Formal design and analysis for DNA implementations of Chemical Reaction Networks

Thesis by
Robert F. Johnson

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2020
Defended January 21, 2020

ACKNOWLEDGEMENTS

There are many people I am grateful to for helping me become the researcher represented in this thesis, in more ways than I can effectively express. I can, however summarize. My advisors, Erik Winfree and Lulu Qian, I would like to thank for introducing me to the field of molecular programming, for showing me the many little skills that make the difference between a problem-solver and a proper researcher, and most of all, for being patient and supportive over the past 6 years while I was learning and practicing said skills. Professors Ellen Rothenberg and Shuki Bruck, whose classes I took in my undergraduate first year, I would like to thank for introducing me to the idea of biology as a logical system, which started the path that eventually led me here. Professors Richard Murray and Niles Pierce, the members of my thesis committee in addition to the four above, I would like to thank for all the advice, and also for showing me the breadth of molecular programming when I was starting out. Finally my mother, Devra Johnson, I would like to thank for teaching me the concepts of math and logic, and for finding opportunities for me to practice those in my high school years, laying the foundations for the direction I would eventually go in.

ABSTRACT

In molecular programming, the Chemical Reaction Network model is often used to describe systems of interacting molecules. This model can describe either real systems, allowing us to analyze and determine their computational function; or describe hypothetical systems, with known computational function but perhaps no known physical example. One significant breakthrough in the field is that any Chemical Reaction Network can be approximated by a system using DNA Strand Displacement mechanisms. This allows the Chemical Reaction Network model to be treated like a programming language, where programs can be written in the abstract and then compiled into physical molecules. Given a programming language and a proof-of-concept compiler, one would want to take the compiler from the proof-of-concept stage into a more reliable, more systematic, and better understood process. This thesis is made up of my contributions to that effort.

First, given a programming language and a compiler, it would be useful to formally verify that the compiler is correct. My collaborators, Qing Dong and Erik Winfree, and I defined a Chemical Reaction Network-specific form of bisimulation equivalence, which can compare two such networks and verify that one is (or is not) a correct implementation of the other. For example, the compiler-produced DNA circuit can be verified as an implementation of its abstract program, although this is not the only possible use. After defining this concept of equivalence, we show that it can be checked by algorithm; although various parts of the problem are NP-complete or PSPACE-complete, we give algorithms that meet these lower bounds. We also prove a number of interesting properties of Chemical Reaction Network bisimulation equivalence, including transitivity and modularity properties which are particularly useful for stepwise checking of large systems. We further extend this bisimulation method to linear Polymer Reaction Networks, a strictly more powerful abstraction which has been occasionally used in molecular programming. Again we prove complexity hardness results, which in this case are as expected uncomputable in the general case; however, many practical systems can still be verified, and we give one such example. Finally, we use bisimulation to identify a class of *single-locus networks* that are practical to implement. Thus we show a method of verification which can simplify use of the above-mentioned compiler by proving general statements of correctness about its results.

Second, given a programming language and a concept of compiling it, it would be useful to optimize the result of the compilation. One particular area of optimization is the number of DNA strands per prepared complex; some experiments suggest that systems with no more than 2 strands per complex are more robust. Lulu Qian and I developed some proposed DNA Strand Displacement schemes for general Chemical Reaction Network implementations with no more than 2 strands per complex, and a number of other desirable properties. Meanwhile, having been shown to be useful for many reasons, the mechanisms of DNA Strand Displacement have recently been formalized, abstracted, and analyzed. I show that this formalization, combined with the bisimulation methods above, can prove various statements about the limits of DNA Strand Displacement systems. For example, a set of desirable conditions including the 2-strand limit cannot be achieved by any general Chemical Reaction Network implementation scheme. I also observe that two of the new schemes we discovered, each meeting all but one condition of the impossible set, were found in the process of coming up with this proof. I thus argue that through formalization of DNA Strand Displacement we can have a more systematic method of finding and designing molecular programs, and of knowing when the programs we want do not exist.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] Robert F. Johnson. Impossibility of sufficiently simple chemical reaction network implementations in DNA strand displacement. In Ian McQuillan and Shinnosuke Seki, editors, *Unconventional Computation and Natural Computation*, pages 136–149. Springer International Publishing, 2019. ISBN 978-3-030-19311-9. doi: 10.1007/978-3-030-19311-9_12.
Contributions: RFJ formulated the question of impossibility, proved the theorems, and wrote the manuscript.

[2] Robert F. Johnson, Qing Dong, and Erik Winfree. Verifying chemical reaction network implementations: A bisimulation approach. *Theoretical Computer Science*, 2018. doi: 10.1016/j.tcs.2018.01.002.
Contributions: EW formulated the initial questions and definitions of CRN bisimulation, provided advice and assistance throughout, and reviewed and edited the manuscript. QD gave the initial formulation and proof of the equivalence of three notions theorem and the reactionsearch algorithm, contributed an initial manuscript describing those parts, and reviewed the final manuscript. RFJ improved existing definitions and proofs, proved the transitivity and modularity theorems, designed the loopsearch and graphsearch algorithms, proved the completeness results, defined and analyzed the extension for implicit catalysts, and was the primary writer of the final manuscript.

TABLE OF CONTENTS

Acknowledgements iii
Abstract iv
Published Content and Contributions vi
Table of Contents vii
List of Illustrations ix
List of Tables xi
List of Algorithms xii
Chapter I: Introduction 1
1.1 The promise of molecular programming 1
1.2 Theory in molecular programming 2
1.3 A type of molecular compiler 2
1.4 Formal verification 3
1.5 Exploring formal DNA strand displacement 4
Chapter II: Verifying Chemical Reaction Network implementations: A bisimulation approach 5
2.1 Perspective 5
2.2 Background 8
2.3 The Chemical Reaction Network Model 16
2.4 The Meaning of Correctness 19
2.5 Checking Bisimulation 41
2.6 Additional Features of CRN Bisimulation 88
2.7 Discussion 98
Chapter III: Verifying Polymer Reaction Networks using bisimulation . 103
3.1 Perspective 103
3.2 Introduction 107
3.3 Definitions 110
3.4 Verifying the DNA Stack Machine 130
3.5 Hardness Results 143
3.6 Single-Locus Networks 146
3.7 Alternate polymer models and extended models 156
3.8 Discussion 160
Chapter IV: Simplifying Chemical Reaction Network implementations with two-stranded DNA building blocks 164
4.1 Perspective 164
4.2 Introduction 166
4.3 Two-stranded motifs 167
4.4 Chemical Reaction Network implementations 170
4.5 Correctness of the schemes 174
4.6 Discussion 175

Chapter V: Impossibility of sufficiently simple Chemical Reaction Network implementations in DNA strand displacement 178
5.1 Perspective 178
5.2 Introduction 181
5.3 Formalizing DNA Strand Displacement 183
5.4 The 2-r4 Condensed Reaction 188
5.5 Chemical Reaction Network Implementations 192
5.6 Discussion 196
5.7 Omitted Proofs 197
Bibliography 200

LIST OF ILLUSTRATIONS

Number *Page*
2.1 Implementation of $A + B \rightarrow C + D$ using the scheme described in [66]. 20
2.2 Interpretation of the implementation CRN in Figure 2.1. 21
2.3 Example CRNs with the same set of (interpreted) trajectories but different behavior. 23
2.4 Modified version of the rock-paper-scissors oscillator [25, 44] and an incorrect implementation. 24
2.5 The translation scheme from [57], when used as a general CRN implementation, violates the delimiting condition. 30
2.6 An example implementation CRN with multiple possible correct interpretations as different formal CRNs. 31
2.7 An implementation CRN that satisfies the modularity condition. 39
2.8 Example set of minimal implementation states for the reaction $A + B \rightarrow C$. 43
2.9 Example use of minimal states to reduce the dependence of the permissive condition on null species. 46
2.10 Example graphs of minimal states. 48
2.11 Example of the loopsearch graph for the formal reaction $A+B \rightarrow C$ with implementation CRN from Figure 2.10B. 50
2.12 An implementation CRN with a correct interpretation if and only if the corresponding space-bounded Turing machine accepts. 63
2.13 A pictorial illustration of the search tree explored by the reactionsearch algorithm for the given pair of formal and implementation CRNs. 71
2.14 Example formal and implementation CRN corresponding to an instance of the 3-SAT problem. 80
2.15 An example spurious reaction in a variant of the translation scheme by Soloveichik et al. [66]. 94
3.1 Scaling of number of states in well-mixed versus polymer systems.104
3.2 An example implementation CRN with interpretation. 114
3.3 Illustrated definition of a Polymer Reaction Network. 117

3.4 Example Polymer Reaction Networks to demonstrate various features of the PRN model. 122
3.5 Features of a polymer interpretation. 127
3.6 Choice of monomers in Qian et al.'s DNA stack machine [57]. 133
3.7 Enumeration of reaction schemata and interpretation of the monomers in Qian et al.'s DNA stack machine [57] as enumerated in Figure 3.6. 136
3.8 Examples of the three types of modules. 139
3.9 The "minimal states argument" from our previous work on CRN bisimulation [40] is often the most effective way to prove the permissive condition. 141
3.10 Intuitive illustration of single-locus reaction schemata. 148
4.1 Four previously studied reversible 2-stranded DSD motifs, shown through common examples. 168
4.2 A cooperative 4-way branch migration mechanism. 171
5.1 The reaction rules of Definition 5.3.2. 185
5.2 A locality theorem. 187
5.3 The two-toehold-mediated reversible 4-way branch migration (2-r4) condensed reaction mechanism. 189
5.4 An example $O(1)$ toeholds systematic DSD implementation. . . 193
5.5 An example information-bearing set. 195

LIST OF TABLES

Number *Page*

3.1 The transition rules of the Turing machine whose polymer implementation is shown in Figure 3.4. 124

4.1 List of species for the 4-way $O(n)$-toeholds scheme. 173

LIST OF ALGORITHMS

2.1 The loopsearch algorithm to check the permissive condition in polynomial space. 57

2.2 The graphsearch algorithm to check the permissive condition in time and space polynomial in the number of minimal states. 59

2.3 The reactionsearch algorithm to complete a partial interpretation or assert that no completion exists, in polynomial space. 68

Chapter 1

INTRODUCTION

1.1 The promise of molecular programming

The works I will present in this thesis are all within the field of molecular programming. If forced to summarize in one sentence, I would describe molecular programming as taking the abstract definition of “computer” and studying how the behavior of molecules can satisfy it. Often the molecules in question are biological—DNA, RNA, proteins, and others. The belief of molecular programming is that if the behavior of these molecules satisfies the same abstractions as a computer, then it can be understood and programmed in the same sense as a computer can. The promise of molecular programming is the ability to understand and control the way molecules compute, in order to add another dimension to our understanding of what exactly computation is, and to build practical devices that can solve complex computational problems with likely different strengths and weaknesses than more typical modern computers. What form this will take is not yet known; some promising properties of molecules compared to modern computers are their hyper-parallelism and their potential biocompatibility.

Currently, the field has produced a number of interesting molecular devices. As a survey of examples: DNA tiles with programmed interactions can count in binary [30], construct uniquely-addressed shapes [79], and execute 6-bit layered circuits [82]. DNA origami can fold large DNA scaffolds into detailed shapes [59], including hundred-nanometer-scale objects with nanometer-scale addressability [73], walkers [62, 75] with the potential for tasks such as sorting objects [71], and lockboxes that release a chemical in response to a specific signal [27]. Dynamic DNA technologies such as DNA strand displacement can implement large boolean circuits [55, 72] and simple abstract Chemical Reaction Networks such as majority determination [19] and controlled oscillations [67]. Of course, there are many other interesting molecular devices demonstrated in the past few decades.

1.2 Theory in molecular programming

Behind all of these devices lies a rich theoretical structure, generalizing classes of molecular programs and connecting them to classical computation. DNA tile assembly results are based on an abstract Tile Assembly Model, which was proven to be Turing universal [80], the theory of which has since been significantly expanded. The Chemical Reaction Network model describes abstract chemical systems, and serves as the foundation for much of the dynamic DNA nanotechnology devices. All of the results in this thesis are related to the Chemical Reaction Network model or an expansion thereof.

