inflection point.

6.3.3 Toggle Switch Example

So far deterministic and LNA semantics have been considered. The tool supports several widely-used choices of deterministic kinetic laws (Mass Action, Hill and Michaelis-Menten), and allows different reactions in the same CRN Sketch to use different forms of kinetic laws. This is useful in the case that the user wishes to approximate reactions of the network with the enzyme kinetics, but then give exact network specification for other parts. In this example we consider the Hill kinetics as outlined in Chapter 4 Definition 18.

As an example, we synthesize a bistable system that can switched from an 'off' state to an 'on' state. The creation of such a toggle switch was one of the early breakthroughs in synthetic biology [68] and bistability is an important property of many natural biological systems, including the λ-phage decision circuit, the MAPK cascade, and cell cycle regulation.

This desired behaviour can be described with a specification that requires the concentration of a species designated as the output to initially be low, but to rise above a threshold within a certain time of an input being supplied, and to remain above this threshold for a certain time after the input is removed.

Consider then a genetic toggle switch consisting of two repressor-promoter pairs (*lacI* and *cIts*), and treat *cIts* protein as the output. Switching on is achieved by addition of an inducer that prevents lacI from acting as a repressor. This system is a CRN consisting of four reactions: production of *lacI* and *cIts* protein (following Hill kinetics), and degradation of

ODEs	δ	Time (s)
9	10^{-1}	53
9	10^{-3}	370
9	10^{-5}	719
37	10^{-1}	1052
37	10^{-3}	11276
37	10^{-5}	39047

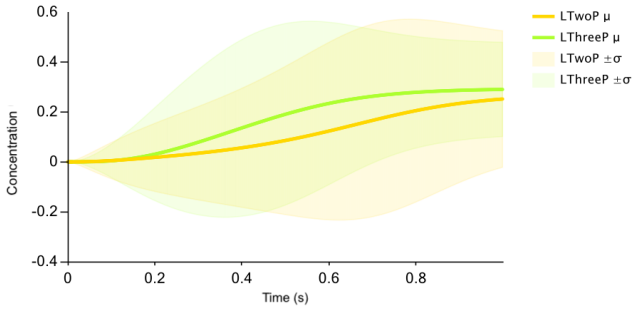
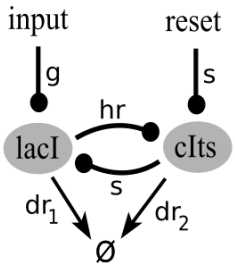


Figure 6.12: In (a) the runtimes for different precisions are provided, as well as those for the two variants of the phosphorelay we consider: one without a threshold on the variance and one with a variance threshold. In (b), oobserve the synthesized outcome with the LNA, with the variance of $L3p$ given by the light green shading. Note that the clear inflection point still exists for $L3p$, shown by the green line.

lacI and *cIts* protein (following mass action kinetics). The complete dynamics for the toggle switch given by Hill activation and degredation functions is demonstrated in Figure 6.13.



$$g = \frac{\alpha_2}{(1 + k_2 \text{ lacI } k_1 \text{ input } + k_2)^2}$$

$$hr = \frac{1}{(1 + \alpha_1)^2}$$

$$k_1, k_2, dr1, dr2 \in [0, 2]$$

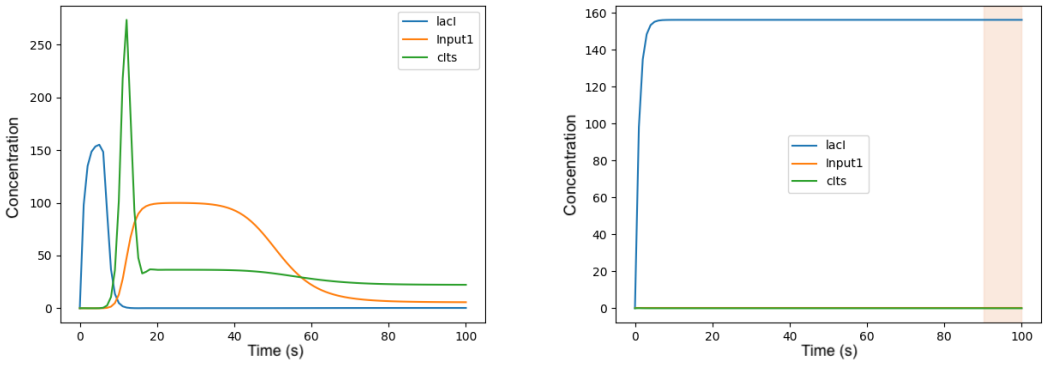
$$\frac{dInput}{dt} = -356 * e^{(-178*(-0.5+t)^2)*(-0.5+t))}$$

Figure 6.13: The toggle switch example demonstrating use of Michaelis-Menten Kinetics where the values $\alpha_1, \alpha_2, k_1, k_2, dr_1$ and dr_2 are to be synthesized. The rates g and hr are the Hill rate parameters whilst dr_1 and dr_2 obey mass-action laws. The rate s acts as a reset for the toggle switch, but is not used in this example. $CIts$ is considered the output.

Parameter bounds were defined via the *in silico* data given for this experiment [182]. Given a Gaussian input, we synthesize parameters such that the switch turns on and remains on (seen in the output species *cIts*) even after the input concentration has fallen to near zero (Figure 6.14a). For comparison, a case where no input is supplied has been provided in Figure 6.14b. Here *cIts* remains below the threshold. The total synthesis time was 28 minutes for 5 open parameters at a precision of $\delta = 0.01$.

6.3.4 Synthesis of Qualitative Responses to Inputs

So far, for large systems, only cases with at most 11 open continuous parameters have been considered - as in the case of our phosphorelay under LNA semantics example. However, in



(a) (b)

Figure 6.14: Synthesizing a bistable circuit. In (a) we see an example output where a condition for bistability is specified: $cIts$ must rise above a threshold, in this case $Th > 10$ and remain there until $t = 100$. Observe that there is an initial spike but that the concentration does in fact remain above the threshold for the total duration. In (b), the input is changed to 0. Here we see that $cIts$ is no longer repressed by $lacI$ and rises naturally.

a final example to follow, an attempt is made to synthesize a system with 24 rate parameters. A large number of open parameters allows flexibility in the output that is synthesized. In this example, an attempt is made to synthesize CRNs which produce an output profile with a particular qualitative shape in response to a bellshaped input.

A family of systems consisting of three proteins is considered, each of which can exist in an inactive (P_i) or active (P_i^*) form [183]. Conversion between the active and inactive forms can occur spontaneously, or be catalysed by particular active proteins. Additionally, the concentration of input species affects the rate of activation of the first protein. The complex reaction kinetics modelled in the original paper can be broken down into simpler mass-action kinetics which are represented by the 24 reactions between six species.

This given CRN is captured with the following reaction network that includes six parametric species ($P_1, P_2, P_3, P_1^*, P_2^*$ and P_3^*) $\in [0, 2]$ with parametric rates $k_1 \ldots k_{24} \in [0, 1]$ belonging to 24 reactions. An additional input reaction $I \rightarrow P_1$ is included, as being produced from the input I by a reaction with rate k_{inpt}. The equation for the input species is given by a user-defined bellshape input with equation $e^{-(2/3*t-1/3)^2}$ derived from the general equation for Gaussian input. Given this input equation, rates, and species the CRN is defined as follows:

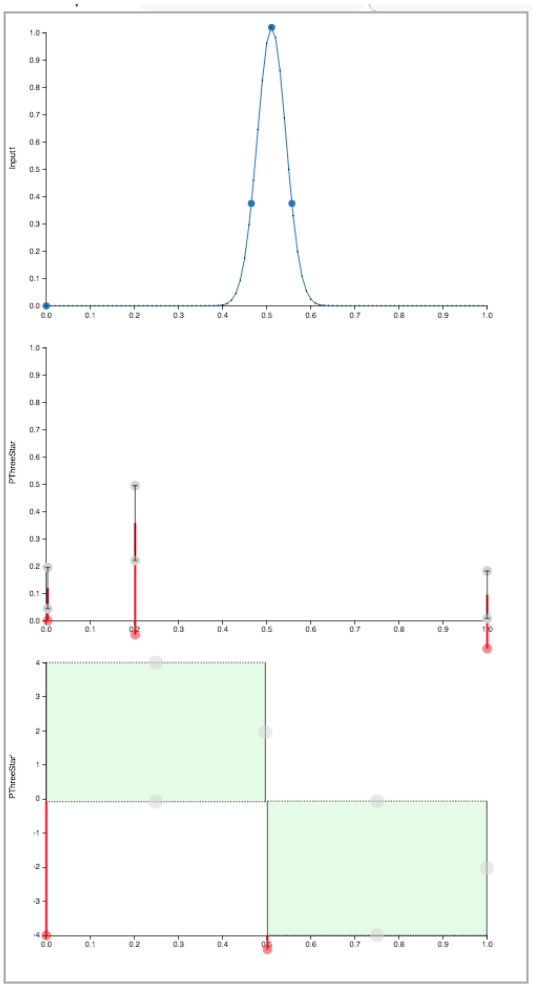
$$
\begin{array}{rrr}
P_1 \xrightarrow{k_1} P_1^* & P_2 \xrightarrow{k_9} P_2^* & P_3 \xrightarrow{k_{17}} P_3^* \\
P_1 + P_1^* \xrightarrow{k_2} P_1^* + P_1^* & P_2 + P_2^* \xrightarrow{k_{10}} P_2^* + P_2^* & P_3 + P_3^* \xrightarrow{k_{18}} P_3^* + P_3^* \\
P_1 + P_2^* \xrightarrow{k_3} P_1^* + P_2^* & P_2 + P_3^* \xrightarrow{k_{11}} P_2^* + P_1^* & P_3 + P_2^* \xrightarrow{k_{19}} P_3^* + P_1^* \\
P_1 + P_3^* \xrightarrow{k_4} P_1^* + P_3^* & P_2 + P_1^* \xrightarrow{k_{12}} P_2^* + P_3^* & P_3 + P_1^* \xrightarrow{k_{20}} P_3^* + P_2^* \\
P_1^* \xrightarrow{k_5} P_1 & P_2^* \xrightarrow{k_{13}} P_2 & P_3^* \xrightarrow{k_{21}} P_3 \\
P_1^* + P_1^* \xrightarrow{k_6} P_1 + P_1^* & P_2^* + P_2^* \xrightarrow{k_{14}} P_2 + P_2^* & P_3^* + P_3^* \xrightarrow{k_{22}} P_3 + P_3^* \\
P_1^* + P_2^* \xrightarrow{k_7} P_1 + P_2^* & P_2^* + P_3^* \xrightarrow{k_{15}} P_2 + P_3^* & P_3^* + P_2^* \xrightarrow{k_{23}} P_3 + P_2^* \\
P_1^* + P_3^* \xrightarrow{k_8} P_1 + P_3^* & P_2^* + P_1^* \xrightarrow{k_{16}} P_2 + P_1^* & P_3^* + P_1^* \xrightarrow{k_{24}} P_3 + P_1^*
\end{array}
$$

The network topology determines which proteins are able to catalyse the activation and inactivation of which other proteins, and hence which dynamics are possible for the system.

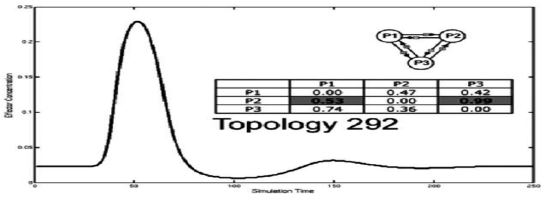
As part of a validation that our network and synthesis of parameters matches that of [183], observe demonstrated in Figure 6.15 a quantitative comparison of the user-specified synthesis of a Gaussian output compared to one given in [183]. The specification for our model is given as a `TimeRails` specification in Figure 6.15a and equates to the MTL specification:

$$t = 0 : \frac{dP3*}{dt} >= 0, P3* < 0.2; t < 1 : \frac{dP3*}{dt} = 0, P3* > 0.3; t = 1 : \frac{dP3*}{dt} < 0, P3* < 0.2.$$

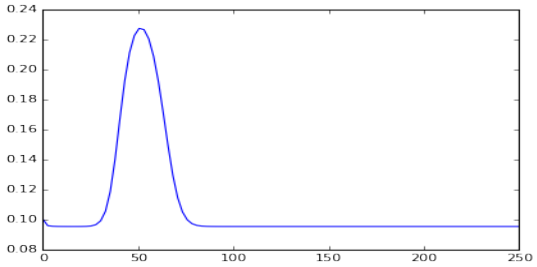
The authors of [183] not only identified circuits that displays a response that was bell-shaped, but also responses that were oscillatory, derivative, sigmoidal and linear. We are able to directly synthesize a CRN belonging to any specific one of these classes at far lower computational cost, whilst also jointly searching over parameter values. Another advantage over [13], is that we are able to automatically correctly classify each output as it is satisfiable for a given specification, whereas the original outputs had to be classified by hand. We have been able to synthesize all of the outcomes given in the original work, as demonstrated in Figure 6.16. Observe that in each case we are able to synthesize the corresponding behaviour. Finally, with each of these synthesized profiles, a comparison of scalability is given in Table 6.3. Each result was synthesized at a precision $\delta = 10^{-3}$. These experiments found that, as the complexity of the specification increased in the number of modes, so too did the time it took to synthesize a solution. It can be observed that, with 16 fixed rates and 8 open rate parameters chosen at random from the rates $k_1, \ldots, k_24$, the time for synthesis becomes quickly intractable. Note that a cost function was not employed here as different qualitative outputs were desired.



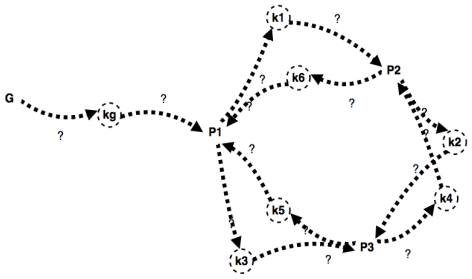
(a) TimeRails specification, including input



(b) Original Gaussian Example



(c) Comparative Gaussian Example



(d)

Figure 6.15: Synthesizing one of the specified outcomes for the 24 reaction network example. We capture the complete process of reproducing and synthesizing a bellshape behaviour, comparable to the result in (b) which is taken from [183]. A qualitatively similar result is captured by the specification in (a), namely that the output (c) in purple must start below a value of 0.2 at $t = 0$ and rise above a value of 0.4 before diminishing below 0.2 before $t = 10$. In (a) the top graphic represents the user-defined input specification, the second panel represents the specification for $P3*$ (mainly it starts at a low value, grows and finally diminishes again) and the bottom panel represents the derivative of $P3*$, the derivative increases and then decreases. A graphical demonstration of the 24 reaction network is shown in the GUI in (d).