The Chemical Reaction Network (CRN) model is of some types of abstract chemical species in a well-mixed solution with specified reactions; a technical definition will be given in future chapters. It serves as a language to describe existing chemical and biochemical systems, to design new systems, and to reason about the capabilities of systems that fit within its paradigm. (The CRN syntax, a list of species and reactions, can be interpreted as either large numbers of molecules interacting when measured as concentration of molecules per volume, or as small numbers of molecules interacting measured by individual molecules. These are often called the *deterministic* and *stochastic* models of CRNs; my work, and all of this discussion, refers to the stochastic model.) Examples of interesting CRNs include small CRNs for approximate majority [2] and sustained oscillations [25, 67]. Larger CRNs can simulate space-bounded Turing machines without error [41, 69] or unbounded Turing machines with small probability of error [65]. There are a number of useful results about the computational power of (stochastic) CRNs: for example, that their reachability problem is decidable [47], implying that CRNs with no errors allowed are less powerful than Turing machines, and in particular they can compute exactly the semilinear functions [1, 16, 26].

1.3 A type of molecular compiler

While the CRN language is abstract, it turns out that any CRN can be approximated by DNA molecules using DNA Strand Displacement (DSD) mechanisms. A number of implementation schemes have been constructed; some I consider noteworthy are the first proof of concept by Soloveichik et al. [66], a scheme used as part of a stack machine by Qian et al. [57], and a compact two-domain scheme by Cardelli [6]. In some sense, these schemes can be considered a compiler for CRNs as a programming language; in a more literal sense, Shin [63] followed up by Badelt et al. [4] have worked on the Nuskell program, which takes a CRN and a translation scheme and outputs the compiled DSD circuit. These implementation schemes promise to take the abstract CRNs described above and turn them into real devices, and to some extent this promise has been fulfilled: Chen et al. have demonstrated an approximate majority CRN using Cardelli's scheme [19], and Srinivas et al. have demonstrated a 3-species oscillator using Soloveichik et al.'s scheme [67]. The next step, then, is to make these schemes work reliably for CRNs large enough to compute more complex functions and tasks.

1.4 Formal verification

To make this compiler work more reliably for larger systems, we would like a solid theoretical understanding of how it works. One part of this understanding is formal verification of the compiler: a proof that the DSD system output by the compiler is correctly approximating the desired CRN. Lakin et al. have developed a serializability method [46], and Shin et al. a pathway decomposition method [64], to answer this question. In this thesis I discuss a method, a CRN-specific adaptation of bisimulation as used in concurrency theory [52], that checks whether one CRN is a correct implementation of another CRN. Because DSD systems (without infinite polymerization) can be described as CRNs, this method is particularly useful for verifying that the DSD system produced by a compiler correctly implements the intended CRN, although this is not the only use case. The Nuskell compiler mentioned above, after compiling a CRN, can verify the result using either pathway decomposition or CRN bisimulation [4].

Chapter 2 covers my work with Qing Dong and Erik Winfree on the bisimulation method applied to CRNs. CRN bisimulation is, informally, based on an *interpretation* of each implementation species as a bunch of formal species, then asking if the two systems are equivalent up to that interpretation. We define CRN bisimulation equivalence, prove properties such as transitivity and modularity, show that CRN bisimulation can be checked algorithmically but is NP- or PSPACE-complete depending on the assumptions made, and give algorithms that meet those lower bounds. Chapter 3 covers my work with Erik Winfree expanding this concept to linear Polymer Reaction Networks (PRNs). We extend the interpretation used in CRN bisimulation so that the interpretation of a polymer is made up of the interpretations of its monomers and show that CRN bisimulation can be adapted to PRNs; we show that PRN bisimulation is undecidable in the general case, but can be used to verify some practical implementations; and we use PRN bisimulation to identify a class of *single-locus PRNs* that is promising for DNA implementation. All of this sets up bisimulation as a theoretical foundation with many uses for the CRN compiler, including the obvious use of verifying a given implementation; the use of transitivity and modularity to prove that an implementation scheme will always produce correct implementations; or, as in the single-locus case, as a guideline to use when searching for new correct implementation schemes.

1.5 Exploring formal DNA strand displacement

To make the CRN-to-DSD compiler(s) work more reliably for larger systems, we would like a solid theoretical understanding of how they work. The other side of this understanding is modeling the behavior of DSD systems. To start with, models such as Visual DSD [45, 53] and Peppercorn [36] formally define the known strand displacement reactions and, given a set of DNA complexes, will enumerate the reactions they can undergo (to the best of our knowledge) as a CRN. Peppercorn in particular is used by Nuskell when verifying its results [4]. This allows us to understand the systems we design, and is necessary for the above verification work.

A belief I hold about this field is that we can use this theory not only to analyze systems already designed, but to guide the design of new systems. This thesis contains three results I feel are examples of this belief. First, the single-locus PRN result mentioned in Chapter 3 involves using knowledge of bisimulation to identify an interesting class of PRN. Second, I have been trying to use the formalizations of DSD to investigate its limits; in particular, what are the simplest implementations of CRNs, and whether implementations with certain desirable properties are in fact impossible. One such desirable property is using only 2-stranded input complexes; in the process of this investigation, Lulu Qian and I found new 2-stranded mechanisms that can be used for CRN implementations, which I present in Chapter 4. Third, Chapter 5 presents the impossibility result itself, or at least a preliminary result showing one of the limits of DNA Strand Displacement. I believe that following up on this line of investigation will produce a new understanding of DNA Strand Displacement, possibly allowing systematic design of DSD circuits or otherwise easier ways of writing molecular programs.

Chapter 2

VERIFYING CHEMICAL REACTION NETWORK IMPLEMENTATIONS: A BISIMULATION APPROACH

2.1 Perspective

I have so far mentioned Chemical Reaction Networks (CRNs) as a molecular programming language, DNA strand displacement (DSD) as a toolbox of molecular mechanisms, and the ability to compile CRNs to DSD systems. In this chapter I discuss a method to check whether the output of such a compiler will correctly implement its input.

Why do we need such a method? There's a straightforward reason: some of our DSD implementation schemes might be wrong. A scheme might be wrong just from basic human error by its designers, or the problem might be a more subtle one, failing to do something that the designers didn't realize was necessary. Moreover, more complex tasks require larger and more complex CRNs and implementations, and thus makes it more likely for errors in the implementations to occur and/or be undetected. When we want molecular programs to do practical tasks, we need larger and larger programs, and a formal—ideally, computer-checkable—verification method becomes essential. In addition, having a definition of "correct" implementation allows us to prove that a correct implementation must or can't have certain properties, which I expand on in chapters 4 and 5.

So what exactly is formal verification? We mentioned that some systems have errors; formal verification is a process for finding those errors if they exist. More specifically, formal verification is a process that, given a specification and an implementation, says either "correct" or "incorrect". Ideally, if the process says "correct" it implies a correspondence between the implementation and the specification, and if the process says "incorrect" it identifies an error in the implementation. For example, a correspondence between CRNs might mean that any state reachable in the specification is reachable in the implementation, and vice versa. Stronger guarantees are both possible and desirable. Similar questions have been extensively explored in (non-molecular) computer science, and part of the contribution of this work is the adaptation of those concepts to molecular systems. In particular, simulation and bisimulation methods are a well-studied approach for verifying a correspondence between two systems [52], and the method discussed in this chapter is a type of bisimulation.

Our CRN bisimulation method is one method of verifying CRN implementations, but not the only one. Other methods include pathway decomposition [64] and serializability analysis [46]. Each of these methods is based on a different idea: CRN bisimulation is based on a correspondence between specification and implementation species; pathway decomposition is based on a correspondence between sequences of reactions in the specification and in the implementation; and serializability analysis is based on re-ordering sequences of reactions in the implementation to a standard form. People familiar with the three methods could argue which ones are more or less suited for various situations; for example, we know a number of implementation features that pathway decomposition analyzes properly while CRN bisimulation fails on, and vice versa. However, to my knowledge no thorough comparison of these methods has been done.

Given all the above motivation, what exactly did we do in this paper? In one sentence, we defined CRN bisimulation and discussed how to use it. CRN bisimulation is a formal verification method based on an "interpretation" of species in the implementation CRN as sets of species in the specification CRN. If that interpretation has certain properties, then (as we prove) the rate-independent behaviors of the two CRNs will be equivalent; while not the end-all be-all of equivalence, it is a desirable property. We then go on to discuss various properties of CRN bisimulation, many of which have implications for its theoretical nature and/or practical use. A reader interested in analyzing implementations by hand, for example, might want to look at the examples of bisimulation, and the transitivity and modularity properties. One interested in computational checking of bisimulation might focus on the algorithms for doing so, and hardness results that prove optimality of said algorithms. For those familiar with bisimulation in non-molecular systems, we discuss how the structure of CRNs gives rise to a natural subclass of bisimulation relations, which are more restricted but in some ways more powerful.

With all this, how can CRN bisimulation be used, and how does it fit into the big picture of the field? What I argue is that there is a larger project, of constructing a high-level programming language for molecules, of which CRN bisimulation is a necessary step. We can write and have written meaningful programs in the CRN model, including approximate majority functions [2], oscillators [25, 44], and Turing machines [65]. However, the CRN model is more of a low-level description of what molecules do than a high-level programming language with powerful programming features. In my ten-year plans I imagine such a language built out of pieces each of which can be compiled to a CRN, such that powerful programs can be written in that language, compiled reliably into CRNs, and those CRNs can be compiled reliably into DSD circuits and further to DNA molecules which physically execute the intended program. A few examples of such a language have been proposed, such as CRN++ [76]; none are in wide use. CRN bisimulation is then the verification step—the "reliably"—of the CRN-to-DSD compilation, and is meant to build a foundation for a future higher-level language.

More concretely, CRN bisimulation is currently part of the Nuskell verifying compiler, a program that uses any of the various translation schemes to convert a CRN to a DSD system then verifies that the result is correct [4]. CRN bisimulation and Nuskell are both part of the larger ecosystem built up around formalizing DSD and compiling CRNs. In the CRN-to-DSD layer, we have reaction enumerators such as Visual DSD [45] and Peppercorn [36], programs that given a set of DNA complexes will enumerate what reactions are possible according to the DSD model; and further work done on formalizing this model [53]. Even more work has been done on the DSD-to-physical-DNA layer, in finding actual DNA sequences that have the correct abstract behavior of the "domain-level" DSD model. Relevant software packages include Nupack, for analysis and design of short DNA strands' static behaviors [24, 81, 84]; StickyDesign, searching for short orthogonal sequences (although originally designed for tile assembly rather than DSD) [29]; Multistrand, which models the dynamic behavior of short DNA strands [61]; and KinDA, which interfaces with Nupack and Multistrand to check for discrepancies between the behavior of the actual DNA sequences and the desired DSD system [5]. For a simpler option, the Seesaw Compiler [55, 72] uses a pool of domains which will likely work well for more general DSD circuits. So while the layers of programming languages above CRNs are not very well explored, the layers between CRNs and physical DNA are well-developed.

The remainder of this chapter is a slightly modified version of the following previously published work:

> Robert F. Johnson, Qing Dong, and Erik Winfree. Verifying chemical reaction network implementations: A bisimulation approach. *Theoretical Computer Science*, 2018. doi: 10.1016/j.tcs.2018.01.002.
> Contributions: EW formulated the initial questions and definitions of CRN bisimulation, provided advice and assistance throughout, and reviewed and edited the manuscript. QD gave the initial formulation and proof of the equivalence of three notions theorem and the reactionsearch algorithm, contributed an initial manuscript describing those parts, and reviewed the final manuscript. RFJ improved existing definitions and proofs, proved the transitivity and modularity theorems, designed the loopsearch and graphsearch algorithms, proved the completeness results, defined and analyzed the extension for implicit catalysts, and was the primary writer of the final manuscript.

Abstract

Efforts in programming DNA and other biological molecules have recently focused on general schemes to physically implement arbitrary Chemical Reaction Networks. Errors in some of the proposed schemes have driven a desire for formal verification methods. By interpreting each implementation species as a multiset of formal species, the concept of weak bisimulation can be adapted to CRNs in a way that agrees with an intuitive notion of a correct implementation. The theory of CRN bisimulation can be used to prove the correctness of a general implementation scheme or to detect subtle problems. Given a specific formal CRN and a specific implementation CRN, the complexity of finding a valid interpretation between the two CRNs if one exists, and that of checking whether an interpretation is valid are both PSPACE-complete in the general case, but are NP-complete and polynomial-time, respectively, under an assumption that holds in many cases of interest. We present effective algorithms for both of those problems. We further discuss features of CRN bisimulation including a transitivity property and a modularity condition, the precise connection to the general theory of bisimulation, and an extension that takes into account spurious catalysts.

2.2 Background

Introduction

In molecular programming, many real and abstract systems can be expressed in the language of Chemical Reaction Networks (CRNs). A CRN specifies a set of chemical species and the set of reactions those species can do, and the CRN model allows us to deduce the global behavior of the system from that local specification. CRNs are a useful way to separately analyze the computational and the physical aspects of a system. We can use the CRN model to help analyze real systems [7, 8] or design engineered systems [19, 67].

The CRN model is particularly useful as a programming language for molecular computation. Small CRNs have been designed to exhibit simple behaviors and to compute simple problems, such as the “rock-paper-scissors” oscillator, which oscillates between high concentrations of three species in consistent order [25, 44], and the approximate majority network, which converts all of the initial populations of two species to whichever one was initially greater [2]. Other examples of CRNs designed to compute include a CRN that produces an output with count equal to the larger of two input counts [16], and a CRN that simulates a given Turing machine with arbitrarily small probability of error [65]. To show that using CRNs as a programming language can apply to real molecular systems, Chen et al. experimentally demonstrated an implementation of the approximate majority CRN [19], and Srinivas et al. demonstrated an implementation of the rock-paper-scissors oscillator [67], both using DNA strand displacement cascades [87].

In order to discuss how these CRNs compute, we first define a semantics that specifies the behavior of a CRN, then define computation in terms of, for example, from a given input state, how many of or with what probability a given output molecule is produced. The two most common semantics are *deterministic* or *ordinary differential equation (ODE) semantics*, and *stochastic* or *continuous-time Markov chain (CTMC) semantics*. Deterministic semantics assumes real-valued *concentrations* of species which evolve according to a system of ODEs determined by the reactions. Stochastic semantics assumes integer-valued *counts* of species which transition discretely at random times according to the reactions, with rates based on the counts and the reaction rate constants. Any CRN can be interpreted in either semantics, but its behavior may be slightly or entirely different. In deterministic semantics, for example, the rock-paper-scissors oscillator will oscillate correctly indefinitely, and the approximate majority CRN will correctly convert all molecules to whichever species was initially greater quickly, even as the difference between initial counts approaches zero, except for an equilibrium with both species present when the difference is exactly 0. In stochastic semantics, on the other hand, the rock-paper-scissors oscillator will oscillate for a while but eventually undergo an extinction event and stop oscillating [25], while the approximate majority CRN will still compute correctly with high probability, but may convert all molecules to the wrong species with error probability increasing with smaller initial differences [2]. The CRN from [16] that computes the maximum of its input counts functions identically in deterministic and stochastic semantics, which we suspect is related to the rate-independent property discussed below, a connection which has been partially explored by Chen, Doty and Soloveichik [17]. Finally, the CRN to simulate a Turing machine has only been explored in the stochastic model, since it depends on exact integer counts of certain species; its dynamics in deterministic semantics are probably uninteresting.

There is a certain subclass of CRNs that, while they are interpreted in the stochastic model, do not depend on randomness or relative probabilities of certain trajectories to compute an interesting function. This type of computation has been called "deterministic computation" by some [16], but to avoid confusion with deterministic semantics we refer to it in this paper as *rate-independent computation*. Rate-independent computation requires that the correct answer is produced no matter what reactions fire in what order (subject to a fairness condition), a property that implies that for any choice of positive rate constants, the correct answer is produced with probability 1 in stochastic semantics. Rate-independent computation can compute exactly the semilinear functions [1, 16, 26], an example of which is computing the maximum of two input counts [16]. A concept of rate-independent computation in the deterministic model has been explored by Chen, Doty and Soloveichik [17], where it can compute exactly positive continuous and piecewise rational linear functions, but the exact relation between the two concepts of rate-independent computation is unknown.

Despite the current progress in CRN computation, there remains a gap between abstract and real CRNs. To illustrate this gap, consider the approximate majority CRN [2, 19]:

$$X + Y \xrightarrow{k} 2B$$
$$X + B \xrightarrow{k} 2X$$
$$Y + B \xrightarrow{k} 2Y$$

This abstract CRN quickly and with high probability converts all of the initial X and Y molecules into the same amount of whichever one was initially greater [2]. However, no three molecules with exactly this behavior are known to exist. (In a strict sense, no three molecules with *exactly* this behavior can exist, because for all three reactions to be driven forward would require $X + Y$ to be both lower-energy and higher-energy than $2B$.) For contrast, consider the DNA strand displacement system built by Chen et al. [19] meant to implement this abstract CRN. The DNA system uses additional molecules which are consumed as "fuel" to drive these three reactions, ending up with over 25 each of species and reactions. Without knowing that it is meant to be an implementation of the approximate majority CRN, it might be difficult to tell what the DNA system was meant to do. Even knowing the correspondence, it is not obvious that there is no mistake in that complex implementation.

The issue of verifying correctness is exacerbated by the recent profusion of experimental and theoretical implementations in synthetic biology and molecular programming. Of particular interest to us, Soloveichik et al. [66] designed a systematic way to construct a DNA system to simulate an arbitrary CRN. Since then there have been a number of methods to translate an arbitrary CRN into a DNA strand displacement circuit [6, 57, 66]. While each one gave arguments for why it was a correct implementation, they did not come with a general theory of what it means to correctly implement a CRN. In some cases this led to subtle problems, of which we will give examples later. To be certain that such implementations are correct, CRN verification methods were invented. Such methods include Shin's pathway decomposition [64], Lakin et al.'s serializability analysis [46], and Cardelli's morphisms between CRNs [7].

The concept of "bisimulation" was developed as a way of comparing the observable behavior of concurrent systems [52]. Roughly speaking, bisimulation involves coming up with a relation on states such that any action in one of a pair of related states can be matched in the other, leading to another pair of related states. Weak bisimulation, in particular, abstracts away from the details of the system by focusing on "observable" actions and allowing matches between sequences with arbitrary numbers of "silent" actions and one observable action. From this local concept of equivalent states, one can prove global properties of equivalence of the behavior of the systems. Bisimulation has been studied in finite-state systems, Petri nets, and hybrid systems, among others [3, 31, 39]. The complexity of bisimulation in finite systems and in Petri nets (which are equivalent to CRNs) has also been studied; particularly relevant to us, whether an (arbitrary) bisimulation relation exists that relates the initial states of two Petri nets is undecidable [39]. However, the direct application of bisimulation to Petri nets used in that result ignores the structure of a Petri net (or CRN) in the following sense: where bisimulation allows matching arbitrary states with each other, CRNs (and equivalently, Petri nets) have a structure on the state space that allows addition of states, and we might require that the bisimulation relation preserves that addition. For example, if $A \cong B$ and $C \cong D$, where $\cong$ is the bisimulation relation, we might require that $A + C \cong B + D$. If such constraints better capture the notion of "equivalent CRNs", they could also be exploited to simplify the tasks of finding a satisfactory bisimulation or proving that none exists.

Motivated by the expectation that there is a natural class of restricted bisimulations that respect the structure of CRNs in a way well-suited for molecular implementations—and that makes analysis tractable—we present a method for comparing an implementation CRN with an abstract CRN based on the concept of bisimulation from concurrency theory [52]. Our method associates each implementation species with a multiset of formal species, then asserts correctness if the reactions reachable from any implementation state are the same as those in the corresponding state in the abstract CRN. Like pathway decomposition [64] and serializability [46] but unlike Cardelli's morphisms [7], our bisimulation method works with the stochastic model for low-copy-number CRNs and doesn't take into account rates or kinetics. Therefore, like pathway decomposition and serializability, bisimulation cannot prove that a property true with some probability in the formal CRN will be true with the same or even approximately the same probability in an implementation CRN, but it can guarantee that any rate-independent behaviors or computations of the formal CRN will carry over to the implementation. The use of interpretations instead of pathways means that some implementations considered correct by pathway decomposition are considered incorrect by bisimulation and vice versa. Interpretations also make bisimulation more local than pathway decomposition or serializability, in that many properties can be checked on individual reactions rather than pathways; we hope this makes bisimulation more understandable and tractable. We show how bisimulation can be used to prove a CRN implementation correct or identify subtle problems. We present an algorithm to check whether a particular interpretation between two CRNs is a bisimulation relation, and an algorithm to find such an interpretation if one exists. We analyze the computational complexity of both problems. We prove that both are PSPACE-complete in the general case but become polynomial time and NP-complete, respectively, when formal reactions are limited to a constant number of reactants. We hope this method can be used in both verifying that engineered systems match their specification and in comparing natural systems to a system simple enough to analyze.

Related Works

Our research into verification of CRN implementations is inspired by a number of works on implementing arbitrary CRNs with DNA strand displacement, such as [6, 57, 66]. Soloveichik et al. in [66] present a general construction for DSD implementations of arbitrary CRNs, give an argument that the ODE semantics of their construction should approximate the original CRN in a certain limit, and demonstrate similar (but not identical) behavior on some example CRNs. However, this argument does not address the stochastic model, or rate-independent computation within the stochastic model. (We will show in Section 2.4 that the construction in [66] is in fact correct for rate-independent computation according to our definition of bisimulation, but this is not proven in [66].) Qian et al. in [57] present a "history-free" general CRN implementation suited to the stack machines they then describe, and give an argument for its qualitative correctness in the stochastic model, but the argument is non-rigorous. In fact, the argument in [57] misses an error in the construction as published when applied to certain combinations of reactions, allowing some probability of incorrect behavior when run with low counts of species, which we will discuss further in Section 2.4. Cardelli in [6] presents a simplified, history-free general CRN implementation using only nicked double-stranded gates and single-stranded signals, and defines an algebra which can be used to prove statements about the behavior of such systems. This algebra can prove some desirable properties, but Cardelli in [6] acknowledges that properties such as lack of crosstalk require exploring large state spaces and are thus difficult to prove with that technique.

Visual DSD [45] and Peppercorn [36] provide formal semantics for the behavior of a DNA strand displacement such as the ones mentioned in [6, 57, 66]. Both [45] and [36] also provide an algorithm to, given a DSD system, produce a CRN that models its behavior. This allows us to reduce the problem of verifying DSD implementations of CRNs to the problem of checking whether one CRN is a correct implementation of another.

In addition to the bisimulation method we propose, other methods have been proposed for verifying CRN equivalence for the use case of DNA strand displacement implementations of CRNs, usually focusing on rate-independent computation in the stochastic model. Lakin et al. define a method of verification of an implementation CRN that is already divided into one module for each formal reaction [46]. This verification method gives properties of each module individually and of non-interaction between modules that, if satisfied, imply that every trajectory of the implementation CRN is equivalent to a serial execution of trajectories corresponding to some sequence of formal reactions, which is at least sufficient for correctness of rate-independent computation. Shin et al. define a method of computing the "formal basis" of a CRN based on pathways in the implementation CRN that begin and end in formal species [64]. If every pathway can be decomposed into the pathways that make up the formal basis, and certain niceness conditions are satisfied, then this "pathway decomposition" method can guarantee that the implementation CRN will be equivalent, at least in rate-independent computation, to its formal basis. Both of these methods depend on a division of the implementation CRN into "formal" species, "fuel" species, "waste" species, and "intermediate" species. This division is a typical feature of engineered DNA strand displacement implementations of CRNs such as the ones mentioned above, but may not generalize to other types of implementations or to comparing natural systems. Lakin et al. [46] give no algorithm for computationally checking the conditions of their serializability method when they are not already known by design; Shin et al. [64] give an algorithm for finding the formal basis of a CRN, but its complexity is not known and it seems to require more than polynomial space in the general case. Both of these methods have been applied to DNA strand displacement implementations such as those in [6, 57, 66]. Where Lakin et al. [46] discuss a method of verification that they apply to a specific class of implementations (two-domain strand displacement), and Shin et al. [64] discuss a method with an algorithm that can be applied to any individual implementation CRN, we hope to present a method with an algorithm that can verify any individual implementation CRN, but is also tractable enough to permit proofs that, for example, any implementation created by a given translation scheme will be correct.