Conclusion

Scalability of the Synthesis Method Despite performance gains on notable previous work, the `CRNSketch` tool still suffers from scalability issues. Both the increase in the number of modes (complexity of the specification) and the size of the parameter space have substantial impact on its performance. However, interestingly, discrete parameters were seen to have

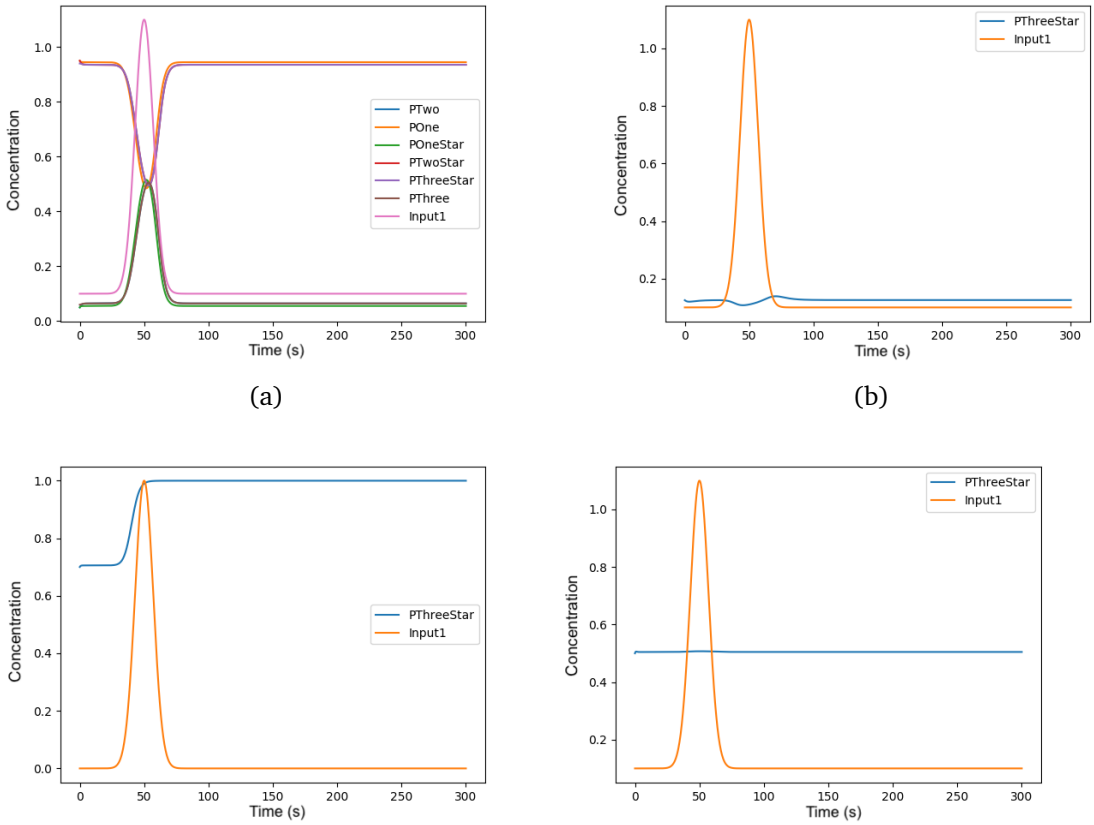


Figure 6.16: Four other outcomes from the complete synthesis of the 24-Reaction Network. In (a), observe the result of the negative inflection bellshape, with input species $Input_1$ in pink and output $P3*$ in purple. This demonstrates the complete dynamics of the ODE system represented by all of the species in the network. In (b), the visible system is reduced to just input and output for clarity, and we try to synthesize the derivative of the input. Given in (c), is an example output where a condition for bistability is specified: $P3*$ must rise above a threshold $Th > 0.9$, and we correctly synthesize switch behaviour. In (d), we demonstrate the synthesis of a constant output based upon a bellshape input. The network and specifications can be found in the examples folder of the tool for further exploration.

a much smaller impact on performance, allowing users to keep stoichometric constraints relatively open. As outlined in Section 6.3.4, despite the success of our synthesis method when synthesizing up to 8 open parameters — covering the majority of specifications — we were unable to synthesize 24 open parameters. There are, therefore, still clear limitations to the synthesis capabilities of this method. The number of modes drastically decreases performance, as the flow of a system is forced to satisfy a more complex specification.

In generalizing our framework into a tool a drop in performance was observed, due to the fact an exact symbolic representation of the derivative had to be used, as seen in Table 6.1. A related problem is that, in general, it is impossible to know at what time the system will achieve a steady-state and therefore an arbitrary time horizon is usually given. This poses the problem that large time horizons take a long time to compute. Users are warned of the long runtimes of SMT-solvers on non-linear parametric ODEs on the tools website.

Using the `iSAT-ODE` and `dReach` tools Both the tool `iSAT-ODE` and `dReach` are still in active development. In general the solvers are being improved constantly and dReach received a major performance upgrade during the duration of the PhD. As `iSAT` and `dReach` continue to improve, improvements in their stability and performance can be transferred to `CRNSketch`.

Multiple Candidate Solutions One of the disadvantages of using SMT-ODE solvers is that they only produce candidate solution boxes of width δ. This implies that there is only one solution provided to the user. When replicating results, if there are multiple candidate

	Open $[0, 1]$ Parameters	No. of Modes	Time (m)
Constant	1	2	2
	4	2	15
	8	2	180
Switch	1	3	4
	4	3	24
	8	3	220
Inverted-BellShape	1	4	12
	4	4	45
	8	4	5hrs+
Derivative	1	5	15
	4	5	230
	8	5	5hrs+

Table 6.3: A table outlining the time for synthesis of experimental results from the 24-reaction network example. Open parameters refers to continuous rate parameters in the interval $[0, 1]$ excluding any discrete choices over the topology of the network. The number of modes refers to the complexity of the specification in that an increase in the number of the modes means a more complex specification (and behaviour). The examples are listed here in order of complexity of specification. The time for synthesis is given in minutes.

solutions, different solutions can be given by the solver, seemingly at random dependent upon solver techniques and parameters used. A feature that would therefore be useful to experimenters, would be the ability to have parameter regions for solutions. This would give the user a notion of robustness of the candidate solution, which would in turn improve the likelihood of a successful experiment. It would also provide the option for multiple candidate solutions which may give insights into the accuracy of the specification.

Conclusion Parameter synthesis of CRNs is an active area of research and therefore our presented tool, `CRNSketch`, would be of interest to academics in the field. We developed a tool based upon a novel method for SMT-based optimal synthesis of stochastic CRNs from well-defined temporal specifications and sketches, the theory that is defined in the previous chapter. It provides the user with a method for synthesizing challenging systems and reasoning over partially known networks. Our tool provides a scalable method for parametric design automation for provably-correct molecular devices. Several biologically relevant and well-motivated case studies have been provided which back up this claim. For future work we wish to improve this framework and apply local search methods in order to speed up the synthesis process.

CHAPTER 7

CRN Control

Contents

Control theory is a catch-all term that embodies design strategies aimed at improving the stability, robustness, and performance of physical systems in a number of applications from power networks, space systems, and chemical processes [141]. Negative feedback occurs when some function of an output of a system, process or mechanism is fed back in a manner that tends to reduce error or fluctuations in the output. A common real world example in the body is homeostasis. Perhaps the most common type of negative feedback control is that of Proportional-Integral-Derivative (PID) control. PID control has widespread application from altitude monitoring in planes, to temperature control in spacecraft. Whilst another controller, the Proportional-Integral (PI) controller, has been implemented within CRNs [154], there is yet to be a full CRN implementation of a PID controller. We provide a background for PID control in Chapter 3 Section 3.4.

Originally, the work on control came from addressing problems of tractability within the parameter synthesis method. While the reason for transitioning away from parameter synthesis to control might not at first appear obvious, there are several notable analogues between the two methods. The reference signal can be viewed as an alternative to expressing a continuous profile (as opposed to a specification), and the control of a plant can be viewed as a way of forcing the system to track a reference signal; similar to parameter synthesis of rates to match a specification. The difference however, comes from the type

of system or plant in question. Parameter synthesis addresses the domain of open systems, where rates and species can be modified. Control addresses closed systems, in which the system can only be modified or influenced externally i.e. rates and species are pre-defined.

This chapter presents and expands upon our work presented in [30]. First a derivative component as a CRN is defined which is needed for PID control. Section 7.1 explains the derivative CRN, showing examples and proving correctness. More than just a CRN for PID control, this is the first CRN under mass-action kinetics that outputs the derivative of an input concentration. It is then explored how this derivative component can be used in a wider PID context in Section 7.2. Defined are both a proportional and integral component and also proof of their correctness are given. Finally, in Section 7.3, the PID controller is evaluated by comparing and contrasting it to a PI controller on biologically motivated examples in gene expression. A discussion is provided on how the PID controller may address some of the shortcomings in our synthesis method when we wish to drive a system, with unknown or incomplete information, to a desired output behaviour. The full CRN PID controller is listed in Appendix C.

7.1 Derivative CRN

Before discussing a full controller, this section discusses building a CRN that computes the derivative of a function, i.e. given a function as input, a CRN outputs a derivative of that function. The derivative block computes a multiple of the derivative of the input signal where the multiple is a constant factor. The derivative component is used in control systems to predict the future error given its current trend, and thus to help dampen oscillations introduced by P and I components. For more information see Background Section 3.4. In [156] a biologically inspired implementation of a system that computes the derivative of an input is provided, however it is presented as non-mass action Hill functions.

Building a derivative module by chemical reactions is challenging because on-the-fly differentiation can only be done by comparing a signal at two time points, thus inherently requiring an approximation dependent on the time difference. The original idea for the derivative component involved trying to create a system where an output could track the input with a delay. Different rates were tested to find ones that successfully created a system where the output approximated the derivative of the input. This was then extended to dual rail. This results in the circuit below in Definition 24, which handles dual-rail input and output. Throughout this chapter we use the deterministic semantics defined in Chapter 4 Definition 15. For shorthand, a species A is denoted by $x_A(t)$, or x_A when the time dependence is clear from the context, the concentration of A at time t. It is important to recall the definition from Chapter 4 Definition 14, that describes how a species A can be

encoded as a dual-rail equivalent A^+ and A^-. This allows us to represent negative values since species concentrations are strictly positive.

Definition 24. Derivative Component We define a CRN derivative component as follows. For input species E^+, E^-, auxiliary species A^+, A^-, output species D^+, D^-, and rate parameters $q, s, v \in \mathbf{R}_{>0}$ and the multiplier $r \in \mathbf{R}_{\geq 0}$, the derivative block is a CRN composed by the following reactions (note we define this diagrammatically in Figure 7.1a):

$$
\begin{aligned}
E^+ &\xrightarrow{rv} E^+ + A^+ & E^- &\xrightarrow{rv} E^- + A^- \\
E^+ &\xrightarrow{rvs} E^+ + D^+ & E^- &\xrightarrow{rvs} E^- + D^- \\
A^+ &\xrightarrow{v} \varnothing & A^- &\xrightarrow{v} \varnothing \\
A^+ &\xrightarrow{vs} A^+ + D^- & A^- &\xrightarrow{vs} A^- + D^+ \\
D^+ &\xrightarrow{s} \varnothing & D^- &\xrightarrow{s} \varnothing & D^+ + D^- &\xrightarrow{q} \varnothing
\end{aligned}
$$

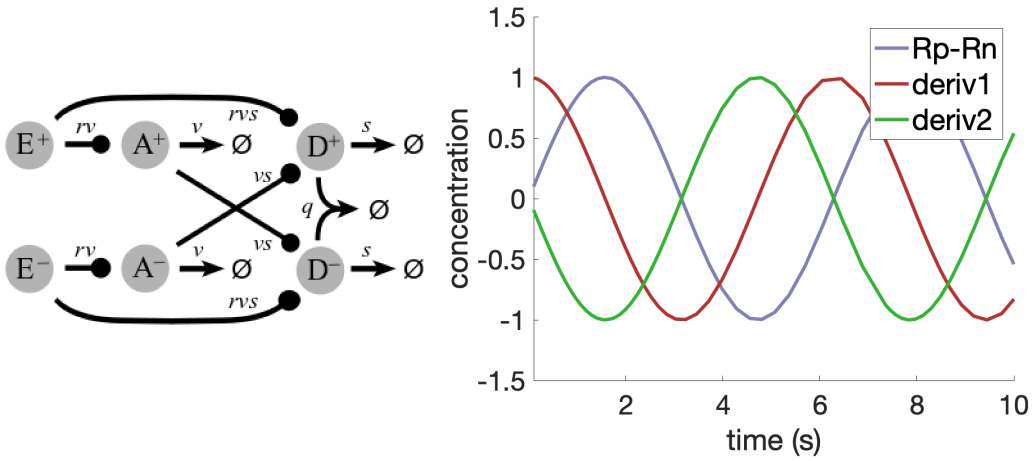
The inputs E^+ and E^- are sampled at two time points, E^+, A^+ and E^-, A^-, respectively, and a multiple of their difference is provided via D^+, D^-. The two reactions $E^+ \xrightarrow{rv} E^+ + A^+$ and $A^+ \xrightarrow{v} \varnothing$ cause x_{A^+} to track rx_{E^+} with a (slight) delay dependent on v. Intuitively, this yields $rx_{E^+} - x_{A^-} \approx \partial_t r x_{E^+}$. The two symmetric reactions similarly cause x_{A^-} to track rx_{E^-}, which ensures that $rx_{E^-} - x_{A^-} \approx \partial_t r x_{E^-}$. The three reactions $E^+ \xrightarrow{rvs} E^+ + D^+$, $A^- \xrightarrow{vs} A^- + D^+$, and $D^+ \xrightarrow{s} \varnothing$ cause x_{D^+} to track $rvx_{E^+} + vx_{A^-}$ with delay dependent on s. The three symmetric reactions similarly cause x_{D^-} to track $rvx_{E^-} + vx_{A^+}$. Thus $x_{D^+} - x_{D^-}$ tracks $v(rx_{E^+} + x_{A^-}) - v(rx_{E^-} + x_{A^+}) = v(rx_{E^+} - x_{A^+}) - v(rx_{E^-} - x_{A^-})$ with delay dependent on s. This and the above discussion allow us then to conclude that $x_{D^+} - x_{D^-} \approx r\partial_t(x_{E^+} - x_{E^-})$.

Testing the Derivative Component

We wish to test that our derivative component does indeed compute the derivative of an input species. For an example CRN, on which our derivative component will be tested, we therefore introduce a sine component. For species A^+, A^-, and output species R^+, R^-, parameter $q \in \mathbf{R}_{>0}$ the sine component is a CRN composed by the following reactions:

$$
\begin{aligned}
A^+ &\xrightarrow{q} R^+ + A^+ & A^- &\xrightarrow{q} R^- + A^- \\
R^+ &\xrightarrow{q} R^+ + A^- & R^- &\xrightarrow{q} R^- + A^+ \\
A^+ + A^- &\xrightarrow{q} \varnothing & R^+ + R^- &\xrightarrow{q} \varnothing
\end{aligned}
$$

There is a cyclical catalysis between the species A^+, R^+, A^-, R^-, so, provided the rate q is constant throughout all reactions, the CRN creates a cyclical time evolution. This is demonstrated in Figure 7.2 in which we simulate our sine component over time. The output is given by the difference in species $R^+ - R^-$, hence the negative value representation.

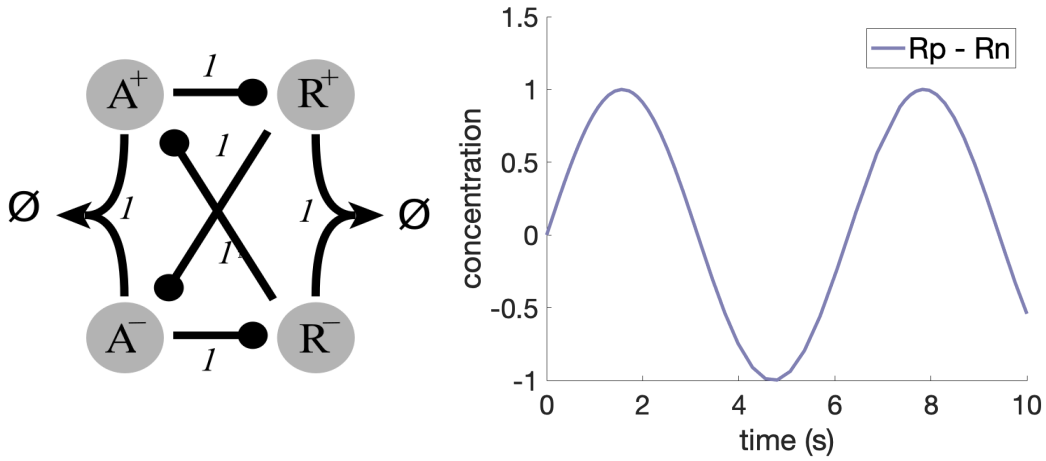


(a) (b)

Figure 7.1: We test our derivative component by tracking the derivative of a sine wave. The derivative component in our diagrammatic CRN notation is seen in (a). In (b), the output of the derivative component is compared against the sine wave in Figure 7.2b. At slower rates the derivative component is slow to track the derivative so we place two of these components in conjunction.

Using the sine reference as the input, it is demonstrated that the derivative component correctly outputs a cosine wave form as seen in Figure 7.1. We take the dual-rail output of the sine component and apply its output as a reference to the input of the derivative component. Note that two derivative components are used in order to correctly compute the cosine. This is because the rates of the derivative component are not near infinite and therefore cannot perfectly compute the derivative. Through initial testing then we have shown that the derivative component appears to operate correctly, however, for our purposes we wish to also provide more formal proof of its correctness.

Definition 25. Tikhonov's Theorem applied to CRNs We wish to show that all of our CRNs within the PID controller, including the derivative component, correctly compute a function not just for steady-state values where $t \to \infty$ but for all time t. We can leverage Tikhonov's theorem to show this. In applied mathematics, Tikhonov's theorem on dynamical systems is a result on stability of solutions of systems of differential equations. It has wide ranging use in chemical kinetic proofs [184]. We show the asymptotic correctness for all of the CRNs in this chapter by applying Tikhonov's theorem [184, Section 8.2] to the ODE system underlying the CRN. Essentially, this amounts to conducting a *rigorous* quasi steady-state approximation [185] where ODE variables are partitioned into *fast* and *slow* ones. With the intuition being that fast variables attain their equilibria almost instantaneously compared to slow variables, the approximation suggests to drop the ODEs of fast variables and to replace



(a) (b)

Figure 7.2: We introduce a CRN that acts as a non-stationary reference signal within our PID framework. In (a), we see the diagrammatic notation for this CRN. In (b), the output is seen as the difference of the two dual-rail output species $R^+ - R^-$.

the fast variables by their equilibrium values (which, in general, depend on the values of slow variables). When applying Tikhonov's theorem, it is convenient to call the ODEs of fast and slow variables as the fast and slow ODE system, respectively. In our proofs, the main objective is to identify the slow and fast ODE systems and to show that the requirements of Tikhonov's theorem are satisfied.

Assume we have a set of slow variables $\mathbf{y}$ and a set of fast variables $\mathbf{z}$, we have two separate parts of an ODE system $G(\mathbf{y}, \mathbf{z})$ and $H(\mathbf{y}, \mathbf{z})$ representing the quasi-equilibrium, and a reaction rate r. We further require that:

- Any drifts of $\mathbf{y}$ and $\mathbf{z}$ are smooth. For CRNs this is trivially true since the ODE representation of CRNs are continuous variables.
- A unique solution $\mathbf{y}^r(t), \mathbf{z}^\mathbf{r}$ of the initial value problem exists. All of our CRNs under deterministic semantics are represented by ODEs, which are polynomial and also Lipschitz continuous, and therefore have a unique solution in the fast variables when $r \to \infty$.
- A unique solution $\bar{\mathbf{y}}(t), \bar{\mathbf{z}}(t)$ of the reduced initial value problem exists. The reduced problem is defined by:

$$\frac{d\mathbf{y}}{dt} = G(\mathbf{y}, \mathbf{z}),\ \mathbf{y}(0) = \mathbf{y}_0, \tag{7.1}$$

  This states that we have a reduced system where we only consider the fast variables and that this must be differentiable. Since the ODEs within this chapter are polynomial, they are infinitely differentiable.

- Equation $0 = H(\mathbf{y}, \mathbf{z})$ is solved by $\mathbf{z} = \phi(\mathbf{y})$ where ϕ is continuous, and it is an isolated root. This states that there is a unique steady state solution for H, which is assumed to be true for a polynomial.
- $\mathbf{z} = \phi(\mathbf{y})$ is an asymptotically stable solution of $d\mathbf{z}/dt = H(\mathbf{y}, \mathbf{z})$ uniformly in $\mathbf{y}(t)$, considered as a fixed parameter. We assume the solution is stable — this is a common assumption in CRNs that is reaffirmed with the simulations of the derivative component.
- $\mathbf{z}(0)$ is contained in an interior subset of the domain of attraction of $\mathbf{z} = \phi(\mathbf{y})$ for $\mathbf{y} = \mathbf{y}(0)$.

Tikhonov's theorem states the following. Under the above conditions, for all $t < \infty$

$$\lim_{r \to 0} \mathbf{y}^r(t) = \bar{\mathbf{y}}(t), \lim_{r \to 0} \mathbf{z}^r(t) = \bar{\mathbf{z}}(t), 0 < t < \infty, \tag{7.2}$$

which, given its reference to the derivative, is incredibly useful for a proof of correctness. We go on to demonstrate how this theorem can be leveraged to prove our components operate correctly. As our proof of correctness relies on a reference to the full system, we expand upon the above later and now provide a stricter formalism for PID control of CRNs.

7.2 PID Control of CRNs

We introduce the remaining implementations of the proportional and integral components in the context of a PID controller. We describe them as blocks where the input species are $E^{\pm}$, which indicate the dual-rail error signal between the species representing the set-point and the plant output. The output of the PID controller is denoted by $U^{\pm}$ (we have followed the notation defined in Chapter 3 Definition 11). The proportional and integral components have already been introduced for linear control systems [154]. Here we prove their correctness in the presence of non-linearity, as CRNs tend to produce non-linearities. We provide additional information on how we might leverage Tikhonov's theorem, defined in Definition 25, in order to prove correctness of the derivative block.

Figure 7.3, as in a classic feedback loop, illustrates the signals synthesized by the PID controller acting on the CRN plant, whose output is measured and sent back as input of the PID controller after comparison with the reference signal. The output of the plant is always given by a species Y. The objective of the controller is to have Y follow the reference signal. Within CRN plants, any of the species (alternatively, state variables) must be non-negative, hence our use of the dual-rail notation whereby a signal can be negative as the difference of two non-negative signals [186]. A CRN component for the subtraction of two signals

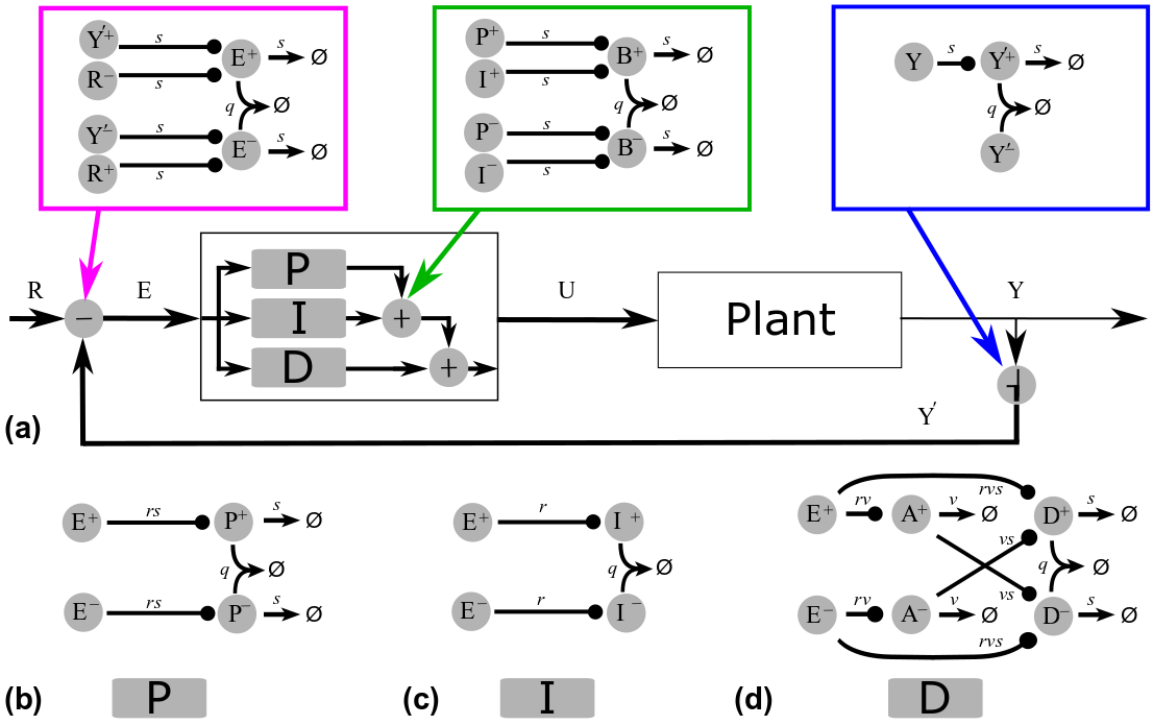


Figure 7.3: We present our feedback loop which broadly follows the notation defined in Chapter 3 Section 11. (a) takes a dual-rail reference signal R (such as our previously defined sine component) and, along with the feedback Y^I, produces an error E (computed $Y^I - R$). Signal E is obtained by the subtraction block (in magenta). Thick arrows imply dual-rail whereas the thin arrow Y implies single rail. E is fed to the controller, the chemical composition of which is described in (b),(c),(d). The proportional and integral blocks (b),(c) are taken from [154]. The proportional block adjusts the input x_A as rx_A for some multiplier $r \geq 0$. The integral block takes an input x_A and produces $r\int_0^t x_A(\tau)d\tau$ for some multiplier $r \geq 0$. The derivative block (d) takes an input x_A and produces the output $r\partial_t x_A$, where $r \geq 0$ is a multiplier (as per our definition of Tikhonov's theorem). The PID blocks are summed by the addition block (in green), yielding a control signal U which guides the plant by the CRN encoding presented in Section 7.2. As a result, the plant produces a signal Y which is converted to a dual-rail signal Y^I by the dual-rail converter block (in blue). The presence of a multiplier in each block allows the weights of each block to be adjusted. We define in Chapter 3 Section 3.4 how these weights can be adjusted.

was first presented, for linear systems, in [154], but we repeat its definition in Figure 7.3 in magenta.

Formal Definition of PID controller We now wish to consider the construction of the whole PID controller as the sum of its individual components. To this end, we let $(\Lambda_\Sigma, \mathcal{R}_\Sigma)$ denote the mass-action CRN representing the plant and construct a CRN encoding of a PID feedback controller as indicated in Figure 7.3. Given that output and control signals are vectors, the blocks of Figure 7.3 are multidimensional components. In particular, assuming that $n \geq 1$ denotes the dimension of the output and control vector, the CRN encoding of the

PID feedback law is given by interconnected:

- subtraction blocks $(\Lambda_i^S, \mathcal{R}_i^S)_{1\le i\le n}$,
- addition blocks $(\Lambda_i^A, \mathcal{R}_i^A)_{1\le i\le n}$,
- proportional blocks $(\Lambda_i^P, \mathcal{R}_i^P)_{1\le i\le n}$,
- integral blocks $(\Lambda_i^I, \mathcal{R}_i^I)_{1\le i\le n}$,
- derivative blocks $(\Lambda_i^D, \mathcal{R}_i^D)_{1\le i\le n}$,
- dual-rail converter blocks $(\Lambda_i^C, \mathcal{R}_i^C)_{1\le i\le n}$.

With this, the overall CRN is given by:

$$(\Lambda_\Sigma, \mathcal{R}_\Sigma) \cup (\Lambda_F, \mathcal{R}_F), \tag{7.3}$$

where the feedback controller CRN is defined by:

$$(\Lambda_F, \mathcal{R}_F) = \bigcup_{i=1}^{n} \bigcup_{X\in\mathcal{X}} (\Lambda_i^X, \mathcal{R}_i^X), \qquad \mathcal{X} = \{S, A, P, I, D, C\}.$$

Now that we have formally constructed the PID controller, we address the correctness of each individual block type. While the rates discussed within the controller are arbitrary, exact initial molecular counts and rates used in the experiments are discussed in Appendix C.

7.2.1 Proportional, Addition, Subtraction and Dual Rail Converter Blocks

The following PI components of our PID are given in [154], but we list the definitions for completeness. This section expands on Klavin's result for PI controllers by showing completeness as the original proofs were just applied to linear systems.

We begin by presenting the proportional block which computes an output signal that is proportional to the input signal. This component is used in control systems to react proportionally to the current error with respect to the reference signal, but can never solely converge upon a reference signal. Proportional blocks produce linear combinations of their inputs. They are used for combining signals from the integral and differentiator within our PID. A single-input block takes an input signal $A(t)$ and produces an output signal $B(t) = kA(t)$, where $k \in \mathbf{R}$ is the gain. A summation block takes input signals $\{A_i(t)\}_{i=1}^n$ and produces an output $B(t) = \sum_{i=1}^n A_i(t)$. The following CRN implements both of these functions for the dual-rail representation of signals.

Definition 26. Proportional Block [154]

For input species E^+, E^-, output species P^+, P^-, parameters $s, q \in \mathbf{R}_{>0}$, and the multiplier $r \in \mathbf{R}_{\geq 0}$, the *proportional block* is a CRN composed by the following reactions:

$$E^+ \xrightarrow{rs} E^+ + P^+ \qquad E^- \xrightarrow{rs} E^- + P^-$$
$$P^+ + P^- \xrightarrow{q} \varnothing \qquad P^+ \xrightarrow{s} \varnothing$$
$$P^- \xrightarrow{s} \varnothing$$

Theorem 1. On any bounded time interval, the solution of the ODE system induced by the CRN (7.3) converges, as $s \to \infty$ in all proportional blocks, to a solution satisfying $x_{P_i^+} = r x_{E_i^+}$ and $x_{P_i^-} = r x_{E_i^-}$ for all $1 \leq i \leq n$.

Proof (Sketch). Note that:

$$\partial_t x_{P_i^+} = rs x_{E_i^+} - s x_{P_i^+} - q x_{P r_i^+} x_{P_i^-},$$
$$\partial_t x_{P_i^-} = rs x_{E_i^-} - s x_{P_i^-} - q x_{P_i^+} x_{P_i^-}.$$

This motivates the interpretation of all $x_{P_i^+}$ and $x_{P_i^-}$ as fast variables in the sense of the quasi-equilibrium decomposition mentioned in our description of Tikhonov's theorem in Definition 25. For further information, see [184, Section 8.2]. To see that the requirements of the theorem are satisfied, we note that the fast ODE system (i.e. the set consisting of fast variables) admits, for any fixed vector of slow variables (i.e. all variables that are not fast), exactly one possible equilibrium point. Moreover, it is an asymptotically stable equilibrium of the fast ODE system. We finish the proof by noting that smooth exogenous reference signals can be captured because Tikhonov's theorem applies to non-autonomous smooth ODE systems. □

Remark 1. Theorem 1 extends the result of [154] to nonlinear control systems. The same holds true for the other blocks of this section.

Remark 2. The proof of Theorem 1 reveals that reaction $P^+ + P^- \xrightarrow{q} \varnothing$ is not strictly needed to ensure correctness. However, if we assume $t \to \infty$ the values of P^+ and P^- would also tend to ∞, so it makes sense to keep the values under a certain threshold.

The correctness of the subtraction and converter blocks introduced next and depicted in Figure 7.3 is shown similarly to Theorem 1 (i.e. they all leverage Tikhonov's Theorem). More specifically, the addition block is given by:

Definition 27. Addition block [154] For input species P^+, I^+ and P^-, I^-, output species B^+, B^-, and parameters $s, q \in \mathbf{R}_{>0}$ the *addition block* is a CRN composed by the following reactions

$$\begin{aligned} P^+ &\xrightarrow{s} P^+ + B^+ & I^+ &\xrightarrow{s} I^+ + B^+ \\ P^- &\xrightarrow{s} P^- + B^- & I^- &\xrightarrow{s} I^- + B^- \\ B^+ + B^- &\xrightarrow{q} \varnothing & B^+ &\xrightarrow{s} \varnothing & B^- &\xrightarrow{s} \varnothing \end{aligned}$$

The subtraction block, instead, is defined as follows.

Definition 28. Subtraction block [154] For input species Y'^+, R^+ and Y'^-, R'^-, output species E^+, E^-, and parameters $s, q \in \mathbf{R}_{>0}$ the *subtraction block* is a CRN composed by the following reactions:

$$\begin{aligned} Y'^+ &\xrightarrow{s} Y'^+ + E^+ & R^- &\xrightarrow{s} R^- + E^+ \\ Y'^- &\xrightarrow{s} Y'^- + E^- & R^+ &\xrightarrow{s} R^+ + E^- \\ E^+ + E^- &\xrightarrow{q} \varnothing & E^+ &\xrightarrow{s} \varnothing & E^- &\xrightarrow{s} \varnothing \end{aligned}$$

Note that the input R^- produces an output E^+ and an input R^+ produces an output E^- which is different from the addition block presented above.

Finally, the converter block is described by the following:

Definition 29. Dual rail converter block For input species Y, output species Y'^+, Y'^-, and parameters $s, q \in \mathbf{R}_{>0}$ the *single to dual-rail converter block* is a CRN composed by the following reactions:

$$Y \xrightarrow{s} Y + Y'^+ \qquad Y'^+ \xrightarrow{s} \varnothing \qquad Y'^+ + Y'^- \xrightarrow{q} \varnothing$$

7.2.2 Integral Block

The integral component computes a multiple of the integral of the input signal. This action is widely used in control systems due to its ability to collect past information about the error to be corrected. A CRN implementation of the integral component has been proposed in [154] and is reported in Definition 30.