Another area of related work has explored more abstract forms of model reduction and CRN equivalence not shaped by DNA strand displacement CRN implementations. Gay et al. discuss a method meant to be useful for systems biology models, where the exact structure of the network has a number of uncertainties and unknowns [33]. This method is based on two operations, merge and delete, which can be applied to graphs of the reactions of a CRN, such that there is an epimorphism from the detailed graph to the more simple graph if and only if the simple graph is equivalent to the result of some sequence of these rules applied to the detailed graph. Some of these concepts correspond loosely to concepts in bisimulation, for example the merging of two species in [33] is intuitively similar to two implementation species being interpreted as the same formal species, but [33] does not have any equivalent of the requirement in bisimulation that any reaction involving one of two equivalent species has an equivalent reaction that involves the other instead. Possibly for this reason, Gay et al. do not prove any properties of the behavior of their networks that a graph epimorphism implies carry over from one network to another.

In a line more directly related, Cardelli, Tribastone, Tschaikowski, Vandin, and Tognazzi have developed multiple concepts of CRN equivalence based on strong bisimulation. *Forward* and *backward bisimulation* respectively imply two different forms of equivalence of the ODE semantics behavior of the equivalent CRNs [10, 12], while *syntactic Markovian bisimulation* implies equivalence of the CTMC semantics [14]. Each of these types of bisimulation is an equivalence relation between species of a single CRN, with properties that imply that a simpler CRN, whose species are the equivalence classes of that relation, has dynamics that are well-defined and equivalent in the appropriate sense to the original CRN. In all three cases it is easily checkable whether a given relation is a (forward, backward, or syntactic Markovian) bisimulation, and the same authors give algorithms and software tools to find such relations

[11, 13, 14, 74]. All three of these methods correspond to strong bisimulation in the sense of [52]: one reaction in one system must be matched by exactly one reaction in the other, and vice versa, as opposed to weak bisimulation, where one reaction in one system can be matched by zero or more reactions in the other. This allows these methods to imply equivalence of kinetics in the deterministic and stochastic models, respectively, which would be much more difficult with weak bisimulation. These methods also use a relation on species to induce the relation on states required by bisimulation in the sense of [52], which disallows phenomena such as a single strand of a DSD circuit representing that one copy of the formal species C and two copies of formal D are all present. Since DSD CRN implementations tend to use both one DSD species representing multiple formal species and one formal reaction taking multiple DSD reactions to implement, none of the DSD schemes presented in [6, 57, 66] are equivalent to their formal specifications according to forward, backward, or syntactic Markovian bisimulation.

2.3 The Chemical Reaction Network Model

We work within the *Chemical Reaction Network* (CRN) model. A CRN is a pair $(\mathcal{S}, \mathcal{R})$, where $\mathcal{S}$ is a finite set of species and $\mathcal{R}$ a finite set of reactions. A *reaction* is itself a triple (R, P, k), where the *reactants* R and *products* P are both multisets of species, and $k > 0$ is a real number. We require that in any reaction $R \neq P$, and that no two reactions (R, P, k_1) and (R, P, k_2) with the same reactants and products exist in the same CRN. Once we define semantics, we will see that these requirements can be met without loss of generality; in both deterministic and stochastic semantics, a reaction (R, R, k) has no effect on the CRN's behavior, and a CRN with two reactions (R, P, k_1) and (R, P, k_2) behaves exactly the same as one where those two are replaced by one reaction $(R, P, k_1 + k_2)$.

We use the notation $\{\!|\ldots|\!\}$ for multisets interchangeably with the chemical notation, e.g. $2A + B$, $\{\!|A, A, B|\!\}$, and $\{\!|2A, B|\!\}$ all refer to the same state. Similarly, we sometimes use the chemical notation for reactions, e.g. $A + B \xrightarrow{k} 2C$ is the same as $(\{\!|A, B|\!\}, \{\!|2C|\!\}, k)$. The "reversible reaction" notation $A + B \underset{k_2}{\overset{k_1}{\rightleftharpoons}} 2C$ is a shorthand for the two reactions $(\{\!|A, B|\!\}, \{\!|2C|\!\}, k_1)$ and $(\{\!|2C|\!\}, \{\!|A, B|\!\}, k_2)$. Where $S \in \mathbb{N}^{\mathcal{S}}$ is a multiset and $X \in \mathcal{S}$, $S(X)$ refers to the count of X in S; this matches the formal definition of $\mathbb{N}^{\mathcal{S}}$ as the set of functions $\mathcal{S} \to \mathbb{N}$. Multisets can be added and multiplied by scalars componentwise, and can be compared componentwise: $S \leq T \iff \forall_X S(X) \leq T(X)$, and $S < T$ if $S \leq T$ and $S \neq T$. If $S \leq T$ then subtraction $T - S$ is defined componentwise. Set operations involving multisets implicitly treat each multiset as the set of all objects which appear at least once; e.g. $\{\!|1,1,2|\!\} \subset \{1,2,3\}$ but $\{\!|1,1,2|\!\} \not\subset \{1\}$. $S \wedge T$ is the componentwise minimum, $(S \wedge T)(X) = \min(S(X), T(X))$. $S \backslash T = S - (S \wedge T)$ is the removal of T from S: $(S \backslash T)(X) = \max(S(X) - T(X), 0)$. The size of a multiset $S \in \mathbb{N}^{\mathcal{S}}$ is the number of objects in it, $|S| = \sum_{X \in \mathcal{S}} S(X)$.

The CRN model comes with two common interpretations or semantics: deterministic semantics and stochastic semantics. In deterministic semantics, the state of the CRN at any given time is a vector $s \in \mathbb{R}^{\mathcal{S}}_{\geq 0}$ of the concentrations of each species, which evolves according to a set of ordinary differential equations. These differential equations come from the reactions, and for each $X \in \mathcal{S}$, where $r_i = (R_i, P_i, k_i)$, $\frac{ds(X)}{dt} = \sum_{r_i \in \mathcal{R}} \rho(r_i, s)(P_i(X) - R_i(X))$. Here $\rho(r_i, s)$ refers to the "rate" of reaction r_i in state s, which according to *mass-action kinetics* is given by $\rho(r_i, s) = k_i \prod_{Y \in \mathcal{S}} s(Y)^{R_i(Y)}$. Since this paper mostly does not deal with deterministic semantics, we only briefly mention it.

In stochastic semantics, the state of the CRN at any given time is a vector $S \in \mathbb{N}^{\mathcal{S}}$ of the counts of each species, which transitions probabilistically to other states, forming a continuous-time Markov chain. In any given state S, each reaction $r_i = (R_i, P_i, k_i)$ has a *propensity* of firing, which in stochastic mass-action kinetics is $\rho(r_i, S) = k_i \prod_{X \in \mathcal{S}} \frac{S(X)!}{(S(X) - R_i(X))!}$, and the reaction takes the CRN from state S to state $S - R_i + P_i$. Note that this expression is defined if and only if $S \geq R_i$; when $S \not\geq R_i$ we set $\rho(r_i, S) = 0$. These propensities play the role of state transition rates in the continuous-time Markov chain.

In the stochastic model, each possible behavior of a CRN is specified by a timed trajectory: an initial state $S_0 \in \mathbb{N}^{\mathcal{S}}$ together with a (finite or infinite) sequence of reactions $r_i = (R_i, P_i, k_i) \in \mathcal{R}$ and times t_i at which they occur, with $t_i > t_{i-1} > 0$. When we care only which reactions happened in what order but not at what exact time, we can define a *trajectory* as an initial state followed by a sequence of reactions, without the times; each trajectory can be identified with the set of all timed trajectories with that initial state and sequence of reactions. A trajectory implicitly specifies a sequence of states $S_i = S_0 + \sum_{j \leq i}(P_j - R_j)$, but a sequence of states is not enough to specify a trajectory. For example, if $A \xrightarrow{k_1} B$ and $X + A \xrightarrow{k_2} X + B$ are both reactions, then the sequence of two states $(S_0, S_1) = (\{\!|X, A|\!\}, \{\!|X, B|\!\})$ does not specify which of those two reactions happened, which is sometimes important. The CTMC implicitly specifies a probability distribution over timed trajectories, and since a trajectory interpreted as a set of timed trajectories will be a measurable set in this probability space, we also get a probability distribution over trajectories. For rate-independent computation, we care only about which trajectories are possible, ignoring the times of the reactions and the relative probabilities. We say that a finite trajectory is *valid* if and only if it has nonzero probability, and an infinite trajectory is valid if every finite prefix is valid. Note that a trajectory is valid if and only if every reaction is possible in the state it occurs, i.e. $R_i \leq S_{i-1}$ for all i. Since whether a trajectory is valid in a CRN does not depend on the rate constants of any reaction (as long as they are all positive), and from here on we are generally working with rate-independent computation, we write reactions as a pair (R, P) or $R \to P$ instead of a triple (R, P, k) or $R \xrightarrow{k} P$. In general when we speak of "the trajectories of a CRN" we mean the valid trajectories.

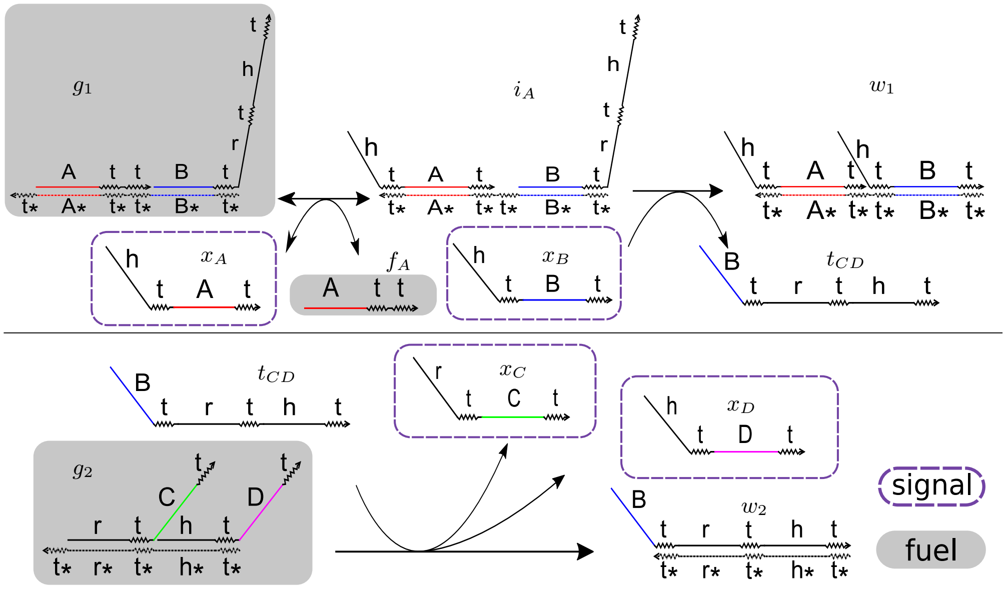
A state T is *reachable* from a state S if T is the result of a valid finite trajectory that starts in S. We say a state T is *coverable* from a state S if there is some $T' \geq T$ such that T' is reachable from S. While the set of reachable states (from any given initial state) is an important aspect of the behavior of a CRN, it does not contain all the information about that CRN. For example, the two CRNs $(\{A, B, C\}, \{A \to B, B \to C, C \to A\})$ and $(\{A, B, C\}, \{A \to C, C \to B, B \to A\})$ have exactly the same set of reachable states T from any given initial state S, but in an external context that distinguishes A, B, and C from each other these two sets of reactions are clearly different in a meaningful way. If however the sets of (valid) trajectories of two CRNs are the same, then the two CRNs must be identical: since in particular the length-zero trajectories (i.e. states) are the same, so the sets of species are the same, and the length-one trajectories (single reactions) are the same. We say that two CRNs are *isomorphic* if there is a bijection between the sets of species such that the set of reactions of one, after applying this bijection, equals the set of reactions of the other.

2.4 The Meaning of Correctness

Interpretations

Schemes for translating an arbitrary abstract CRN into a DNA Strand Displacement (DSD) implementation [6, 57, 66] provide designs for the necessary DNA molecules, but how these molecules interact is best described by a model of the relevant biophysics. Reaction enumerators such as Visual DSD [45] and Peppercorn [36] produce, given a set of DNA molecules, a description of their predicted interactions as a CRN, allowing us to compare it to the original CRN using the same language. Since most molecular systems can be described as CRNs, defining correctness as a comparison of CRNs will also cover much more general cases, not limited to DNA strand displacement. We refer to the original abstract CRN as the *formal CRN* $(\mathcal{S}, \mathcal{R})$ and the model's enumerated CRN as the *implementation CRN* $(\mathcal{S}', \mathcal{R}')$, which is usually larger than the formal CRN. As a convention, we assume that the formal CRN and the implementation CRN make use of disjoint sets of species. When using verification to compare a detailed model of a natural system with unknown function to a simpler abstract CRN with known function, the natural system is the implementation and the abstract system is the formal CRN.

Although the definition of correctness we will propose is general, some of its parts are inspired by engineered implementations such as DNA strand displacement. There are three important features typical of engineered implementation CRNs that a concept of correctness must deal with. First, there is typically for each formal species A an implementation species x_A intended to correspond specifically to it, sometimes called a "signal species". Second, certain implementation species must always be present for the system to work, and are designated "fuel species". Fuel species are typically assumed to be held at a constant concentration, for example by setting their initial concentration high enough that it does not vary significantly over the running time of the CRN. In this situation, we can approximate the implementation CRN by a simplified CRN with all fuel species removed; e.g. if g_1 is a fuel, the reaction $x_A + g_1 \to i_A$ can be replaced by $x_A \to i_A$ with no loss of meaning. This approximation holds not just for rate-independent computation, but for both stochastic and deterministic semantics in the following sense: if a species g_1 is (approximately) at a constant concentration c over the time interval considered, the equations introduced in Section 2.3 for the dynamics of all other species will be (approximately) the same if the reaction $x_A + g_1 \xrightarrow{k} i_A$ is replaced by $x_A \xrightarrow{kc} i_A$ [66]. Third, certain species are inert side products whose further presence or absence has no effect on the correctness of the system behavior, and are thus designated "waste species". Such species can also be removed with no effect on the system dynamics. However, one advantage of our theory is that it does not need to distinguish signal species or waste species from each other or from other species: while knowing that a given species is a signal or a waste can be a helpful hint for finding an interpretation in our theory, it is not necessary, and there are no special rules for signal or waste species. Our theory still requires that fuel species be removed before applying the theory.



$$\begin{aligned} x_A + g_1 &\rightleftharpoons i_A + f_A \\ i_A + x_B &\rightarrow t_{CD} + w_1 \\ t_{CD} + g_2 &\rightarrow x_C + x_D + w_2 \end{aligned}$$

$$\begin{aligned} x_A &\rightleftharpoons i_A \\ i_A + x_B &\rightarrow t_{CD} + w_1 \\ t_{CD} &\rightarrow x_C + x_D + w_2 \end{aligned}$$

Figure 2.1: Implementation of $A + B \rightarrow C + D$ using the scheme described in [66]. Top: DNA complexes and reactions, given as a diagram of the DNA strand displacement circuit. Each complex shown in the diagram is one species in the enumerated CRN, and arrows are reactions that would be enumerated by a reaction enumerator. Designated "signal" species are enclosed in dashed boxes, and designated "fuel" species in grey boxes. Bottom left: Direct translation of reactions in the implementation CRN. Bottom right: Implementation CRN after removing fuels.

Figure 2.1 gives an example of this process for the formal reaction $A + B \rightarrow C + D$, yielding an implementation CRN with four reactions. (Names such as x_A and t_{CD} are based on the intent of the designers of the CRN, but the subscripts have no theoretical meaning.) The signal species x_A can freely convert to and from i_A, and the strand t_{CD} can produce the signals x_C and x_D (and waste w_2). Intuitively, i_A *is* an A and t_{CD} *is* a C and a D; in this sense the first and third reactions are silent, and the second *is* $A + B \to C + D$. We use this intuition as a basis for our definition of correctness. Formally, we define an *interpretation* of the implementation species (Figure 2.2):

Definition 2.4.1. An *interpretation* is a function $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ from implementation species to multisets of formal species. We extend this linearly from species to states: $m(\sum_{i=1}^{n} a_i X_i) = \sum_{i=1}^{n} a_i m(X_i)$. We also define an interpretation of reactions: $m(R' \to P') = m(R') \to m(P')$ unless $m(R') = m(P')$, in which case $m(R' \to P') = \tau$ and we say the reaction is *trivial*. For example, if $m(i_{AB}) = A+B$, $m(x_A) = A$, and $m(t_{BC}) = B+C$ then $m(i_{AB}+x_A) = 2A+B$, and $m(i_{AB} \to x_A + t_{BC}) = A + B \to A + B + C$.

The interpretation of an implementation reaction is always a pair (R, P) of multisets of formal species, or τ, but (R, P) may not be in $\mathcal{R}$. Any such pair is a *reaction in the language of the formal CRN*, but is a *formal reaction* only if $(R, P) \in \mathcal{R}$. Similarly, (R', P') is an *implementation reaction* only if it is in $\mathcal{R}'$.

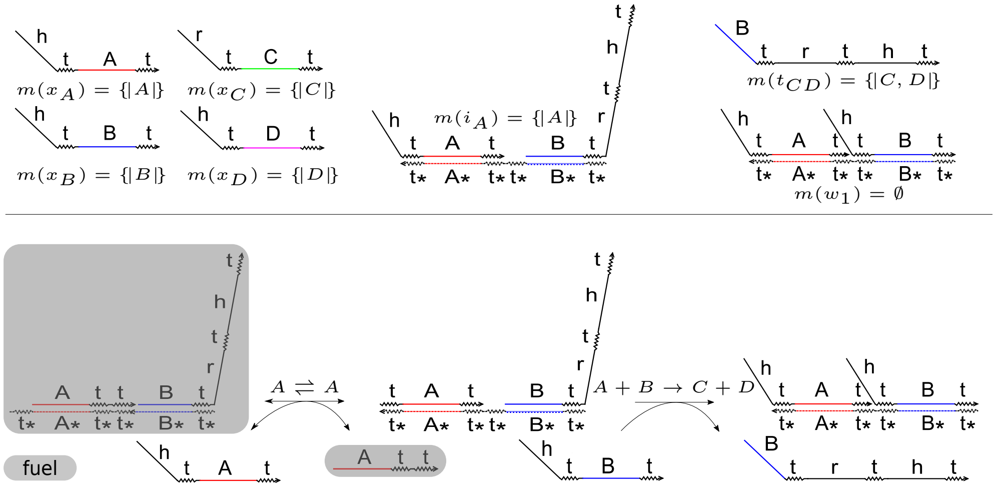


Figure 2.2: Interpretation of the implementation CRN in Figure 2.1. $m(t_{CD}) = A + B$ would also be a valid interpretation for this CRN.

In the following notation, S', T', S'', and T'' refer to implementation states; S and T to formal states; r' to an implementation reaction; and r to a reaction in the language of the formal CRN or τ. When a formal reaction r takes state S to state T, we write $S \xrightarrow{r} T$; $S' \xrightarrow{r'} T'$ is similar. Note that if $S \xrightarrow{r} T$, then $r = (R, P) \in \mathcal{R}$ as well as $S - R + P = T$, and analogously for the implementation. Further, we write $S' \xrightarrow{r} T'$ when $S' \xrightarrow{r'} T'$ for some r' with $m(r') = r$, which does not require $r \in \mathcal{R}$ (but does require $r' \in \mathcal{R}'$). Note that if $S' \xrightarrow{\tau} T'$ then $m(S') = m(T')$. One of the core concepts of weak bisimulation is the behavior of the system when we abstract away from trivial reactions. To discuss this behavior, we introduce the relation $S' \overset{r}{\Rightarrow} T'$: we say $S' \overset{\tau}{\Rightarrow} T'$ when S' can reach T' via 0 or more trivial reactions, and $S' \overset{r'}{\Rightarrow} T'$ when $S' \overset{\tau}{\Rightarrow} S'' \xrightarrow{r'} T'' \overset{\tau}{\Rightarrow} T'$. Note that $S' \overset{\tau}{\Rightarrow} S'$ and $S \overset{\tau}{\Rightarrow} S$ are always true. $S' \overset{r}{\Rightarrow} T'$ for $r \neq \tau$ is again defined as $S' \overset{r'}{\Rightarrow} T'$ for some r' with $m(r') = r$. $S \overset{r}{\Rightarrow} T$ for $r \neq \tau$ is defined but trivial: $S \overset{r}{\Rightarrow} T \iff S \xrightarrow{r} T$. When the final state is irrelevant, we sometimes write $S' \overset{r'}{\Rightarrow}$, etc., as appropriate. We say an implementation state S' *can reach* an implementation reaction r' when $S' \overset{r'}{\Rightarrow}$, and we say S' *can implement* a formal reaction r when $S' \overset{r}{\Rightarrow}$. (These definitions are based on notation used by Milner in [52], a connection that will be further discussed in Section 2.6.)

Three Notions of Correctness

Our notion of correctness is motivated by the earlier observation that the set of valid trajectories defines equivalence between formal CRNs, and allowing renaming of species defines isomorphism. Applying this notion to an implementation CRN with an interpretation introduces two difficulties. First, due to trivial reactions, the implementation trajectory may involve more steps. This is easily solved by defining the interpretation of an implementation trajectory to remove trivial reactions. Second, and more seriously, the full set of interpreted implementation trajectories may cover the formal trajectories, yet particular implementation trajectories may experience restricted options for alternative paths. Two examples of this are shown in Figures 2.3 and 2.4.

In the first example (Figure 2.3), there are two implementation species, x_B and y_B, that are both interpreted as B. Because x_B can do anything in the implementation CRN that B can do in the formal CRN, and because x_A can react to become x_B, the implementation CRN can match any trajectory of the formal CRN using x_B and ignoring y_B. However, an implementation trajectory that reaches—or starts from—y_B cannot proceed, whereas, the formal CRN cannot get stuck. To see that the key issue concerns limiting options, of which

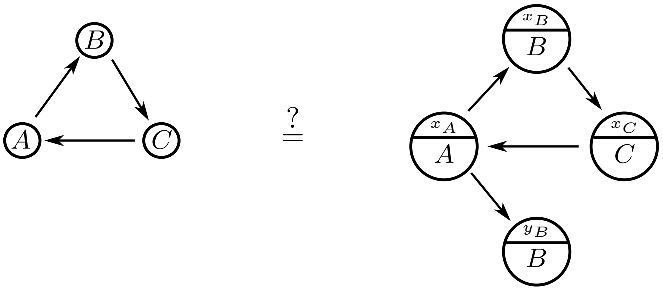


Figure 2.3: Example CRNs with the same set of (interpreted) trajectories but different behavior. Circles represent species and arrows represent reactions; implementation species are given with their name above the line and interpretation below the line, so for example x_A is an implementation species with $m(x_A) = \{\!|A|\!\}$. Both CRNs have the same set of trajectories (after interpretation): from any initial count of As, Bs, and Cs, each species can independently change from A to B, B to C, and C to A any finite or infinite number of times. However, the implementation CRN (right) can convert all species to y_B, from which no further reactions are possible, while the corresponding formal state of all Bs can react further.

getting stuck is just a special case, the reader is encouraged to construct an example where the formal CRN is the one in Figure 2.3 augmented by the three reverse reactions, but the implementation CRN can become trapped in a subspace where only the “clockwise” trajectories are possible, although it can never get stuck. To appreciate the subtlety of the problem, in our second example (Figure 2.4), there are two forms of each formal species, and while the x forms can copy the formal CRN exactly, if all species are converted to y forms in the implementation CRN then no further reaction can happen. As mentioned earlier, this paper is not concerned about differences in kinetics or the probability distribution over trajectories; however, we would like to be able to ensure that properties about what states can be visited in the future, and in what order, are preserved in the implementation. Effectively, the naive definition of trajectory equivalence requires that for every formal state there *exists* an implementation state with the same interpretation and behavior, while we need a finer-grained notion of trajectory equivalence that requires for every formal state, *all* implementation states with the same interpretation have the same behavior. As defined formally below, the finer-grained notion becomes a satisfactory definition of correctness.