Definition 30. Integral block [154] For input species E^+, E^-, output species I^+, I^-, some constant $q \in \mathbf{R}_{>0}$ and multiplier $r \in \mathbf{R}_{\geq 0}$, the integral block is given by the following CRN:

$$E^+ \xrightarrow{r} E^+ + I^+ \qquad E^- \xrightarrow{r} E^- + I^- \qquad I^+ + I^- \xrightarrow{q} \varnothing$$

Proposition 2. The proof for the integral can be observed easily by taking the ODEs of the input and output species and following the proof of [154]. The integral blocks introduced in Definition 30 are correct. More formally, for all $1 \leq i \leq n$, it holds that

$$x_{I_i^+}(t) - x_{I_i^-}(t) = r \int_0^t x_{E_i^+}(\tau)d\tau - r \int_0^t x_{E_i^-}(\tau)d\tau.$$

Proof. This is straightforward via differentiation as we pick appropriate values at $t = 0$. □

7.2.3 Derivative Block

With reference to Tikhonov's Theorem defined in Definition 25, the following theorem shows that the above CRN is such that, under certain scaling of the rates, $(x_{D^+} - x_{D^-})$ produces a correct approximation of the derivative of $r(x_{E^+} - x_{E^-})$.

Theorem 2. The derivative blocks are asymptotically correct. In particular (the conditions unique to the derivative), the following holds:

1. The solution of (7.3) converges, on any bounded time interval, to an ODE system that satisfies $x_{D_i^+} - x_{D_i^-} = \partial_t x_{A_i^+} - \partial_t x_{A_i^-}$, for all $1 \leq i \leq n$, if $s \to \infty$ in all derivative blocks.
2. The solution of (7.3) converges, on any bounded time interval, to an ODE system that satisfies $x_{A_i^+} = r x_{E_i^+}$ and $x_{A_i^-} = r x_{E_i^-}$ for all $1 \leq i \leq n$, if $v \to \infty$ in all derivative blocks.

Proof (Sketch). To see point 1), we first note that Definition 24 yields:

$$\begin{aligned}
\partial_t x_{D_i^+} &= svrx_{E_i^+} + vsx_{A_i^-} - sx_{D_i^+} - qx_{D_i^+}x_{D_i^-},\\
\partial_t x_{D_i^-} &= svrx_{E_i^-} + vsx_{A_i^+} - sx_{D_i^-} - qx_{D_i^+}x_{D_i^-},\\
\partial_t x_{A_i^+} &= vrx_{E_i^+} - vx_{A_i^+},\\
\partial_t x_{A_i^-} &= vrx_{E_i^-} - vx_{A_i^-}.
\end{aligned}$$

Since this implies

$$\partial_t(x_{D_i^+} - x_{D_i^-}) = s(vrx_{E_i^+} - vx_{A_i^+}) - s(vrx_{E_i^-} - vx_{A_i^-}) - s(x_{D_i^+} - x_{D_i^-}),$$

this motivates us to add to the ODE system of (7.3) the additional ODE

$$\partial_t z_i = s\underbrace{(vrx_{E_i^+} - vx_{A_i^+})}_{=\partial_t x_{A_i^+}} - s\underbrace{(vrx_{E_i^-} - vx_{A_i^-})}_{=\partial_t x_{A_i^-}} - sz_i,$$

and to replace each instance of $(x_{D_i^+} - x_{D_i^-})$ in the ODE system with z_i. Recall that we need multiple derivative blocks to compute the derivative as we cannot set the rates to

∞. By processing the other derivative blocks in a similar fashion, we introduce new ODE variables $z = (z_1, \ldots, z_n)$. To see that the requirements of Tikhonov's theorem are satisfied, we note that the fast ODE system (i.e. the set of fast variables containing z, $D_1^+, \ldots, D_n^+$ and $D_1^-, \ldots, D_n^-$) admits, for any fixed vector of slow variables, exactly one equilibrium point. Additionally, it is an asymptotically stable equilibrium of the fast ODE system. To see point 2), instead, we apply Tikhonov's theorem in the case where, in every derivative block, $x_{A_1^+}, \ldots, x_{A_n^+}$ and $x_{A_1^-}, \ldots, x_{A_n^-}$ are treated as fast variables (while all other variables are considered to be slow) and $v \to \infty$ in all derivative blocks. □

7.3 PID control of Gene Expression

We wish to evaluate our PID controller on some biologically motivated examples. Gene expression is the process by which information from a gene is used in the synthesis of a functional gene product. These products are often proteins, but in non-protein coding genes such as transfer RNA (*tRNA*) or small nuclear RNA (*snRNA*) genes, the product is a functional *RNA*. This section we applies the feedback control architecture to a gene expression model. In particular, a $microRNA$ regulated gene expression model from [29] is considered, for which synthetic implementations have already been proposed in [187]. The model is composed by the following reactions, where for simplicity we fixed unitary kinetic parameters (i.e. for all of the rates $k = 1$):

$$\begin{aligned} \varnothing &\xrightarrow{1} mRNA & mRNA &\xrightarrow{1} \varnothing \\ mRNA &\xrightarrow{1} mRNA + Pro & Pro &\xrightarrow{1} \varnothing \\ mRNA + microRNA &\xrightarrow{1} \varnothing & microRNA &\xrightarrow{1} \varnothing \\ \varnothing &\xrightarrow{1} microRNA. \end{aligned}$$

That is, we have $mRNA$ which catalyses the production of the protein Pro. This production is dampened by an annihilation reaction with the $microRNA$.

The objective of our control is to have the protein Pro track a given reference signal (i.e. Pro is the output species of the Gene Expression CRN). Given U^+ and U^-, the control signals synthesized by the controller, we assume that these can act on the plant by regulating the expression rate of $mRNA$ and $microRNA$, respectively. This assumption is justified by the fact that these mechanisms can be implemented synthetically [187]. We consider the following reactions to model such actuation:

$$U^+ \to^1 U^+ + mRNA \qquad\qquad U^- \to^1 U^- + microRNA. \tag{7.4}$$

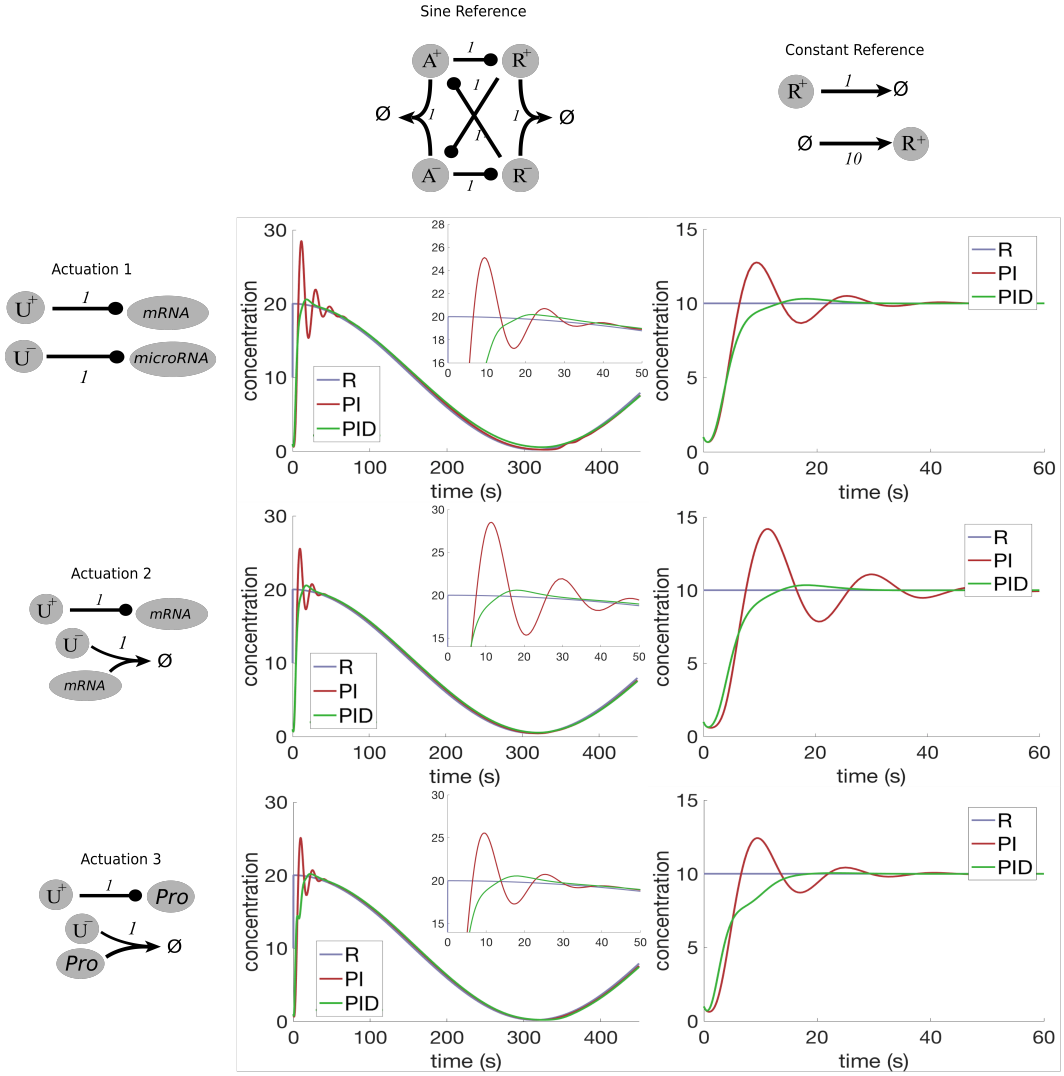


Figure 7.4: We consider the gene expression model given in Section 7.3 and compare the time evolution of the species Pro with different reference signals for PID and PI feedback control with the actuation models described in Equations (7.4), (7.5), and (7.6) - shown in the left, middle, and right column of the plots, respectively. It is important to note that the tuned parameters $K_p = 0.8, K_i = 0.3, K_d = 0.2$ are kept consistent throughout experimentation. We consider both a constant and a sine wave reference signal - shown in top and bottom rows of the plots, respectively. It is possible to observe that while Pro already tracks the reference signals for PI control correctly, in the case of a PID controller, the output has reduced oscillations around the reference signals. This is emphasized in the insets seen on the sine reference row where we examine the first 50 seconds of the time evolution.

In this model, a high concentration of U^+ increases the production rate of $mRNA$ and so of Pro, whereas a high concentration of U^- decreases the amount of $mRNA$ by producing $microRNA$ with a higher rate.

In the actuation model considered above, we have that the control signals act on two different species. This is not a requirement of our architecture. Another possible actuation is that U^- annihilates $mRNA$ directly. This can be modelled with the following reactions:

$$U^+ \rightarrow^1 U^+ + mRNA \quad U^- + mRNA \rightarrow^1 \varnothing. \tag{7.5}$$

Finally, another possibility is that U^+ and U^- act directly on the target species Pro. In this case, the actuation is:

$$U^+ \rightarrow^1 U^+ + Pro \quad U^- + Pro \rightarrow^1 \varnothing. \tag{7.6}$$

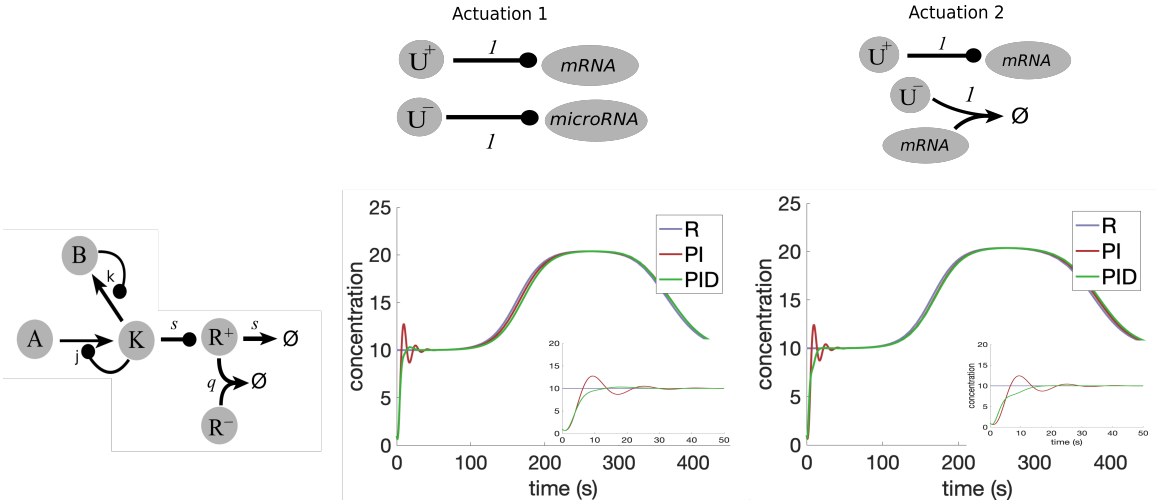


Figure 7.5: We consider the gene expression model given in Section 7.3 and compare the time evolution of the species Pro with a bellshape reference signal (taken from our synthesized CRN in Chapter 4 Example 4.1.1) for PID and PI feedback control with the actuation models described in Equations (7.4), (7.5). The bellshape reference signal is encoded as a dual-rail output via the CRN depicted. This shows that we can control plants to enact desired behaviours, such as those we have synthesized in previous chapters.

In Figure 7.4 we consider the different actuation mechanisms described above and compare the performance of PI and PID controllers for two different reference signals: a constant signal and an oscillatory signal. In Figure 7.5, we apply the same principles but this time with the bellshape reference signal (see Example 4.1.1 and Example 5.1.4). The motivation behind this reference signal is to show that we can control the plant in order to produce certain continuous profiles like the ones synthesized in previous chapters. If we want our plant to track a specific profile, we need to know the CRN (or ODE) for the specific profile to begin with. For common behaviours, this may be known, but for complex reference signals it may not. This motivates a combination of our synthesis methods to synthesize a reference signal in order to control a larger unknown system.

For all the plots in the figure, we consider the same parameters for PI and PID controllers. Whilst for the full parameter range, we can refer to Appendix C we define $K_p, K_i,$ and K_d from Chapter 3 Section 3.4 as follows:

```
rate Psum = 0.8;
rate Isum = 0.3;
rate Dsum = 0.2;
rate PIDsum = 1.4;
```

where `PIDsum` is the sum of the other components. The rates were tuned manually using the tuning methods described in the background chapter, however, one could improve the performance of the PID controller by using a tuning tool such as the one provided in `Matlab`.

It is easy to observe that in both the case of Figure 7.5 and Figure 7.4, whereas a negative feedback with PI control can already track both signals correctly, in the case of a PID controller, the time evolution of the concentration of Pro has reduced oscillations around the reference signals. In all cases the time for the convergence of the plant to the reference signal has decreased due to the action of the derivative block. This is due to the action of the derivative block which we know as the parameter values for `Psum,Isum` and `Dsum` are kept constant. In fact, while it is well known that the derivative component in a PID does not necessarily reach zero error at steady state, it can help to reduce the transient error between the output and the reference signals and to dampen oscillations around the set points.

Our work on CRN controllers was born out of discussions about addressing scalability issues with our synthesis methods. This chapter only produced work on deterministic systems and it is unclear whether the controller would operate under stochastic semantics, or even if solely the plant was a stochastic process. This problem could be addressed by considering the LNA semantics of a system as per Definition 17. Using the LNA we could observe the variance of the system over time and more accurately describe its time evolution under low molecular counts. We have already analyzed large CRN systems, that is CRNs that have many components, under the LNA semantics in our previous work [15].

7.4 Conclusion

This chapter considers feedback control with PID controllers and proposes a CRN implementation for this control architecture. This relies on a novel CRN, which computes the derivative of an input molecular signal that we have proved the correctness of. We applied our framework to control the protein expression in a $microRNA$ regulated gene expression

model and showed improved performance compared to a PI feedback control. We applied this with several reference signals, including the bellshape model referenced as a running example throughout this thesis (Chapter 4 Example 4.1.1), highlighting the fusion of the two approaches. An interesting future work, which has not been considered in this chapter, is to study the effect that the proposed control system has on noise [188]. We discuss this in more detail in the next chapter.

CHAPTER 8

Conclusions

This chapter concludes by first reflecting on the results and achievements of this thesis before discussing possible future works.