Although trajectory equivalence as defined below has the desired meaning,

$$\begin{aligned} A + B &\xrightarrow{1.0} 2B \\ B + C &\xrightarrow{1.0} 2C \\ C + A &\xrightarrow{1.0} 2A \\ 2A &\xrightarrow{0.01} 2B \\ 2B &\xrightarrow{0.01} 2C \\ 2C &\xrightarrow{0.01} 2A \end{aligned}$$

$$\begin{aligned} x_A + x_B &\xrightarrow{1.0} 2x_B & y_A + x_B &\xrightarrow{1.0} x_B + y_B \\ x_B + x_C &\xrightarrow{1.0} 2x_C & y_B + x_C &\xrightarrow{1.0} x_C + y_C \\ x_C + x_A &\xrightarrow{1.0} 2x_A & y_C + x_A &\xrightarrow{1.0} x_A + y_A \\ 2x_A &\xrightarrow{0.01} 2x_B & x_A + y_A &\xrightarrow{0.01} 2y_B \\ 2x_B &\xrightarrow{0.01} 2x_C & x_B + y_B &\xrightarrow{0.01} 2y_C \\ 2x_C &\xrightarrow{0.01} 2x_A & x_C + y_C &\xrightarrow{0.01} 2y_A \end{aligned}$$

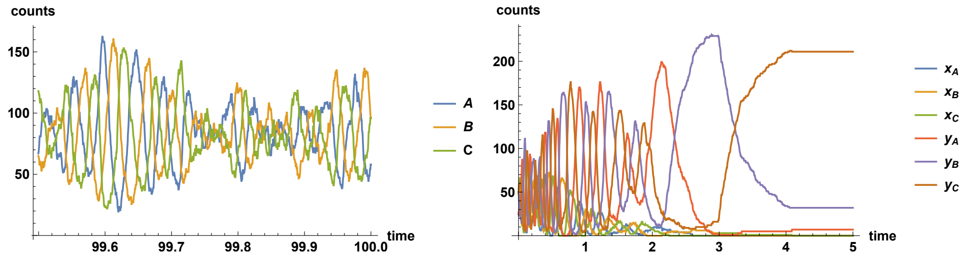


Figure 2.4: Modified version of the rock-paper-scissors oscillator [25, 44] and an incorrect implementation. Adding the reactions $2A \xrightarrow{0.01} 2B$ etc. ensures that the formal CRN oscillates forever under stochastic semantics (left CRN, left image); without these reactions, eventually the count of one will hit zero and can never be recovered [25]. An implementation CRN with two variants (x_A, y_A, etc.) of each formal species oscillates correctly over short periods of time, and the sets of trajectories of the two CRNs are the same; however, the implementation CRN can and eventually will reach a state where all species are in y form, in which case no further reactions can happen (right CRN, right image).

since the sets of trajectories are generally infinite, we would like a more local definition that facilitates efficient computational analysis. We define three local conditions on the interpretation which we show are equivalent to trajectory equivalence. As further evidence that our notion of correctness is sound, we show that these three conditions are equivalent to a special case of weak bisimulation from concurrency theory [52]. (We discuss this connection further in Section 2.6.) This gives us three notions of correctness, given a formal CRN, an implementation CRN, and an interpretation:

Definition 2.4.2 (Three notions of correctness)**.** An implementation CRN $(\mathcal{S}', \mathcal{R}')$ is a correct implementation of a formal CRN $(\mathcal{S}, \mathcal{R})$ if a correct interpretation exists. An interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ is correct if any of the following three sets of conditions are true:

I **Equivalence of trajectories**

(i) The set of formal trajectories and interpretations of implementation trajectories are equal.

(ii) For every implementation state S', the set of formal trajectories starting from $m(S')$ and interpretations of implementation trajectories starting from S' are equal.

II **Three conditions on the interpretation**

(i) *Atomic condition:* For every formal species A, there exists an implementation species x_A such that $m(x_A) = \{\!|A|\!\}$.

(ii) *Delimiting condition:* The interpretation of any implementation reaction is either trivial or a valid formal reaction.

(iii) *Permissive condition:* If $S \xrightarrow{r}$ and $m(S') = S$, there exists an implementation reaction r' such that $m(r') = r$ and $S' \overset{r'}{\Rightarrow}$.

III **Weak bisimulation**

(i) For all implementation states S',
if $S' \xrightarrow{r} T'$, then $S \overset{r}{\Rightarrow} T$ where $S = m(S')$ and $T = m(T')$.

(ii) For all formal states S, there exists S' with $m(S') = S$, and for all such S',
if $S \xrightarrow{r} T$, then for some T', $S' \overset{r}{\Rightarrow} T'$ and $m(T') = T$.

A few comments may help explain these definitions. It may seem that the second condition for trajectory equivalence supersedes the first, but it does not: for example, the second condition may be satisfied even if there is no implementation state S' that is interpreted as formal state S, whereas the first condition will not be satisfied in that case. In our definition of weak bisimulation, the use of T and T' is in some sense redundant due to the structure of CRNs: the resulting state of a reaction is determined by the initial state and the reaction, so for example if $S \xrightarrow{r} T$ and $S' \overset{r}{\Rightarrow} T'$ then it already must be the case that $m(T') = T$. The definitions of the delimiting and permissive conditions are thus more suited to the structure of CRNs; we gave the definition of weak bisimulation as we did to match the definitions used in concurrency theory [52], and Theorem 2.4.1 proves that this definition is still equivalent. Because of the connection to bisimulation theory, when $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ is a correct interpretation we often say that m is a bisimulation from $(\mathcal{S}', \mathcal{R}')$ to $(\mathcal{S}, \mathcal{R})$. When the distinction is important, we refer to a bisimulation that satisfies our additional restrictions (i.e., is an interpretation that satisfies the atomic condition) as a "CRN bisimulation", but for most of this paper when not explicitly comparing the different theories of bisimulation we use "bisimulation" to mean "CRN bisimulation".

Theorem 2.4.1. *The three definitions of correctness, namely trajectory equivalence, the three conditions on the interpretation, and weak bisimulation, are equivalent.*

Proof. We show that trajectory equivalence implies the three conditions formulation; the three conditions imply weak bisimulation; and weak bisimulation implies trajectory equivalence.

Given trajectory equivalence, we prove the three conditions on m. First, for the atomic condition, consider applying condition I.(i) of trajectory equivalence to formal trajectories of length 0, which are just formal states, and in particular formal states $S_A = \{\!|A|\!\}$ for each formal species A. That the set of trajectories are equal implies that there is an implementation trajectory whose interpretation is the (zero-length trajectory) state S_A, i.e. an implementation state S'_A with $m(S'_A) = \{\!|A|\!\}$. Since implementation species cannot be interpreted as fractional or negative formal species, there is some species $x_A \in S'_A$ with $m(x_A) = \{\!|A|\!\}$, satisfying the atomic condition. Second, for the delimiting condition, consider implementation trajectories of length 1, specifically for each implementation reaction $r' = (R', P')$ the trajectory $R' \xrightarrow{r'} P'$. If r' is trivial, that is $m(r') = \tau$, its interpreted trajectory is a zero-length trajectory; if not, its interpreted trajectory is $m(R') \xrightarrow{m(r')} m(P')$, which by trajectory equivalence must be a formal trajectory. For that to be so, $m(r')$ must be a reaction in $\mathcal{R}$, thus satisfying the delimiting condition. And third, for the permissive condition, for every formal reaction $r = (R, P)$ and implementation state S' with $m(S') \geq R$, the trajectory $m(S') \xrightarrow{r} T$, where $T = m(S') - R + P$, is a formal trajectory. By condition I.(ii) of trajectory equivalence, there is an implementation trajectory starting in S' whose interpreted trajectory is $m(S') \xrightarrow{r} T$. (Note that condition I.(i) implies this for *some* S' with $m(S') = S$, but not necessarily for every S'.) To have that interpretation, that implementation trajectory must have some reaction r' with $m(r') = r$ and all other reactions trivial; this is the definition of $S' \overset{r}{\Rightarrow}$, satisfying the permissive condition.

Given the three conditions, we prove weak bisimulation. Given any S' with $m(S') = S$ and $S' \xrightarrow{r'} T'$ where $r' = (R', P')$, by the delimiting condition either $m(r') = \tau$ is trivial or $m(r') = r = (R, P) \in \mathcal{R}$. If trivial, then $m(T') = m(S') = S$ and $S \overset{\tau}{\Rightarrow} S$ is true by convention. If nontrivial, then $r \in \mathcal{R}$; since $S' \xrightarrow{r'}$ we must have $S' \geq R'$, thus $m(S') \geq m(R') = R$, and $S \xrightarrow{r} T$ (therefore $S \overset{r}{\Rightarrow} T$) where $T = S - R + P$. Since $T' = S' - R' + P'$, $m(T') = m(S') - m(R') + m(P') = T$, satisfying condition III.(i) of weak bisimulation. Given any S, we find an S' with $m(S') = S$ by taking $S' = \sum_A \alpha_A x_A$, where $S = \sum_A \alpha_A A$ and $m(x_A) = \{\!|A|\!\}$ must exist by the atomic condition. Given any such S' with $S \xrightarrow{r} T$ where $r = (R, P)$, by the permissive condition there is some r' with $m(r') = r$ and $S' \overset{r'}{\Rightarrow}$, which is an abbreviation for $\exists_{T'} S' \overset{r'}{\Rightarrow} T'$, which is further an abbreviation for $\exists_{S''} S' \overset{\tau}{\Rightarrow} S'' \xrightarrow{r'} T'$. (Strictly speaking $S' \overset{r'}{\Rightarrow} T'$ means there is some $S' \overset{\tau}{\Rightarrow} S'' \xrightarrow{r'} T'' \overset{\tau}{\Rightarrow} T'$, but since we are choosing an arbitrary T' we can take $T' = T''$.) Then $m(S') = m(S'') = S$ since they are connected by trivial reactions, and where $r' = (R', P')$ with $m(R') = R$ and $m(P') = P$ we have $T' = S'' - R' + P'$ so $m(T') = S - R + P = T$, satisfying condition III.(ii) of weak bisimulation.