8.1 Conclusions

This thesis presents two approaches to the problem of inducing certain behaviours in Chemical Reaction Networks (CRNs), via parameter synthesis and control engineering methods. In order to synthesize CRNs, we have retrospectively developed a framework that captures unknown or partial information within CRNs, known as a sketching language, based on the `metasketches` framework for computer programming languages. This sketching language can adequately and concisely capture syntactic constraints, while at the same time allowing under-specification of the network.

We use this framework to synthesize both discrete structures of a network and rate parameters, based upon the Linear Noise Approximation (LNA) semantics, an approximation on the stochastic dynamics of a CRN. Based on a cost function that reflects the structure of a given CRN, we aim to produce a CRN optimal for that given cost function. We employ SMT-ODE solvers to solve parametric ODEs - a combination of the LNA dynamics and our choice encodings. We successfully demonstrate the generalization of the problem to include non mass-action based semantics, as well as ODE inputs to a CRN network. We construct several examples to systematically demonstrate this.

We present the first derivative CRN component implementation which, given any continuous input, produces the derivative of that input as output. By relying on this component,

we develop a CRN implementation of a feedback control loop with a Proportional-Integral-Derivative (PID) controller and apply the resulting control architecture to regulate the protein expression in a *microRNA* regulated gene expression model. Proofs for the workings of these components are also provided.

We provide an evaluation of this work that discusses possible shortcomings, and therefore presented possible directions for how to widen this field of research as well as new pathways of research to explore. Parameter synthesis has applications in synthetic and systems biology, for example, design automation for molecular devices [17], or the construction of predictive molecular models [18]. In general, as CRN implementations in molecular devices such as DNA become more complicated the need for greater automation methods becomes yet more apparent. Molecular implementation of control systems has been shown to naturally occur in living organisms [150, 151, 152]. As a consequence, in view of the potential applications, CRN designs that implement control mechanisms are sought after [9, 141]. With this in mind, this thesis has sought to contribute methods for parameter synthesis and control engineering that has applications within real world systems and, with the sketching language and methods proposed, can be extended to wider frameworks of synthesis.

8.2 Future Developments

This section discusses future developments in this field of research, either from building on previous chapters or originating from co-authors.

8.2.1 Local Search

Given a nonlinear parametric dynamical system and an MTL specification for that system, it has been shown that a trajectory can violate or satisfy a specification using a local search [189]. By implementing a similar method we could provide a local search for the user which can yield a concrete CRN, though without guarantee on accuracy. In situations where a user requires a fast result, such as obtaining an approximate estimate of a solution, this could be advantageous. We could combine our previous work with current state of the art in gradient descent. In [190] a robustness metric with a biologically motivated case study paved the way for near satisfaction of a specification. This near satisfaction relies on a local search algorithm guided by a gradient descent metric. In [109] a method is proposed for guiding an approach based upon *Covariance Matrix Adaptation Evolution Search* (CMA-ES), via a metric for property robustness of an LTL formula over parametric non-linear ODEs. The CMA-ES is a common method for providing ODE traces. The advantage of such a local search is that the degree of satisfaction is quantifiable, and therefore optimizable. The CMA-ES has a Python implementation: `https://github.com/CMA-ES/pycma`.

8.2.2 Falsifying Parameter Regions

An interesting problem that arises from trying to improve search techniques for large search spaces is the falsification of parameter regions. Given a reaction sketch, can we show that certain discrete concrete instantiations never satisfy a specification regardless of their continuous parameters? The problem of testing for the falsification of a property by systems has been investigated by many others (see related research section in [191]). Most of the research focused on parameter estimation [190, 192]. Recently, however, the problem of temporal falsification for hybrid systems has received attention [193, 191]. Unfortunately, the publicly available tool support in this space is limited to *S-Taliro* and *BREACH*. *BREACH* does not support temporal logic falsification for arbitrary models. Usage of the *S-Taliro* tool must be provided through a *Simulink* model.

8.2.3 Specifications for Oscillation

Chapter 6 successfully synthesized behaviours for a bellshape, sigmoidal, switch and Poisson responses based upon a MTL-fragment specification. To specify the desired *temporal behaviour* of the network, we support constraints about the expected number and variance of molecules, and, crucially, their derivatives over time. Metric Temporal Logic (MTL) specifications can capture complex state and timing requirements. However, MTL has its limitations. An extension would be to complement this evolution toward a fusion of time-domain and frequency-domain analysis by proposing a unified logical formalism for expressing hybrid (time-frequency) properties of signals. We would achieve this by employing a fragment of STL that includes frequency operators [130]. STL and equivalents has been used to synthesize interesting harmonic oscillators, most notably in music technology [194].

8.2.4 Control of Stochastic Systems

In Chapter 7, our controller was tested under deterministic semantics. While simulations on a CTMC via the CME is tractable, it requires many simulations in order to provide sufficient accuracy. This is especially true for stiff systems, i.e. those with reactions with rates of different order [195]. It would be possible to test the controller via the LNA semantics which would mitigate these issues as the mean and variance is given by a set of differential equations quadratic in the number of species. Observing whether or not there is a large variance around the mean over time may show that under low molecular counts our controller is infeasible. This could be overcome by simplifying the derivative component.

Bibliography

[1] Nobufusa Serizawa. Biochemical and molecular approaches for production of pravastatin, a potent cholesterol-lowering drug. In *Biotechnology annual review*, volume 2, pages 373–389. Elsevier, 1996.

[2] Sanjay Gurunathan, Dennis M Klinman, and Robert A Seder. DNA vaccines: immunology, application, and optimization. *Annual review of immunology*, 18(1):927–974, 2000.

[3] Irene Otero-Muras, Pencho Yordanov, and Joerg Stelling. Chemical reaction network theory elucidates sources of multistability in interferon signaling. *PLoS computational biology*, 13(4):e1005454, 2017.

[4] Luca Cardelli. On process rate semantics. *Theoretical Computer Science*, 391(3):190–215, 2008.

[5] Tim Liedl, Thomas L Sobey, and Friedrich C Simmel. DNA-based nanodevices. *Nano Today*, 2(2):36–41, 2007.

[6] Bernard Yurke, Andrew J Turberfield, Allen P Mills, Friedrich C Simmel, and Jennifer L Neumann. A DNA-fuelled molecular machine made of DNA. *Nature*, 406(6796):605–608, 2000.

[7] Luca Cardelli. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science*, 23(02):247–271, 2013.

[8] Naama Barkai and Stan Leibler. Robustness in simple biochemical networks. *Nature*, 387(6636):913, 1997.

[9] Corentin Briat, Ankit Gupta, and Mustafa Khammash. Antithetic integral feedback ensures robust perfect adaptation in noisy biomolecular networks. *Cell systems*, 2(1):15–26, 2016.

[10] Neil Dalchau, Niall Murphy, Rasmus Petersen, and Boyan Yordanov. Synthesizing and tuning chemical reaction networks with specified behaviours. In *International Workshop on DNA-Based Computers*, pages 16–33. Springer, 2015.

[11] James Aspnes and Eric Ruppert. An introduction to population protocols. In *Middleware for Network Eccentric and Mobile Applications*, pages 97–120. Springer, 2009.

[12] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

[13] Luca Cardelli, Milan Češka, Martin Fränzle, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, and Max Whitby. Syntax-guided optimal synthesis for chemical reaction networks. In *International Conference on Computer Aided Verification*, pages 375–395. Springer, 2017.

[14] Luca Cardelli, Marta Kwiatkowska, and Max Whitby. Chemical reaction network designs for asynchronous logic circuits. *Natural computing*, 17(1):109–130, 2018.

[15] Luca Cardelli, Marta Kwiatkowska, and Max Whitby. *Chemical Reaction Network Designs for Asynchronous Logic Circuits*, pages 67–81. Springer International Publishing, Cham, 2016.

[16] Lulu Qian, Erik Winfree, and Jehoshua Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475(7356):368–372, 2011.

[17] Adam Noel, Karen C Cheung, and Robert Schober. Joint channel parameter estimation via diffusive molecular communication. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 1(1):4–17, 2015.

[18] Jiri Barnat, Lubos Brim, Adam Krejci, Adam Streck, David Safranek, Martin Vejnar, and Tomas Vejpustek. On parameter synthesis by parallel model checking. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 9(3):693–705, 2012.

[19] Milan Češka, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Informatica*, 54(6):589–623, 2017.

[20] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263. Springer, 2013.

[21] Milan Češka, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Informatica*, pages 1–35, 2016.

[22] C Zimmer and S Sahle. Parameter estimation for stochastic models of biochemical reactions. *J Comput Sci Syst Biol*, 6:011–021, 2012.

[23] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. *Dependable Software Systems Engineering*, 40:1–25, 2015.

[24] Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. Programming by sketching for bit-streaming programs. In *Proc. of PLDI'05*, pages 281–294. ACM, 2005.

[25] Armando Solar-Lezama, Christopher Grant Jones, and Rastislav Bodik. Sketching concurrent data structures. In *Proc. of PLDI'08*, pages 136–148. ACM, 2008.

[26] Ali Sinan Koksal, Yewen Pu, Saurabh Srivastava, Rastislav Bodik, Jasmin Fisher, and Nir Piterman. Synthesis of biological models from mutation experiments. In *Proc. of POPL'13*, pages 469–482. ACM, 2013.

[27] James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze. Optimizing synthesis with metasketches. In *Proc. of POPL'16*, pages 775–788. ACM, 2016.

[28] Jörn M Schmiedel, Sandy L Klemm, Yannan Zheng, Apratim Sahay, Nils Blüthgen, Debora S Marks, and Alexander van Oudenaarden. Microrna control of protein expression noise. *Science*, 348(6230):128–132, 2015.

[29] Luca Laurenti, Attila Csikasz-Nagy, Marta Kwiatkowska, and Luca Cardelli. Molecular filters for noise reduction. *Biophysical Journal*, 114(12):3000–3011, 2018.

[30] Max Whitby, Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Mirco Tribastone, and Max Tschaikowski. PID control of biochemical reaction networks. *CoRR*, abs/1903.10390, 2019.

[31] James Scott-Brown. *Visualisation in the Synthetic Biology Design Cycle*. PhD thesis, University of Oxford, 2018.

[32] Andreas Eggers, Martin Fränzle, and Christian Herde. Sat modulo ODE: A direct SAT approach to hybrid systems. In *ATVA'08*, pages 171–185. Springer, 2008.

[33] Sicun Gao, Soonho Kong, and Edmund M Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *CADE'13*, pages 208–214. Springer, 2013.

[34] Péter Érdi and János Tóth. *Mathematical models of chemical reactions: theory and applications of deterministic and stochastic models*. Manchester University Press, 1989.

[35] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

[36] Olaf Wolkenhauer, Mukhtar Ullah, Walter Kolch, and Kwang-Hyun Cho. Modeling and simulation of intracellular dynamics: choosing an appropriate framework. *IEEE transactions on nanobioscience*, 3(3):200–207, 2004.

[37] Markus Hegland, Conrad Burden, Lucia Santoso, Shev MacNamara, and Hilary Booth. A solver for the stochastic master equation applied to gene regulatory networks. *Journal of computational and applied mathematics*, 205(2):708–724, 2007.

[38] Béla Novák and John J Tyson. Design principles of biochemical oscillators. *Nature reviews Molecular cell biology*, 9(12):981–991, 2008.

[39] Gheorghe Craciun and Martin Feinberg. Multiple equilibria in complex chemical reaction networks: semiopen mass action systems. *SIAM Journal on Applied Mathematics*, 70(6):1859–1877, 2010.

[40] Luca Cardelli. Strand algebras for DNA computing. In *DNA computing and molecular programming*, pages 12–24. Springer, 2009.

[41] George F Oster and Alan S Perelson. Chemical reaction dynamics. *Archive for rational mechanics and analysis*, 55(3):230–274, 1974.

[42] Oleg N Temkin, Andrew V Zeigarnik, and DG Bonchev. *Chemical reaction networks: a graph-theoretical approach*. CRC Press, 1996.

[43] Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. In *Algorithmic Bioprocesses*, pages 543–584. Springer, 2009.

[44] John H Conway. Fractran: A simple universal programming language for arithmetic. In *Open Problems in Communication and Computation*, pages 4–26. Springer, 1987.

[45] Richard M Karp and Raymond E Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969.

[46] Javier Esparza. *On the decidability of model checking for several μ-calculi and Petri nets*. Springer, 1994.

[47] Marvin L Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

[48] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[49] Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 292–299. ACM, 2006.

[50] Ho-Lin Chen, David Doty, and David Soloveichik. Rate-independent computation in continuous chemical reaction networks. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 313–326. ACM, 2014.

[51] Rachel Cummings, David Doty, and David Soloveichik. Probability 1 computation with chemical reaction networks. In *DNA Computing and Molecular Programming*, pages 37–52. Springer, 2014.

[52] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *natural computing*, 7(4):615–633, 2008.

[53] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, 2008.

[54] Gianluigi Zavattaro and Luca Cardelli. Termination problems in chemical kinetics. In *CONCUR 2008-Concurrency Theory*, pages 477–491. Springer, 2008.

[55] Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural computing*, 13(4):517–534, 2014.

[56] Nicolaas Godfried Van Kampen. *Stochastic processes in physics and chemistry*, volume 1. Elsevier, 1992.

[57] Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. Stochastic analysis of chemical reaction networks using linear noise approximation. In *International Conference on Computational Methods in Systems Biology*, pages 64–76. Springer, 2015.

[58] Martin Feinberg. Lectures on chemical reaction networks. *Notes of lectures given at the Mathematics Research Center, University of Wisconsin*, 1979.

[59] Fritz Horn and Roy Jackson. General mass action kinetics. *Archive for rational mechanics and analysis*, 47(2):81–116, 1972.

[60] David F Anderson, Germán A Enciso, and Matthew D Johnston. Stochastic analysis of biochemical reaction networks with absolute concentration robustness. *Journal of The Royal Society Interface*, 11(93):20130943, 2014.

[61] Daniel T Gillespie. Deterministic limit of stochastic chemical kinetics. *The Journal of Physical Chemistry B*, 113(6):1640–1644, 2009.

[62] Daniel T Gillespie. A rigorous derivation of the chemical master equation. *Physica A: Statistical Mechanics and its Applications*, 188(1):404–425, 1992.

[63] Stewart N Ethier and Thomas G Kurtz. *Markov processes: characterization and convergence*, volume 282. John Wiley & Sons, 2009.

[64] Williams J Stewart. *Introduction to the numerical solutions of Markov chains*. Princeton Univ. Press, 1994.

[65] Abhyudai Singh and João P Hespanha. Approximate moment dynamics for chemically reacting systems. *Automatic Control, IEEE Transactions on*, 56(2):414–418, 2011.

[66] Abhyudai Singh and Joao Pedro Hespanha. Lognormal moment closures for biochemical reactions. In *Decision and Control, 2006 45th IEEE Conference on*, pages 2063–2068. IEEE, 2006.

[67] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[68] Timothy S Gardner, Charles R Cantor, and James J Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339, 2000.

[69] Daniel T Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001.

[70] Luca Bortolussi and Roberta Lanciani. Model checking markov population models by central limit approximation. In *Quantitative Evaluation of Systems*, pages 123–138. Springer, 2013.

[71] Paul Fearnhead, Vasilieos Giagos, and Chris Sherlock. Inference for reaction networks using the linear noise approximation. *Biometrics*, 70(2):457–466, 2014.

[72] Jianshu Cao. Michaelis- menten equation and detailed balance in enzymatic networks. *The Journal of Physical Chemistry B*, 115(18):5493–5498, 2011.

[73] John Heath, Marta Kwiatkowska, Gethin Norman, David Parker, and Oksana Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257, 2008.

[74] Y Yamada and Y Seino. Physiology of GIP-a lesson from GIP receptor knockout mice. *Hormone and metabolic research*, 36(11/12):771–774, 2004.

[75] DF Savage, J Way, and PA Silver. Defossiling fuel: how synthetic biology can transform biofuel production. *ACS chemical biology*, 3(1):13–16, 2008.

[76] Pamela P Peralta-Yahya, Fuzhong Zhang, Stephen B Del Cardayre, and Jay D Keasling. Microbial engineering for the production of advanced biofuels. *Nature*, 488(7411):320, 2012.

[77] Amor A Menezes, Michael G Montague, John Cumbers, John A Hogan, and Adam P Arkin. Grand challenges in space synthetic biology. *Journal of The Royal Society Interface*, 12(113):20150803, 2015.

[78] Joy Sinha, Samuel J Reyes, and Justin P Gallivan. Reprogramming bacteria to seek and destroy an herbicide. *Nature chemical biology*, 6(6):464, 2010.

[79] Lara Tess Bereza-Malcolm, Gulay Mann, and Ashley Edwin Franks. Environmental sensing of heavy metals through whole cell microbial biosensors: a synthetic biology approach. *ACS synthetic biology*, 4(5):535–546, 2014.

[80] Tal Danino, Arthur Prindle, Gabriel A Kwong, Matthew Skalak, Howard Li, Kaitlin Allen, Jeff Hasty, and Sangeeta N Bhatia. Programmable probiotics for detection of cancer in urine. *Science translational medicine*, 7(289):289ra84–289ra84, 2015.

[81] Deboki Chakravarti and Wilson W Wong. Synthetic biology in cell-based cancer immunotherapy. *Trends in biotechnology*, 33(8):449–461, 2015.

[82] Marcelo O. Magnasco. Chemical kinetics is Turing universal. *Phys. Rev. Lett.*, 78:1190–1193, Feb 1997.

[83] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. J. Turberfield. DNA walker circuits: Computational potential, design and verification. *Natural Computing*, 14(2):195–211, 2015.

[84] Toshiki Matsumine, Toshiaki Koike-Akino, and Ye Wang. Polar coding with chemical reaction networks. *CoRR*, abs/1903.03709, 2019.

[85] Luca Cardelli. Morphisms of reaction networks that couple structure to function. *BMC systems biology*, 8(1):84, 2014.

[86] Nils E Napp and Ryan P Adams. Message passing inference with chemical reaction networks. In *Advances in Neural Information Processing Systems*, pages 2247–2255, 2013.

[87] Phillip Senum and Marc Riedel. Rate-independent constructs for chemical computation. *PloS One*, 6(6):e21414, 2011.

[88] A Prasanna de Silva and Nathan D McClenaghan. Molecular-scale logic gates. *Chemistry–A European Journal*, 10(3):574–586, 2004.

[89] Adam Shea, Brian Fett, Marc D Riedel, and Keshab K Parhi. Writing and compiling code into biochemistry. In *Pacific Symposium on Biocomputing*, volume 15, pages 456–464, 2010.

[90] Hua Jiang, Marc D Riedel, and Keshab K Parhi. Digital signal processing with molecular reactions. *IEEE Design and Test of Computers*, 29(3):21–31, 2012.

[91] Lulu Qian, David Soloveichik, and Erik Winfree. Efficient turing-universal computation with DNA polymers. In *DNA computing and molecular programming*, pages 123–140. Springer, 2011.

[92] Chris Thachuk and Anne Condon. Space and energy efficient computation with DNA strand displacement systems. In *DNA Computing and Molecular Programming*, pages 135–149. Springer, 2012.

[93] Xianbin Li, Liangzhong Shen, Xuequn Shang, and Wenbin Liu. Subpathway analysis based on signaling-pathway impact analysis of signaling pathway. *PloS one*, 10(7):e0132813, 2015.

[94] Luca Grieco, Laurence Calzone, Isabelle Bernard-Pierrot, François Radvanyi, Brigitte Kahn-Perles, and Denis Thieffry. Integrative modelling of the influence of mapk network on cancer cell fate decision. *PLoS computational biology*, 9(10):e1003286, 2013.

[95] Marta Z Kwiatkowska and John K Heath. Biological pathways as communicating computer systems. *Journal of cell science*, 122(16):2793–2800, 2009.

[96] Satoru Sasagawa, Yu-ichi Ozaki, Kazuhiro Fujita, and Shinya Kuroda. Prediction and validation of the distinct dynamics of transient and sustained erk activation. *Nature cell biology*, 7(4):365, 2005.

[97] Gérard Berry and Gérard Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 81–94. ACM, 1989.

[98] Charles H Bennett. The thermodynamics of computationa review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.

[99] Paul WK Rothemund. A DNA and restriction enzyme implementation of turing machines. *DNA based computers*, 27:75–119, 1996.

[100] Paul WK Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA sierpinski triangles. *PLoS Biol*, 2(12):e424, 2004.

[101] Nam-phuong Nguyen, Chris Myers, Hiroyuki Kuwahara, Chris Winstead, and James Keener. Design and analysis of a robust genetic Muller C-element. *Journal of theoretical biology*, 264(2):174–187, 2010.

[102] Nam-Phuong D Nguyen, Hiroyuki Kuwahara, Chris J Myers, and James P Keener. The design of a genetic Muller C-element. In *Asynchronous Circuits and Systems, 2007. ASYNC 2007. 13th IEEE International Symposium on*, pages 95–104. IEEE, 2007.

[103] Avigdor Eldar and Michael B Elowitz. Functional roles for noise in genetic circuits. *Nature*, 467(7312):167–173, 2010.

[104] Harley H McAdams and Adam Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences*, 94(3):814–819, 1997.

[105] Ron Weiss, George E Homsy, and Thomas F Knight. Toward in vivo digital circuits. In *Evolution as Computation*, pages 275–295. Springer, 2002.

[106] Mirco Giacobbe, Călin C Guet, Ashutosh Gupta, Thomas A Henzinger, Tiago Paixao, and Tatjana Petrov. Model checking gene regulatory networks. In *TACAS'15*, pages 469–483. Springer, 2015.

[107] Matthew R Lakin, David Parker, Luca Cardelli, Marta Kwiatkowska, and Andrew Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, page rsif20110800, 2012.

[108] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Formal methods for performance evaluation*, pages 220–270. Springer, 2007.

[109] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *International Conference on Computational Methods in Systems Biology*, pages 251–268. Springer, 2008.

[110] Kurt W Kohn. Molecular interaction map of the mammalian cell cycle control and DNA repair systems. *Molecular biology of the cell*, 10(8):2703–2734, 1999.

[111] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996.

[112] Alexandre Donzé and Oded Maler. *Robust satisfaction of temporal logic over real-valued signals*. Springer, 2010.

[113] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[114] Luboš Brim, Petr Dluhoš, David Šafránek, and Tomáš Vejpustek. STL*: Extending signal temporal logic with signal-value freezing operator. *Information and Computation*, 236:52–67, 2014.

[115] Aurore Annichini, Eugene Asarin, and Ahmed Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *International Conference on Computer Aided Verification*, pages 419–434. Springer, 2000.

[116] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

[117] Thomas A Henzinger, Benjamin Horowitz, Rupak Majumdar, and Howard Wong-Toi. Beyond hytech: Hybrid systems analysis using interval numerical methods. In *International Workshop on Hybrid Systems: Computation and Control*, pages 130–144. Springer, 2000.

[118] Goran Frehse, Sumit Kumar Jha, and Bruce H Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *International Workshop on Hybrid Systems: Computation and Control*, pages 187–200. Springer, 2008.

[119] Monika Heiner, David Gilbert, and Robin Donaldson. Petri nets for systems and synthetic biology. In *International school on formal methods for the design of computer, communication and software systems*, pages 215–264. Springer, 2008.

[120] Alexandre Donzé, Gilles Clermont, Axel Legay, and Christopher J Langmead. Parameter synthesis in nonlinear dynamical systems: Application to systems biology. In *Annual International Conference on Research in Computational Molecular Biology*, pages 155–169. Springer, 2009.

[121] Georgios E Fainekos and George J Pappas. Robust sampling for MITL specifications. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 147–162. Springer, 2007.

[122] Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. In *Readings in artificial intelligence and software engineering*, pages 3–34. Elsevier, 1986.

[123] Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. Programming by sketching for bit-streaming programs. In *ACM SIGPLAN Notices*, volume 40, pages 281–294. ACM, 2005.

[124] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. In *ACM Sigplan Notices*, volume 40, pages 213–223. ACM, 2005.

[125] Mukund Raghothaman and Abhishek Udupa. Language to specify syntax-guided synthesis problems. *arXiv preprint arXiv:1405.5590*, 2014.

[126] Xujie Si, Woosuk Lee, Richard Zhang, Aws Albarghouthi, Paraschos Koutris, and Mayur Naik. Syntax-guided synthesis of datalog programs. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 515–527. ACM, 2018.

[127] Simon Peyton Jones, Alan Blackwell, and Margaret Burnett. A user-centred approach to functions in excel. *Acm sigplan notices*, 38(9):165–176, 2003.

[128] Rajeev Alur, Dana Fisman, Rishabh Singh, and Armando Solar-Lezama. Sygus-comp 2016: Results and analysis. *arXiv preprint arXiv:1611.07627*, 2016.

[129] Manos Koukoutos, Mukund Raghothaman, Etienne Kneuss, and Viktor Kuncak. On repair with probabilistic attribute grammars. *arXiv preprint arXiv:1707.04148*, 2017.

[130] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*, pages 239–248. ACM, 2015.

[131] Jinseong Jeon, Xiaokang Qiu, Armando Solar-Lezama, and Jeffrey S Foster. Adaptive concretization for parallel program synthesis. In *International Conference on Computer Aided Verification*, pages 377–394. Springer, 2015.

[132] Qinheping Hu and Loris D'Antoni. Syntax-guided synthesis with quantitative syntactic objectives. In *International Conference on Computer Aided Verification*, pages 386–403. Springer, 2018.

[133] Sicun Gao, Jeremy Avigad, and Edmund M Clarke. δ-complete decision procedures for satisfiability over the reals. In *IJCAR'12*, pages 286–300. Springer, 2012.

[134] Andreas Eggers, Nacim Ramdani, Nedialko S. Nedialkov, and Martin Fränzle. Improving the SAT modulo ODE approach to hybrid systems analysis by combining different enclosure methods. *Softw. Syst. Model.*, 14(1):121–148, 2015.

[135] Alexandre Donzé, Gilles Clermont, and Christopher J Langmead. Parameter synthesis in nonlinear dynamical systems: Application to systems biology. *Journal of Computational Biology*, 17(3):325–336, 2010.

[136] Robin Donaldson and David Gilbert. A model checking approach to the parameter estimation of biochemical pathways. In *International Conference on Computational Methods in Systems Biology*, pages 269–287. Springer, 2008.

[137] Martin Demko, Nikola Beneš, Luboš Brim, Samuel Pastva, and David Šafránek. High-performance symbolic parameter synthesis of biological models: a case study. In *International Conference on Computational Methods in Systems Biology*, pages 82–97. Springer, 2016.

[138] Sergiy Bogomolov, Christian Schilling, Ezio Bartocci, Gregory Batt, Hui Kong, and Radu Grosu. Abstraction-based parameter synthesis for multiaffine systems. In *Haifa Verification Conference*, pages 19–35. Springer, 2015.

[139] Gabriele Lillacci and Mustafa Khammash. Parameter estimation and model selection in computational biology. *PLoS computational biology*, 6(3):e1000696, 2010.

[140] Hans Christian Hensel, David Parker, and Joost-Pieter Katoen. The probabilistic model checker storm: symbolic methods for probabilistic model checking. Technical report, Fachgruppe Informatik, 2019.

[141] Domitilla Del Vecchio, Aaron J Dy, and Yili Qian. Control theory meets synthetic biology. *Journal of The Royal Society Interface*, 13(120):20160380, 2016.

[142] Joseph L Izzo and Addison A Taylor. The sympathetic nervous system and baroreflexes in hypertension and hypotension. *Current hypertension reports*, 1(3):254–263, 1999.

[143] Merav Socolovsky, Michael Murrell, Ying Liu, Ramona Pop, Ermelinda Porpiglia, and Andre Levchenko. Negative autoregulation by fas mediates robust fetal erythropoiesis. *PLoS biology*, 5(10):e252, 2007.

[144] Yuze Mao, Hongsheng Yang, and Rucai Wang. Bioremediation capability of large–sized seaweed in integrated mariculture ecosystem: A review. *Journal of Fishery Sciences of China*, 12(2):225–231, 2005.

[145] Andreas Milias-Argeitis, Sean Summers, Jacob Stewart-Ornstein, Ignacio Zuleta, David Pincus, Hana El-Samad, Mustafa Khammash, and John Lygeros. In silico feedback for in vivo regulation of a gene expression circuit. *Nature biotechnology*, 29(12):1114, 2011.

[146] James E Ferrell Jr. Self-perpetuating states in signal transduction: positive feedback, double-negative feedback and bistability. *Current opinion in cell biology*, 14(2):140–148, 2002.

[147] Douglas J Pitera, Chris J Paddon, Jack D Newman, and Jay D Keasling. Balancing a heterologous mevalonate pathway for improved isoprenoid production in escherichia coli. *Metabolic engineering*, 9(2):193–207, 2007.

[148] Avantika A Shastri and John A Morgan. Flux balance analysis of photoautotrophic metabolism. *Biotechnology progress*, 21(6):1617–1626, 2005.

[149] Ciarán L Kelly, Andreas W K Harris, Harrison Steel, Edward J Hancock, John T Heap, and Antonis Papachristodoulou. Synthetic negative feedback circuits using engineered small rnas. *Nucleic acids research*, 46(18):9875–9889, 2018.

[150] Fuzhong Zhang, James M Carothers, and Jay D Keasling. Design of a dynamic sensor-regulator system for production of chemicals and fuels derived from fatty acids. *Nature biotechnology*, 30(4):354, 2012.

[151] Mary J Dunlop, Jay D Keasling, and Aindrila Mukhopadhyay. A model for improving microbial biofuel production using a synthetic feedback loop. *Systems and synthetic biology*, 4(2):95–104, 2010.

[152] Tau-Mu Yi, Yun Huang, Melvin I Simon, and John Doyle. Robust perfect adaptation in bacterial chemotaxis through integral feedback control. *Proceedings of the National Academy of Sciences*, 97(9):4649–4653, 2000.

[153] Jordan Ang, Sangram Bagh, Brian P Ingalls, and David R McMillen. Considerations for using integral feedback control to construct a perfectly adapting synthetic gene network. *Journal of theoretical biology*, 266(4):723–738, 2010.

[154] Kevin Oishi and Eric Klavins. Biomolecular implementation of linear I/O systems. *IET systems biology*, 5(4):252–260, 2011.

[155] Jordan Ang and David R McMillen. Physical constraints on biological integral control design for homeostasis and sensory adaptation. *Biophysical journal*, 104(2):505–515, 2013.

[156] Wolfgang Halter, Zoltan A Tuza, and Frank Allgöwer. Signal differentiation with genetic networks. *IFAC-PapersOnLine*, 50(1):10938–10943, 2017.

[157] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–13. Springer, 2008.

[158] Paul Hunter, Joël Ouaknine, and James Worrell. Expressive completeness for metric temporal logic. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 349–357. IEEE, 2013.

[159] Sicun Gao, Soonho Kong, and Edmund M. Clarke. Satisfiability modulo ODEs. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 105–112. IEEE, 2013.

[160] Stefanie Winkelmann and Christof Schütte. Hybrid models for chemical reaction networks: Multiscale theory and application to gene regulatory systems. *The Journal of chemical physics*, 147(11):114, 2017.

[161] Flávia Alexandra Mendes Penim. *A stochastic model of gene expression including splicing events*. PhD thesis, University of Lisboa, 2014.

[162] Fabio Luciani, Can Keşmir, Michele Mishto, Michal Or-Guil, and Rob J De Boer. A mathematical model of protein degradation by the proteasome. *Biophysical journal*, 88(4):2422–2432, 2005.

[163] Andreas Eggers. *Direct handling of ordinary differential equations in constraint-solving-based analysis of hybrid systems*. PhD thesis, Universität Oldenburg, 2014.

[164] Andreas Eggers, Nacim Ramdani, Nedialko Nedialkov, and Martin Fränzle. Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods. In *International Conference on Software Engineering and Formal Methods*, pages 172–187. Springer, 2011.

[165] F. Benhamou, F. Goualard, L. Granvilliers, and J. F. Puget. Revising hull and box consistency. In *ICLP'99*, pages 230–244. MIT Press, 1999.

[166] Vu Xuan Tung, To Van Khanh, and Mizuhito Ogawa. rasat: An smt solver for polynomial constraints. In *IJCAR'16*, pages 228–237. Springer, 2016.

[167] Karl Johan Åström and Tore Hägglund. *PID controllers: theory, design, and tuning*, volume 2. Instrument society of America Research Triangle Park, NC, 1995.