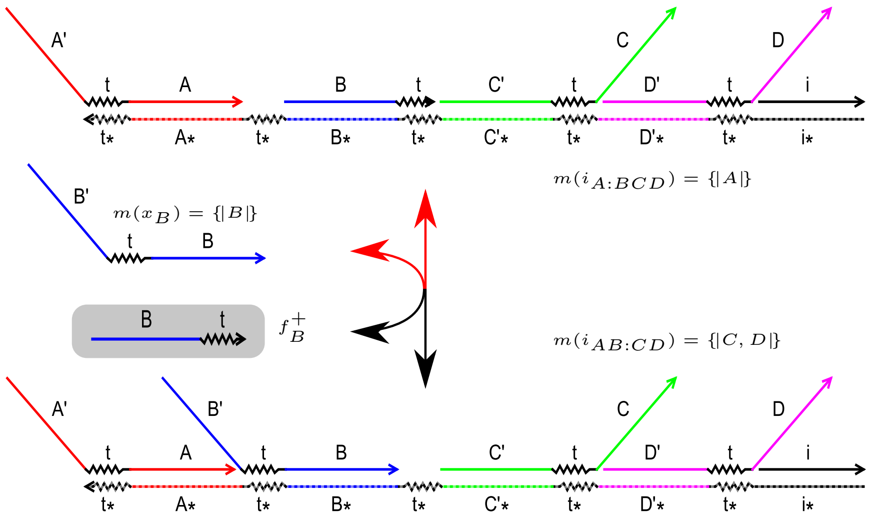
Given weak bisimulation, we prove trajectory equivalence. We first prove condition I.(ii). Given any S'_0 with $S_0 = m(S'_0)$ and any implementation trajectory $(S'_0, r'_1, \ldots, r'_k, \ldots)$ with $r'_k = (R'_k, P'_k)$, let $S'_k = S'_{k-1} - R'_k + P'_k = S'_0 - \sum_{i \leq k} R'_i + \sum_{i \leq k} P'_k$. Letting $S_k = m(S'_k)$ and $r_k = m(r'_k)$, it follows that either $r_k = \tau$ or else $r_k = (R_k, P_k)$ and $S_k = S_{k-1} - R_k + P_k = S_0 - \sum_{i \leq k} R_k + \sum_{i \leq k} P_k$ by linearity of m. From bisimulation, since each $S'_{k-1} \xrightarrow{r'_k} S'_k$ we have either $r_k = \tau$ and $S_{k-1} = S_k$, or $r \neq \tau$ and $S_{k-1} \xrightarrow{r_k} S_k$, since for $r \neq \tau$ in the formal CRN $S \overset{r}{\Rightarrow} T \iff S \xrightarrow{r} T$. The interpretation of that implementation trajectory is exactly S_0 followed by those reactions $S_{k-1} \xrightarrow{r_k} S_k$ for which $r_k \neq \tau$, and thus the interpretation is a formal trajectory. Conversely, given S'_0 with $S_0 = m(S'_0)$ and any formal trajectory $(S_0, r_1, \ldots, r_k, \ldots)$ with $r_k = (R_k, P_k)$, letting $S_k = S_{k-1} - R_k + P_k = S_0 - \sum_{i \leq k} R_k + \sum_{i \leq k} P_k$, we construct an implementation trajectory whose interpretation is that formal trajectory. Given S'_0, define inductively S'_k and r'_k to be an implementation state and reaction such that $S'_{k-1} \overset{r'_k}{\Rightarrow} S'_k$ with $m(r'_k) = r_k$ and $m(S'_k) = S_k$, which exists by condition III.(ii) of weak bisimulation. Expanding each $\overset{r'_k}{\Rightarrow}$ implicitly defines an implementation trajectory $(S'_0, r''_{1,1}, \ldots, r''_{1,l_1}, r'_1, r''_{2,1}, \ldots)$ where each $m(r''_{k,j}) = \tau$ and each $m(r'_k) = r_k$; the interpretation of this trajectory is the formal trajectory $(S_0, r_1, \ldots, r_k, \ldots)$ as desired, proving condition I.(ii). Condition I.(i) follows from condition I.(ii) of trajectory equivalence and condition III.(ii) of weak bisimulation: every implementation trajectory starts from some S' and by condition I.(ii) its interpretation must be a formal trajectory starting from $m(S')$. Conversely, every formal trajectory starts from some S, by condition III.(ii) of weak bisimulation there is some S' with $m(S') = S$, and by condition I.(ii) of trajectory equivalence there is an implementation trajectory starting from S' whose interpretation is that formal trajectory. □

Applying Bisimulation

We now consider how to use bisimulation to analyze our example implementation of $A+B \rightarrow C+D$, as shown in Figure 2.2. We use the three conditions formulation. The atomic condition is satisfied by the "signal species" x_A, x_B, x_C, and x_D. For the delimiting condition, we check each implementation reaction individually: $i_A + x_B \rightarrow t_{CD} + w_1$ is interpreted as $A+B \rightarrow C+D$, which is formal, while $x_A \rightleftharpoons i_A$ and $t_{CD} \rightarrow x_C + x_D + w_2$ are trivial. The permissive condition says that for every formal reaction and for every implementation state in which that reaction should be able to happen, it can. There is one formal reaction, $A + B \rightarrow C + D$, and any state in which it should be able to happen must contain an x_B and either an x_A or i_A, since those are the only species whose interpretations contain either B and/or A. If the state contains x_B and i_A, then the reaction $i_A + x_B \rightarrow t_{CD} + w_1$ can happen and satisfies the permissive condition. If the state contains x_B and x_A, then the trivial reaction $x_A \rightarrow i_A$ followed by $i_A + x_B \rightarrow t_{CD} + w_1$ satisfies the permissive condition.

Now consider a different case. Figure 2.5 shows an implementation of $A+B \rightarrow C + D$ as described by Qian et al. [57] as a means to implement stack machines, along with a natural interpretation. The species $i_{AB:CD}$ is interpreted as $C + D$, while $i_{A:BCD}$ is interpreted as A and x_B as B. This makes the reaction $i_{AB:CD} \rightarrow i_{A:BCD} + x_B$ interpreted as $C + D \rightarrow A + B$, which is not a valid formal reaction. Thus the delimiting condition is unsatisfied, and the implementation is not correct according to bisimulation. Although we only show one candidate interpretation, any interpretation will have the same problem provided $m(x_A) = \{\!|A|\!\}$, $m(x_B) = \{\!|B|\!\}$, $m(x_C) = \{\!|C|\!\}$, and $m(x_D) = \{\!|D|\!\}$. The core of the problem in this implementation is that the irreversible step in the pathway happens only after x_C and x_D are released, so there exist trajectories in which one or both is released and then the pathway reverses, producing x_A and x_B again. When analyzed with bisimulation, such a system will (in the closest possible interpretation) have some step that is interpreted as $A + B \to C + D$ but is also reversible, in which case the reverse reaction will be interpreted as $C + D \to A + B$, which is the problem described above. While this does not lead to a problem for the specific construction of deterministic stack machines used in Qian et al. [57], it does identify an error with this as a translation scheme for arbitrary CRNs: if the reaction $A + B \to C + D$ were put together with a reaction $C \to C + E$, then it would be possible to go from $\{\!|A, B|\!\}$ to $\{\!|A, B, E|\!\}$ in the implementation CRN when it is not possible to do so in the formal CRN. As an aside, the inevitability of a reverse reaction interpreted as $C + D \to A + B$ brings up the question of whether this construction would be a correct implementation of a formal CRN with the *reversible* reaction $A + B \rightleftharpoons C + D$. In fact the construction in Figure 2.5 would not, since although the delimiting condition would be satisfied, the reaction $C + D \to A + B$ cannot take place starting from $x_C + x_D$, failing the permissive condition: the reaction needs $x_C + x_D + i_{ABCD:}$ to reverse itself, but $i_{ABCD:}$ is not a fuel. However, Qian et al. designed a construction intended to implement reversible reactions, also presented in [57], which when enumerated produces a similar CRN with $i_{ABCD:}$ replaced by a fuel $f_{ABCD:}$ and no $i_{ABCD:} + f_i \to w_{ABCD}$ reaction, and this construction is correct according to bisimulation.

A given CRN can be a correct implementation of more than one formal CRN. Given an interpretation, there is only one possible formal CRN for which *that interpretation* might be a bisimulation, but a given implementation CRN may have multiple interpretations to multiple formal CRNs, more than one of which may be correct bisimulations with an appropriate formal CRN. At the extremes, every CRN is an implementation of itself where $m(x) = \{\!|x|\!\}$ for all species x is a bisimulation, and every CRN is an implementation of the CRN with 0 species and 0 reactions where $m(x) = \emptyset$ for all x is a bisimulation. As a more interesting example, consider the implementation CRN shown in Figure 2.6(A). The CRN has 16 species $x_{i,j}$ for $1 \leq i, j \leq 4$, with reactions $x_{i,j} \rightleftharpoons x_{i+1,j}$ and $x_{i,j} \rightleftharpoons x_{i,j+1}$ for all appropriate i, j. For now, consider a formal CRN with 4 species $\{N, S, E, W\}$ (with unspecified reactions), with the constraint $m(x_{1,1}) = \{\!|W|\!\}$, $m(x_{4,1}) = \{\!|S|\!\}$, $m(x_{1,4}) = \{\!|N|\!\}$, and



$$x_A + f_{ABCD} \rightleftharpoons i_{A:BCD} + f_A^+$$

$$x_B + i_{A:BCD} \rightleftharpoons i_{AB:CD} + f_B^+$$

$$i_{AB:CD} + f_C^- \rightleftharpoons i_{ABC:D} + x_C$$

$$i_{ABC:D} + f_D^- \rightleftharpoons i_{ABCD:} + x_D$$

$$i_{ABCD:} + f_i \rightarrow w_{ABCD}$$

Figure 2.5: The translation scheme from [57], when used as a general CRN implementation, violates the delimiting condition. Species named f are fuels. Top: DNA Strand Displacement diagram of the reversible reaction $x_B + i_{A:BCD} \rightleftharpoons i_{AB:CD} + f_B^+$ with interpretation of involved species given. With the closest possible interpretation (shown), the reverse reaction (highlighted) is interpreted as $C + D \rightarrow A + B$, which violates the delimiting condition when the only formal reaction is $A + B \rightarrow C + D$. Bottom: List of enumerated reactions of the full DSD system for $A + B \rightarrow C + D$, before removing fuels, with the violating reaction $x_B + i_{A:BCD} \rightleftharpoons i_{AB:CD} + f_B^+$ highlighted. Although only one candidate interpretation is shown to be invalid in this figure, any other interpretation either has the same problem or is invalid for other reasons.

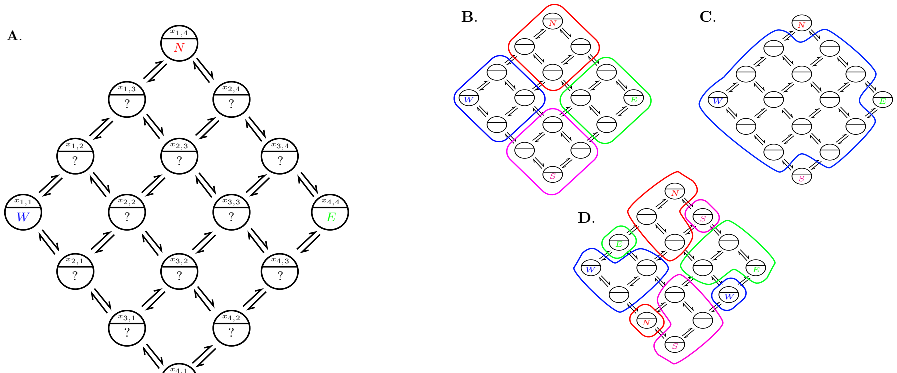


Figure 2.6: An example implementation CRN with multiple possible correct interpretations as different formal CRNs. Here each circle represents one implementation species, with the name of the species above the bar and its interpretation below the bar; a "?" means the interpretation is not yet specified. **A.** The implementation CRN with constraints on the interpretation of 4 of its species. **B.,C.** Two correct interpretations as two distinct formal CRNs. **D.** An interpretation not correct for any formal CRN. For **B., C.,** and **D.,** species in the same colored circle have the same interpretation.

$m(x_{4,4}) = \{\!|E|\!\}$. Figure 2.6(B,C,D) shows three possible interpretations for this implementation CRN. The first one is a valid bisimulation for the formal CRN $\{W \rightleftharpoons N, W \rightleftharpoons S, N \rightleftharpoons E, S \rightleftharpoons E\}$, while the second is a valid bisimulation for $\{W \rightleftharpoons S, W \rightleftharpoons E, W \rightleftharpoons N\}$. The third interpretation is not a valid bisimulation for any formal CRN: the only formal CRN for which the atomic and delimiting conditions are satisfied is $\{N \rightleftharpoons W, N \rightleftharpoons E, N \rightleftharpoons S, W \rightleftharpoons E, W \rightleftharpoons S, E \rightleftharpoons S\}$, but then the permissive condition is not satisfied for (e.g.) the reaction $E \to S$ from initial $x_{1,2}$. $m(x_{1,2}) = E$ but $x_{1,2}$ cannot turn into anything that is interpreted as S without passing through something that is interpreted as either W or N, which would be a nontrivial reaction; and the permissive condition requires that a formal reaction be implemented by a sequence of implementation reactions of which all but the last are trivial. The interested reader may enjoy trying to show that there is no interpretation of this implementation CRN which is a valid bisimulation for the formal CRN $\{N \rightleftharpoons W, N \rightleftharpoons E, N \rightleftharpoons S, W \rightleftharpoons E, W \rightleftharpoons S, E \rightleftharpoons S\}$, unless the constraint on the interpretation of the corner species is relaxed.