[168] John G Ziegler and Nathaniel B Nichols. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942.

[169] Nicolas Le Novere, Michael Hucka, Huaiyu Mi, Stuart Moodie, Falk Schreiber, Anatoly Sorokin, Emek Demir, Katja Wegner, Mirit I Aladjem, Sarala M Wimalaratne, et al. The systems biology graphical notation. *Nature biotechnology*, 27(8):735, 2009.

[170] David F Anderson and Thomas G Kurtz. Continuous time markov chain models for chemical reaction networks. In *Design and analysis of biomolecular circuits*, pages 3–42. Springer, 2011.

[171] Thomas A Henzinger, Maria Mateescu, and Verena Wolf. Sliding window abstraction for infinite markov chains. In *Computer Aided Verification*, pages 337–352. Springer, 2009.

[172] Marta Kwiatkowska and Chris Thachuk. Probabilistic model checking for biology. *Software Systems Safety*, 36:165, 2014.

[173] Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. Stochastic analysis of chemical reaction networks using linear noise approximation. In *Computational Methods in Systems Biology*, pages 64–76. Springer, 2015.

[174] Luca Cardelli. Artificial biochemistry. In *Algorithmic Bioprocesses*, pages 429–462. Springer, 2009.

[175] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Symbolic computation of differential equivalences. In *ACM SIGPLAN Notices*, volume 51, pages 137–150. ACM, 2016.

[176] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Comparing chemical reaction networks: A categorical and algorithmic perspective. In *LICS'16*, pages 485–494. ACM, 2016.

[177] Alessandro Abate, Cristina David, Pascal Kesseli, Daniel Kroening, and Elizabeth Polgreen. Counterexample guided inductive synthesis modulo theories. In *International Conference on Computer Aided Verification*, pages 270–288. Springer, 2018.

[178] Susmit Jha, Sumit Gulwani, Sanjit A Seshia, and Ashish Tiwari. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 215–224. ACM, 2010.

[179] Luca Bortolussi, Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. Approximation of probabilistic reachability for chemical reaction networks using the linear noise approximation. In *QEST'16*, pages 72–88. Springer, 2016.

[180] Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. Programming discrete distributions with chemical reaction networks. In *DNA'16*, pages 35–51. Springer, 2016.

[181] Attila Csikász-Nagy, Luca Cardelli, and Orkun S Soyer. Response dynamics of phosphorelays suggest their potential utility in cell signalling. *Journal of The Royal Society Interface*, 8(57):480–488, 2011.

[182] Suresh Kumar Poovathingal and Rudiyanto Gunawan. Global parameter estimation methods for stochastic biochemical systems. *BMC bioinformatics*, 11:414, 2010.

[183] Orkun S Soyer, Marcel Salathe, and Sebastian Bonhoeffer. Signal transduction networks: topology, response and biochemical processes. *Journal of theoretical biology*, 238(2):416–425, 2006.

[184] Ferdinand Verhulst. *Methods and Applications of Singular Perturbations*. Springer, 2005.

[185] Luca Bortolussi and Rytis Paskauskas. Mean-field approximation and quasi-equilibrium reduction of markov population models. In *QEST*, pages 106–121, 2014.

[186] Tomislav Plesa, Tomáš Vejchodskỳ, and Radek Erban. Chemical reaction systems with a homoclinic bifurcation: an inverse problem. *Journal of Mathematical Chemistry*, 54(10):1884–1915, 2016.

[187] Nicolas Delalez, Aivar Sootla, George H Wadhams, and Antonis Papachristodoulou. Design of a synthetic srna-based feedback filter module. *BioRxiv*, page 504449, 2018.

[188] Corentin Briat, Ankit Gupta, and Mustafa Khammash. Antithetic proportional-integral feedback for reduced variance and improved control performance of stochastic reaction networks. *Journal of The Royal Society Interface*, 15(143):20180079, 2018.

[189] Houssam Abbas, Andrew Winn, Georgios Fainekos, and A Agung Julius. Functional gradient descent method for metric temporal logic specifications. In *2014 American Control Conference*, pages 2312–2317. IEEE, 2014.

[190] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[191] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 211–220. ACM, 2010.

[192] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*, pages 167–170. Springer, 2010.

[193] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. Falsification of ltl safety properties in hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 368–382. Springer, 2009.

[194] Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott Smolka. On temporal logic and signal processing. In *International Symposium on Automated Technology for Verification and Analysis*, pages 92–106. Springer, 2012.

[195] Yang Cao, Daniel T Gillespie, and Linda R Petzold. The slow-scale stochastic simulation algorithm. *The Journal of chemical physics*, 122(1):014116, 2005.

[196] Friedrich C Simmel, Bernard Yurke, and Hari R Singh. Principles and applications of nucleic acid strand displacement reactions. *Chemical reviews*, 119(10):6326–6369, 2019.

APPENDIX A

Appendix A - Bellshape Generation File

In this part of the Appendix we present an SMT-ODE encoding of our bellshape `metasketch` outlined in Chapter 5 Section 5.1.4. The file is generated using our tool `CRNSketch` and is written for the `iSAT-ODE` solver.

```
DECL
define MAX_TIME = 1;
define SF = 100;
-- declare time variables
float [0, MAX_TIME] time;
float [0, MAX_TIME] delta_time;

-- declare cost variables
define MAX_COST = 1000;
define NO_COST_LIMIT = 0;

-- Define State Variables
float[0, 2] A;
float[0, 2] B;
float[0, 2] K;

-- Define Derivative Variables
float [-4, 4] dK_dt;

-- Lambda Variables
float [0, 1] lamA1;
float [0, 1] lamB1;
float [0, 1] lamA2;
float [0, 1] lamB2;

-- Choice Variables
float [0, 1] c7_1;
float [0, 1] c7_2;

float [0, 1] c2_0;
float [0, 1] c2_1;
float [0, 1] c2_2;
```

```
float [0, 1] c3_0;
float [0, 1] c3_1;
float [0, 1] c3_2;

float [0, 1] c1_0;
float [0, 1] c1_1;
float [0, 1] c1_2;


-- Joint choice Variables
float[0, 1] tc_1_0;
float[0, 1] tc_1_1;

-- Rate constants
float [0, 1] k_2;
float [0, 1] k_1;
float [0, 1] k_3;

--Define modes
boole mode_1;
boole mode_2;

INIT
time = 0;

-- cost condition
(((6*c1_1*lamA1 + 6*c1_1*lamB1 + 12*c1_2*lamA1 + 12*c1_2*lamB1 + 6*c2_1 + 12*c2_2
+ 5*c3_1 + 10*c3_2 + 5*c7_1*tc_1_1 + 10*c7_2*tc_1_1 + 5*lamA2*tc_1_0 +
5*lamB2*tc_1_0 + 31) <= MAX_COST) or (NO_COST_LIMIT = 1));

-- cannot be in two modes at the same time. We start in mode_1.
mode_1 = 1;
mode_1 + mode_2 = 1;


-- Integer encoding of lambda variables
((lamA1 = 1) and (lamB1 = 0)) or ((lamA1 = 0) and (lamB1 = 1));
((lamA2 = 1) and (lamB2 = 0)) or ((lamA2 = 0) and (lamB2 = 1));

-- Integer encoding of choice variables
 ((c7_1 = 1) and (c7_2 = 0))  or  ((c7_1 = 0) and (c7_2 = 1)) ;
 ((c2_0 = 1) and (c2_1 = 0) and (c2_2 = 0))  or  ((c2_0 = 0) and (c2_1 = 1) and (c2_2 = 0))
 or  ((c2_0 = 0) and (c2_1 = 0) and (c2_2 = 1)) ;
 ((c3_0 = 1) and (c3_1 = 0) and (c3_2 = 0))  or  ((c3_0 = 0) and (c3_1 = 1) and (c3_2 = 0))
 or  ((c3_0 = 0) and (c3_1 = 0) and (c3_2 = 1)) ;
 ((c1_0 = 1) and (c1_1 = 0) and (c1_2 = 0))  or  ((c1_0 = 0) and (c1_1 = 1) and (c1_2 = 0))
 or  ((c1_0 = 0) and (c1_1 = 0) and (c1_2 = 1)) ;

-- Integer encoding of optional reaction variables

-- Integer encoding of joint choice variables
((tc_1_0 = 1) and (tc_1_1 = 0)) or ((tc_1_0 = 0) and (tc_1_1 = 1));

-- Limits on initial conditions
(A = 0.5)
 and  (A >= 0) and  (A <= 2);
(B = 0.001)
 and  (B >= 0) and  (B <= 2);
(K = 0.001)
```

```
 and  (K >= 0) and  (K <= 2);


TRANS

-- time constraint
time' = time + delta_time;



-- invariant conditions during modes
mode_1  -> (dK_dt >= 0);
mode_1'  -> (dK_dt >= 0);
mode_2  -> (dK_dt <= 0);
mode_2'  -> (dK_dt <= 0);

-- jump conditions between modes
(mode_1 and mode_2') -> ((dK_dt = 0) and (K > 0.4));

-- No state change without time consumption.
(delta_time = 0) -> ((c7_1' = c7_1) and (c7_2' = c7_2));
(delta_time = 0) -> ((c2_0' = c2_0) and (c2_1' = c2_1) and (c2_2' = c2_2));
(delta_time = 0) -> ((c3_0' = c3_0) and (c3_1' = c3_1) and (c3_2' = c3_2));
(delta_time = 0) -> ((c1_0' = c1_0) and (c1_1' = c1_1) and (c1_2' = c1_2));
(delta_time = 0) -> ((c1_0' = c1_0) and (c1_1' = c1_1) and (c1_2' = c1_2));
(delta_time = 0) -> ((k_2' = k_2) and (k_1' = k_1) and (k_3' = k_3));
(delta_time = 0) -> ((lamA1' = lamA1) and (lamB1' = lamB1) and
(lamA2' = lamA2) and (lamB2' = lamB2));


-- Rate constants are fixed
(d.k_2/d.time = 0);
(k_2 = k_2');
(d.k_1/d.time = 0);
(k_1 = k_1');
(d.k_3/d.time = 0);
(k_3 = k_3');

-- Lambda variables are fixed
(d.lamA1/d.time = 0);
(d.lamB1/d.time = 0);
(lamA1 = lamA1') and (lamB1 = lamB1');
(d.lamA2/d.time = 0);
(d.lamB2/d.time = 0);
(lamA2 = lamA2') and (lamB2 = lamB2');

-- Choice variables are fixed
(d.c7_1/d.time = 0);
(d.c7_2/d.time = 0);
(c7_1 = c7_1') and (c7_2 = c7_2');
(d.c2_0/d.time = 0);
(d.c2_1/d.time = 0);
(d.c2_2/d.time = 0);
(c2_0 = c2_0') and (c2_1 = c2_1') and (c2_2 = c2_2');
(d.c3_0/d.time = 0);
(d.c3_1/d.time = 0);
(d.c3_2/d.time = 0);
(c3_0 = c3_0') and (c3_1 = c3_1') and (c3_2 = c3_2');
(d.c1_0/d.time = 0);
```

```
(d.c1_1/d.time = 0);
(d.c1_2/d.time = 0);
(c1_0 = c1_0') and (c1_1 = c1_1') and (c1_2 = c1_2');

-- Joint choice variables are fixed
(d.tc_1_0/d.time = 0);
(d.tc_1_1/d.time = 0);
(tc_1_0 = tc_1_0') and (tc_1_1 = tc_1_1');


-- Flows
(mode_1 or mode_2) ->  (d.K/d.time  =
SF*(-B*K*k_2 + k_1*(c1_0 + c1_1*(A*lamA1 + B*lamB1) + c1_2*(A*lamA1 + B*lamB1)^2)*
(K^2*c2_2 + K*c2_1 +
c2_0)*(-c2_1 - 2*c2_2 + c3_1 + 2*c3_2) + k_3*(c7_1*tc_1_1 + 2*c7_2*tc_1_1)));
(mode_1 or mode_2) ->  (d.A/d.time  =
SF*(k_1*(-c1_1*lamA1 - 2*c1_2*lamA1)*(c1_0 + c1_1*(A*lamA1 + B*lamB1) +
c1_2*(A*lamA1 + B*lamB1)^2)*(K^2*c2_2 + K*c2_1 + c2_0) + k_3*lamA2*tc_1_0));
(mode_1 or mode_2) ->  (dK_dt =
-SF*(B*K*k_2 + k_1*(c1_0 + c1_1*(A*lamA1 + B*lamB1) + c1_2*(A*lamA1 + B*lamB1)^2)*(K^2*c2_2 +
K*c2_1 + c2_0)*(c2_1 + 2*c2_2 - c3_1 - 2*c3_2) - k_3*tc_1_1*(c7_1 + 2*c7_2)));
(mode_1 or mode_2) -> (d.B/d.time  =
SF*(B*K*k_2 + k_1*(-c1_1*lamB1 - 2*c1_2*lamB1)*(c1_0 + c1_1*(A*lamA1 + B*lamB1) +
c1_2*(A*lamA1 + B*lamB1)^2)*(K^2*c2_2 + K*c2_1 + c2_0) + k_3*lamB2*tc_1_0));

TARGET
mode_2 and (time <= 1) ;
```

APPENDIX B

Appendix B - Example Solver Use and Output

Solver Use

Both `iSAT` and `dReach` are command line tools that come with a range of options on solving. While we will not explain all meta-parameters in this appendix, we will mention the most important ones. An example use of the iSAT file from the command line would be:

```
\$ ./isat-ode-r2806-static-x86_64-generic-noSSE-stripped.txt --i hill.hys
--ode-opts=--continue-after-not-reaching-horizon --prabs=0.001
--msw=0.02 --max-depth 2
```

where `--prabs` is the δ-precision of the candidate solution, `--msw` is the minimum splitting width used to control precision over time, i.e. when the solver checks to see if the ODEs satisfy the solution, and the $.hys$ file is our encoding of a problem. `--continue-after-not-reaching-horizon` is used to force the solver to continue even if the bracketing method [165] fails (it will then switch to another enclosure method). `--max-depth` refers to the number of unwindings of the bounded model checking algorithm within `iSAT`.

An example use of the `dReach` file from the command line would be:

```
\$ dreach -k 2  hillkinetics.drh --verbose
--precision 0.001 --visualize
```

where `k` refers to the number of unwindings of the bounded model checking algorithm, the `.drh` file is our encoding, `--verbose` refers to the amount of feedback we recieve on the status of the solver, `--precision` is the δ-precision and `--visualize` attempts to plot the results (although this often failed). `dReach` also produces certificates of correctness for both δ-sat and unsatisfiable solutions (a proof of unsatisfiability).

Solver Output

We present an example solver output for our phosphorelay model detailed in Chapter 6.4. The encoding is using the LNA semantics. This output comes from the synthesis of rate

parameters using the iSAT-ODE tool outlined in Chapter 3. Each variable is listed with its type and its value at each mode change. As per the specification given for a phosphorelay profile outlined in Chapter 6, the second derivative of $L3p$ denoted $L3ptt$ is 0 at the mode switch.

```
CANDIDATE SOLUTION:
mode_1 (boolean):
    @0:  [1,1] (point interval)
    @1:  [1,1] (point interval)
    @2:  [0,0] (point interval)

mode_2 (boolean):
    @0:  [0,0] (point interval)
    @1:  [0,0] (point interval)
    @2:  [1,1] (point interval)

L3p (float):
    @0:  [0,0] (point interval)
    @1:  [0.93779296,0.94667969] (width: 0.00888671875)
    @2:  [0.32216796,0.33105469] (width: 0.00888671875)

delta_time (float):
    @0:  [0.2734375,0.28027344] (width: 0.0068359375)
    @1:  [0.13769531,0.14453125] (width: 0.0068359375)

CL1L2p (float):
    @0:  [0,0] (point interval)
    @1:  [0.703125,0.7109375] (width: 0.0078125)
    @2:  [0.87109375,0.87695313] (width: 0.005859375)

CL3pB (float):
    @0:  [0,0] (point interval)
    @1:  [0.3984375,0.40625] (width: 0.0078125)
    @2:  [0.2265625,0.234375] (width: 0.0078125)

CL2pL3p (float):
    @0:  [0,0] (point interval)
    @1:  [0.33007812,0.3359375] (width: 0.005859375)
    @2:  [0.8828125,0.88867188] (width: 0.005859375)

CL2B (float):
    @0:  [0,0] (point interval)
    @1:  [0.8203125,0.82617188] (width: 0.005859375)
    @2:  [0.01757812,0.0234375] (width: 0.005859375)

CL3L1p (float):
    @0:  [0,0] (point interval)
    @1:  [0.15625,0.1640625] (width: 0.0078125)
```

```
    @2:  [0.61816406,0.62695313] (width: 0.0087890625)

CL1pL3p (float):
    @0:  [0,0] (point interval)
    @1:  [0.4609375,0.46875] (width: 0.0078125)
    @2:  [0.7265625,0.734375] (width: 0.0078125)

CL3B (float):
    @0:  [0,0] (point interval)
    @1:  [0.3359375,0.34375] (width: 0.0078125)
    @2:  [0.3046875,0.3125] (width: 0.0078125)

CL2L3p (float):
    @0:  [0,0] (point interval)
    @1:  [0.2890625,0.296875] (width: 0.0078125)
    @2:  [0.69140625,0.70117188] (width: 0.009765625)

CL1L3p (float):
    @0:  [0,0] (point interval)
    @1:  [0.8671875,0.875] (width: 0.0078125)
    @2:  [0.71191406,0.72070313] (width: 0.0087890625)

CL2pB (float):
    @0:  [0,0] (point interval)
    @1:  [0.46875,0.4765625] (width: 0.0078125)
    @2:  [0.46289062,0.47265625] (width: 0.009765625)

CL3p (float):
    @0:  [0,0] (point interval)
    @1:  [0.765625,0.7734375] (width: 0.0078125)
    @2:  [0.90917968,0.91748047] (width: 0.00830078125)

L3 (float):
    @0:  [0.32999999,0.33000001] (width: 5.55111512e-17)
    @1:  [0.4453125,0.453125] (width: 0.0078125)
    @2:  [0.19726562,0.20703126] (width: 0.009765625)

L2 (float):
    @0:  [0.32999999,0.33000001] (width: 5.55111512e-17)
    @1:  [0.48437499,0.4921875] (width: 0.0078125)
    @2:  [0.38671874,0.39648438] (width: 0.009765625)

L1 (float):
    @0:  [0.32999999,0.33000001] (width: 5.55111512e-17)
    @1:  [0.52343749,0.52929688] (width: 0.005859375)
    @2:  [0.10742187,0.11328125] (width: 0.005859375)

CL1p (float):
```

```
    @0:  [0,0] (point interval)
    @1:  [0.3125,0.3203125] (width: 0.0078125)
    @2:  [0.80566406,0.81445313] (width: 0.0087890625)

CL2p (float):
    @0:  [0,0] (point interval)
    @1:  [0.8359375,0.84277344] (width: 0.0068359375)
    @2:  [0.19824218,0.20507813] (width: 0.0068359375)

L2p (float):
    @0:  [0,0] (point interval)
    @1:  [0.15625,0.1640625] (width: 0.0078125)
    @2:  [0.6875,0.6953125] (width: 0.0078125)

CL1B (float):
    @0:  [0,0] (point interval)
    @1:  [0.0703125,0.078125] (width: 0.0078125)
    @2:  [0.42578125,0.43554688] (width: 0.009765625)

L3pt (float):
    @0:  [0,0] (point interval)
    @1:  [0.171875,0.1796875] (width: 0.0078125)
    @2:  [0.8671875,0.875] (width: 0.0078125)

time (float):
    @0:  [0,0] (point interval)
    @1:  [0.2734375,0.28027344] (width: 0.0068359375)
    @2:  [0.41210937,0.41796875] (width: 0.005859375)

CL3L2p (float):
    @0:  [0,0] (point interval)
    @1:  [0.890625,0.8984375] (width: 0.0078125)
    @2:  [0.82226562,0.82910157] (width: 0.0068359375)

CL1pL2p (float):
    @0:  [0,0] (point interval)
    @1:  [0.1640625,0.171875] (width: 0.0078125)
    @2:  [0.92382812,0.93359375] (width: 0.009765625)

k2 (float):
    @0:  [0.46249999,0.46875] (width: 0.00625)
    @1:  [0.46249999,0.46875] (width: 0.00625)
    @2:  [0.46249999,0.46875] (width: 0.00625)

k3 (float):
    @0:  [0.51874999,0.525] (width: 0.00625)
    @1:  [0.51874999,0.525] (width: 0.00625)
    @2:  [0.51874999,0.525] (width: 0.00625)
```

```
k1 (float):
    @0:  [0.41874999,0.425] (width: 0.00625)
    @1:  [0.41874999,0.425] (width: 0.00625)
    @2:  [0.41874999,0.425] (width: 0.00625)

CL2L3 (float):
    @0:  [0,0] (point interval)
    @1:  [0.875,0.8828125] (width: 0.0078125)
    @2:  [0.55566406,0.56445313] (width: 0.0087890625)

CL1pB (float):
    @0:  [0,0] (point interval)
    @1:  [0.71875,0.7265625] (width: 0.0078125)
    @2:  [0.9375,0.9453125] (width: 0.0078125)

CL3L3p (float):
    @0:  [0,0] (point interval)
    @1:  [0.796875,0.8046875] (width: 0.0078125)
    @2:  [0.24707031,0.25585938] (width: 0.0087890625)

k4 (float):
    @0:  [0.159375,0.16875001] (width: 0.009375)
    @1:  [0.159375,0.16875001] (width: 0.009375)
    @2:  [0.159375,0.16875001] (width: 0.009375)

CL1L1p (float):
    @0:  [0,0] (point interval)
    @1:  [0.90722656,0.9140625] (width: 0.0068359375)
    @2:  [0.34472656,0.3515625] (width: 0.0068359375)

CL2L2p (float):
    @0:  [0,0] (point interval)
    @1:  [0.859375,0.8671875] (width: 0.0078125)
    @2:  [0.43701171,0.4453125] (width: 0.00830078125)

CL1L3 (float):
    @0:  [0,0] (point interval)
    @1:  [0.828125,0.8359375] (width: 0.0078125)
    @2:  [0.1328125,0.140625] (width: 0.0078125)

L1p (float):
    @0:  [0,0] (point interval)
    @1:  [0.67773437,0.68359375] (width: 0.005859375)
    @2:  [0.47070312,0.4765625] (width: 0.005859375)

CL1L2 (float):
    @0:  [0,0] (point interval)
```

```
        @1:  [0.95507812,0.9609375] (width: 0.005859375)
        @2:  [0.39599609,0.40332032] (width: 0.00732421875)

    CL2 (float):
        @0:  [0,0] (point interval)
        @1:  [0.3359375,0.34375] (width: 0.0078125)
        @2:  [0.8046875,0.8125] (width: 0.0078125)

    CL3 (float):
        @0:  [0,0] (point interval)
        @1:  [0.4140625,0.421875] (width: 0.0078125)
        @2:  [0.359375,0.3671875] (width: 0.0078125)

    B (float):
        @0:  [0,0] (point interval)
        @1:  [0.1484375,0.15625] (width: 0.0078125)
        @2:  [0.484375,0.4921875] (width: 0.0078125)

    L3ptt (float):
        @0:  [0,0] (point interval)
        @1:  [0,0] (point interval)
        @2:  [0.4296875,0.4375] (width: 0.0078125)

    CL1 (float):
        @0:  [0,0] (point interval)
        @1:  [0.953125,0.95947266] (width: 0.00634765625)
        @2:  [0.17480468,0.18115235] (width: 0.00634765625)

    CB (float):
        @0:  [0,0] (point interval)
        @1:  [0.359375,0.36621094] (width: 0.0068359375)
        @2:  [0.22167968,0.22851563] (width: 0.0068359375)

    CL2L1p (float):
        @0:  [0,0] (point interval)
        @1:  [0.1875,0.19335938] (width: 0.005859375)
        @2:  [0.36816406,0.37695313] (width: 0.0087890625)


Statistics.
Number of variables                    : 1197
    Boolean variables                  : 460
    Integer variables                  : 513
    Real variables                     : 224
Number of complex bounds               : 348
Number of problem clauses              : 2280
Number of decisions (current / total)  : 565 / 565
Variables preferred for splitting      : (none)
```

```
Number of point splits                : 0 / 0
Number of assignments                 : 5450 / 7171
    Number of IQ additions            : 5379 / 7099
Maximum decision level                : 565 / 565
Maximum number of deductions per DL   : 4028 / 4028
Number of conflicts                   : 0 / 2
Average size of learnt clauses        : 0 / 0
Maximum backjump distance             : 0 / 0
Number of restarts                    : 0 / 0
Number of conflict clauses
implied by conflict clauses           : 0 / 0

Time solver                           : 39047.32 / 39047.33 sec
```

APPENDIX C

Appendix C - CRN PID Controller

We present our full Chemical Reaction Network PID feedback loop with gene expression plant and two reference signals introduced in Section 7.3. We include the three actuation mechanisms given with the plant. For each block we also give the initial conditions used to produce the simulations seen in Figure 7.4. Please note that rates denoted with multiple letters are the products of those individual letters, i.e. a rate rs is the product of two rates $r \times s$

PID Controller

First we report the reactions and parameters of the CRN PID controller of which the proof of correctness and details of operation are outlined in Chapter 7. For all figures we used the same parameters

Proportional Block:

$$E^+ \xrightarrow{rs} E^+ + P^+ \qquad E^- \xrightarrow{rs} E^- + P^-$$
$$P^+ + P^- \xrightarrow{q} \varnothing \qquad P^+ \xrightarrow{s} \varnothing$$
$$P^- \xrightarrow{s} \varnothing$$

Initial Conditions: (rates) $s, rs, q = 1$ (species) P^+,$P^- = 0$

Integral Block:

$$E^+ \xrightarrow{k} E^+ + I^+ \qquad E^- \xrightarrow{k} E^- + I^- \qquad I^+ + I^- \xrightarrow{q} \varnothing$$

Initial Conditions: (rates) $k, q = 1$, (species) I^+, $I^- = 0$

Derivative Block:

$$E^+ \xrightarrow{rv} E^+ + A^+ \qquad E^- \xrightarrow{rv} E^- + A^-$$
$$E^+ \xrightarrow{rvs} E^+ + D^+ \qquad E^- \xrightarrow{rvs} E^- + D^-$$
$$A^+ \xrightarrow{v} \varnothing \qquad A^- \xrightarrow{v} \varnothing$$
$$A^+ \xrightarrow{vs} A^+ + D^- \qquad A^- \xrightarrow{vs} A^- + D^+$$
$$D^+ \xrightarrow{s} \varnothing \qquad D^- \xrightarrow{s} \varnothing \qquad D^+ + D^- \xrightarrow{q} \varnothing$$

Initial Conditions: (rates) v, s, vs = 1, rv, q, rvs = 10 , (species) A^+,A^-,D^+,D^- = 0

C.0.1 Summation and Subtraction Blocks

We provide the CRNs for the two summation blocks which are highlighted in the green block in Figure 7.3. The first adds the output of the P and I blocks together. The second the PI and D blocks together which produces the output species of the controller U (given as PID in the CRNs below). The rates for the summation blocks were tuned using a trial and error approach where the rates were adjusted to reduce error and reduce time to a steady-state. It is important to note that no set of parameters are best in all situations. The rates are within bounds for implementation as a DNA Strand Displacement device [196].

Addition Block $P + I$:

$$P^+ \xrightarrow{s} P^+ + B^+ \qquad I^+ \xrightarrow{s} I^+ + B^+$$
$$P^- \xrightarrow{s} P^- + B^- \qquad I^- \xrightarrow{s} I^- + B^-$$
$$B^+ + B^- \xrightarrow{q} \varnothing \qquad B^+ \xrightarrow{s} \varnothing$$
$$B^- \xrightarrow{s} \varnothing$$

Initial Conditions for $P + I$ summation block: (rates) s = 0.8,q = 0.3, (species) B^+,B^- = 0

Addition Block $PI + D$:

$$PI^+ \xrightarrow{s} PI^+ + PID^+ \qquad D^+ \xrightarrow{s} D^+ + PID^+$$
$$PI^- \xrightarrow{s} PI^- + PID^- \qquad D^- \xrightarrow{s} D^- + PID^-$$
$$PID^+ + PID^- \xrightarrow{q} \varnothing \qquad PID^+ \xrightarrow{s} \varnothing$$
$$PID^- \xrightarrow{s} \varnothing$$

Initial Paramterisation for $PI + D$ summation block: (rates) s = 1.1,q = 0.1 (species) PID^+,PID^- = 0

Subtraction Block:
The subtraction block is used to compute the error of the output of the plant Y with the reference signal:

$$Y'^{+} \xrightarrow{s} Y'^{+} + E^{+} \qquad R^{-} \xrightarrow{s} R^{-} + E^{+}$$
$$Y'^{-} \xrightarrow{s} Y'^{-} + E^{-} \qquad R^{+} \xrightarrow{s} R^{+} + E^{-}$$
$$E^{+} + E^{-} \xrightarrow{q} \varnothing \qquad E^{+} \xrightarrow{s} \varnothing$$
$$E^{-} \xrightarrow{s} \varnothing$$

Initial Conditions for difference block summation block: (rates) $s, q = 1$

C.0.2 Reference Signals

Next we introduce the intitial conditions and reactions for both the constant and sine wave reference signals. These act as a reference which we are trying to control our plant to track.

Constant:
The constant signal can be given simply by stating a non-decaying species with a molecular count equal to the constant signal however it can also be given by the following CRN which is stated in Figure 7.4.

$$\varnothing \xrightarrow{k} R^{+} \qquad R^{+} \xrightarrow{r} \varnothing$$

Initial Conditions: (rates) $k = 10, r = 1$ (species) $R^{+} = 0$, $R^{-} = 0$

Sine wave: The sine wave has a slow reaction rate to allow for the PID controller to properly track the signal.

$$A^{+} \xrightarrow{k} A^{+} + R^{+} \qquad A^{-} \xrightarrow{k} A^{-} + R^{-}$$
$$R^{+} \xrightarrow{k} R^{+} + A^{-} \qquad R^{-} \xrightarrow{k} R^{-} + A^{+}$$
$$A^{+} + A^{-} \xrightarrow{k} \varnothing \qquad R^{+} + R^{-} \xrightarrow{k} \varnothing$$

Initial Conditions: (rates) $k = 0.01$ (species) $A^{+} = 10$, $A^{-} = 0$, $R^{+} = 0$, $R^{-} = 0$

Plant and Actuators

We introduce the gene expression plant used within our model. We also include the three actuations methods from the controller U^{+}, U^{-} seen in Figure 7.4 used to interface with the plant.

Actuator 1:

$$U^+ \xrightarrow{k} U^+ + mRNA \qquad\qquad U^- \xrightarrow{k} U^- + microRNA$$

Initial Conditions: (rates) $k = 1$ (species) $mRNA = 0, microRNA = 0, U^+ = 0.5$, $U^- = 0.5$ where the initial values of U^+, U^- are arbitrarily set.

Actuator 2:

$$U^+ \xrightarrow{k} U^+ + mRNA \qquad\qquad U^- + mRNA \xrightarrow{k} U^-$$

Initial Conditions: (rates) $k = 1$ (species) $mRNA = 0, U^+ = 0.5$, $U^- 0.5$

Actuator 3:

$$U^+ \xrightarrow{k} U^+ + Pro \qquad\qquad U^- + Pro \xrightarrow{k} U^-$$

Initial Conditions: (rates) $k = 1$ (species) $Pro = 1, U^+ = 0.5, U^- 0.5$

Plant:

$$\xrightarrow{k} mRNA \qquad\qquad mRNA \xrightarrow{k} \varnothing$$
$$mRNA^+ \xrightarrow{k} mRNA + Pro \qquad\qquad Pro \xrightarrow{k} \varnothing$$
$$mRNA + microRNA \xrightarrow{k} \varnothing$$

Initial Conditions: (rates) $k = 1$, (species) $mRNA$, $microRNA = 0$, $Pro = 1$

C.0.3 Single to Dual Rail Block

We introduce the block which takes the output of the plant Y and transforms into into a dual rail signal, see blue block Figure 7.3.

$$Y \xrightarrow{s} Y + Y'^+ \qquad\qquad Y'^+ \xrightarrow{s} \varnothing \qquad\qquad Y'^+ + Y'^- \xrightarrow{q} \varnothing$$

Initial Conditions: (rates) $s, q = 1$, (species) Y'^+,$Y'^- = 0$