In many implementations, species with $m(z) = \emptyset$ play a role. We call those species "null species". One type of null species is what theories such as pathway decomposition [64] would call “waste species”: implementation species that never appear as a reactant unless all other species involved in the reaction are also waste species. This allows the waste species to be ignored once produced. The species w_1 and w_2 in the example above from Soloveichik et al. [66] (Figure 2.1) are examples of this type of null species, which are usually easy to handle with bisimulation.

However, not all null species have to be waste species. Consider the following formal and implementation CRNs:

$$\begin{array}{ll}
\mathcal{S} = \{A, B\} & \mathcal{S}' = \{x_A, y_A, z, x_B\} \\
 & \\
A \rightarrow B & x_A \rightarrow y_A \\
 & y_A \rightarrow x_A + z \\
 & x_A + 3z \rightarrow x_B
\end{array}$$

Let $m(x_A) = m(y_A) = A$, $m(x_B) = B$, $m(z) = \emptyset$. The only reaction which implements $A \rightarrow B$ is $x_A + 3z \rightarrow x_B$, which requires the null species z. So starting from either x_A or y_A, to implement $A \rightarrow B$ the system loops $x_A \rightarrow y_A$ and $y_A \rightarrow x_A + z$ as many times as needed, then does $x_A + 3z \rightarrow x_B$. Since any state whose interpretation has an A must have either x_A or y_A, this proves the permissive condition is satisfied. The atomic condition is satisfied by $m(x_A) = A$ and $m(x_B) = B$, and the delimiting condition is satisfied since the first two implementation reactions are trivial and the third is $A \rightarrow B$. Thus this is an example of a correct interpretation with a non-waste null species.

As a final example of the effects of null species, consider a pair of CRNs similar to the previous example:

$$\begin{array}{ll}
\mathcal{S} = \{A, B\} & \mathcal{S}' = \{x_A, y_A, z, x_B\} \\
 & \\
A \rightarrow B & x_A \rightarrow y_A + z \\
 & y_A + z \rightarrow x_B \\
B \rightarrow A & x_B \rightarrow x_A
\end{array}$$

If we consider the same interpretation as the previous CRN pair, $m(x_A) = m(y_A) = A$, $m(z) = \emptyset$, $m(x_B) = B$, then the permissive condition is not satisfied. In the implementation state $\{\!|y_A|\!\}$, since $m(\{\!|y_A|\!\}) = \{\!|A|\!\}$ the reaction $A \to B$ should be possible, but in the implementation CRN no reactions are possible. (In fact, no correct interpretation exists.) Intuitively, this CRN tried to implement $A \to B$ where the combination $y_A + z$ represents one copy of A; in fact, the theory of pathway decomposition [64] would take this view, with x_A and x_B identified as formal species A and B and with y_A and z considered intermediate species, because pathway decomposition only considers initial states consisting exclusively of formal species when evaluating correct behavior. However, our use of interpretations requires that every implementation species must have a meaning on its own, and counts implementations that rely on combinations having meaning as invalid.

One example of CRN comparison, as discussed in [7], is when a larger CRN contains multiple copies of a smaller CRN, and we want to consider the larger CRN as an implementation of the smaller CRN. As a naive example, consider just two copies of a correct implementation as the implementation CRN:

$$\mathcal{S} = \{A, B\} \qquad\qquad \mathcal{S}' = \{x_A, y_A, x_B, y_B\}$$

$$\begin{aligned} A \to B \qquad\qquad & x_A \to x_B \\ & y_A \to y_B \end{aligned}$$

This example, with $m(x_A) = m(y_A) = A$ and $m(x_B) = m(y_B) = B$, is a correct implementation. However, when we try to do the same thing with bimolecular reactions, problems arise. Consider:

$$\mathcal{S} = \{A, B, C\} \qquad\qquad \mathcal{S}' = \{x_A, y_A, x_B, y_B, x_C, y_C\}$$

$$\begin{aligned} A + B \to C \qquad\qquad & x_A + x_B \to x_C \\ & y_A + y_B \to y_C \end{aligned}$$

If $m(x_A) = m(y_A) = A$, $m(x_B) = m(y_B) = B$, and $m(x_C) = m(y_C) = C$, then the permissive condition is not satisfied: the state $\{\!|x_A, y_B|\!\}$ is interpreted as

$\{\!|A, B|\!\}$, and thus should be able to implement $A + B \to C$, but in fact no reactions are possible and it cannot. This sort of problem is the motivation for the modularity condition which we describe soon, and the reason why checking the permissive condition is difficult, as discussed in Section 2.5. One solution, in terms of implementation CRN design, is to allow every possible combination of reactants to any reaction to interact:

$$\mathcal{S} = \{A, B, C\} \qquad \mathcal{S}' = \{x_A, y_A, x_B, y_B, x_C, y_C\}$$

$$\begin{aligned} A + B \to C \qquad & x_A + x_B \to x_C \\ & x_A + y_B \to x_C \\ & y_A + x_B \to y_C \\ & y_A + y_B \to y_C \end{aligned}$$

This, with the same interpretation as above, is now a correct implementation. However, it can scale poorly: if a formal reaction has k reactants and each one has n possible implementation species, then there are roughly n^k combinations which each need their own reaction. A better-scaling way is to allow different implementation species to interconvert:

$$\mathcal{S} = \{A, B, C\} \qquad \mathcal{S}' = \{x_A, y_A, x_B, y_B, x_C, y_C\}$$

$$\begin{aligned} A + B \to C \qquad & x_A + x_B \to x_C \\ & y_A + y_B \to y_C \\ & x_B \rightleftharpoons y_B \end{aligned}$$

Again using the same interpretation, this system is a correct implementation. This approach is the one expected by the modularity condition, which helps the scaling behavior.

Properties of CRN Bisimulation

We describe two properties of CRN bisimulation that are likely to be useful when analyzing larger systems. While bisimulation in the classic sense is an equivalence relation between systems [52], our definition of interpretation-dependent CRN bisimulation is a partial order on the set of CRNs. In particular, CRN bisimulation is transitive, which allows us to do complex proofs of correctness in stages. We also show a modularity condition, where the combination of interpretations can be verified using only properties of each individual interpretation. This is particularly useful for general translation schemes where the translation of a whole CRN is the combination of one "module" for each reaction. As an example, we use modularity to prove that the translation scheme in [66] is correct for any CRN.

We first show that CRN bisimulation is transitive. Consider three CRNs: an abstract CRN $(\mathcal{S}, \mathcal{R})$, an implementation CRN $(\mathcal{S}'', \mathcal{R}'')$, and an intermediate CRN $(\mathcal{S}', \mathcal{R}')$. For example, $(\mathcal{S}, \mathcal{R})$ is an abstract CRN, $(\mathcal{S}'', \mathcal{R}'')$ is a low-level reaction enumeration of a prospective DNA implementation of $(\mathcal{S}, \mathcal{R})$, and $(\mathcal{S}', \mathcal{R}')$ is a more high-level reaction enumeration of the same DNA implementation, which abstracts away from certain details. Say we have proven that $(\mathcal{S}', \mathcal{R}')$ is a valid implementation of $(\mathcal{S}, \mathcal{R})$ by finding an interpretation $m_1 : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ that is a bisimulation, and similarly have found an interpretation $m_2 : \mathcal{S}'' \to \mathbb{N}^{\mathcal{S}'}$ that is a bisimulation from $(\mathcal{S}'', \mathcal{R}'')$ to $(\mathcal{S}', \mathcal{R}')$. We want to prove that $(\mathcal{S}'', \mathcal{R}'')$, the system we actually have, is a valid implementation of $(\mathcal{S}, \mathcal{R})$, the system we want. The natural interpretation $m : \mathcal{S}'' \to \mathbb{N}^{\mathcal{S}}$ is $m(x) = (m_1 \circ m_2)(x) = m_1(m_2(x))$, treating m_2 as a function of species and m_1 as extended to a function of states. It turns out that this interpretation is in fact a bisimulation.

Lemma 2.4.1. *(Transitivity) If m_2 is a bisimulation from $(\mathcal{S}'', \mathcal{R}'')$ to $(\mathcal{S}', \mathcal{R}')$ and m_1 is a bisimulation from $(\mathcal{S}', \mathcal{R}')$ to $(\mathcal{S}, \mathcal{R})$, then $m = m_1 \circ m_2$ is a bisimulation from $(\mathcal{S}'', \mathcal{R}'')$ to $(\mathcal{S}, \mathcal{R})$.*

Proof. We use the three conditions formulation of correctness. We refer to $(\mathcal{S}, \mathcal{R})$ as the "formal" CRN, $(\mathcal{S}'', \mathcal{R}'')$ as the "implementation" CRN, and $(\mathcal{S}', \mathcal{R}')$ as the "intermediate" CRN. We show that each condition for m follows from the corresponding conditions for m_1 and m_2.

For any formal species A, by the atomic conditions for m_1 and m_2 there is an intermediate species x_A with $m_1(x_A) = \{\!|A|\!\}$ and implementation species y_A with $m_2(y_A) = x_A$. Then $m(y_A) = m_1(m_2(y_A)) = m_1(\{\!|x_A|\!\}) = \{\!|A|\!\}$, thus m satisfies the atomic condition.

For any implementation reaction $r'' = R'' \to P''$, by the delimiting condition for m_2 its interpretation $m_2(r'')$ is either an intermediate reaction $R' \to P' \in \mathcal{R}'$ or is τ. If $m_2(r'') = \tau$, that means $m_2(R'') = m_2(P'')$ and $m(R'') = m_1(m_2(R'')) = m_1(m_2(P'')) = m(P'')$, so $m(R'' \to P'') = m(r'') = \tau$. If $m_2(r'') = R' \to P'$ is a valid intermediate reaction, then $m(r'') = m_1(R' \to P')$, which by the delimiting condition for m_1 is either a valid formal reaction or trivial.

For any formal state S and reaction r with $S \xrightarrow{r}$ and any implementation state S'' with $m(S'') = S$, that means $S' = m_2(S'')$ is an intermediate state with $m_1(S') = S$. By the permissive condition on m_1, there is some r' with $m_1(r') = r$ and $S' \overset{r'}{\Rightarrow}$. Using the permissive condition on m_2 and the argument used in Theorem 2.4.1 to show that the permissive condition implies trajectory equivalence, there is a sequence of implementation reactions starting from S'' which implements the intermediate trajectory by which $S' \overset{r'}{\Rightarrow}$. This means that one of those reactions r'' has $m_2(r'') = r'$, some of them interpret via m_2 to various intermediate reactions in that pathway that are trivial under m_1, and the rest of which are trivial under m_2. An implementation reaction trivial under m_2 is trivial under m, as is a reaction which interprets under m_2 to an intermediate reaction trivial under m_1, thus all reactions in this pathway except r'' are trivial under m, so when viewed under m, $S'' \overset{r}{\Rightarrow}$. □

Lemma 2.4.2. *(Partial order) The relation $\gtrsim$ as described by the following is a partial order: $(\mathcal{S}', \mathcal{R}') \gtrsim (\mathcal{S}, \mathcal{R})$ if there exists an $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ which satisfies the atomic, delimiting, and permissive conditions with equality defined as $(\mathcal{S}', \mathcal{R}') \equiv (\mathcal{S}, \mathcal{R})$ if there exists a bijection $n : \mathcal{S}' \to \mathcal{S}$ such that $(n(\mathcal{S}'), n(\mathcal{R}')) = (\mathcal{S}, \mathcal{R})$ where n is extended naturally to sets and reactions.*

Proof. A partial order must be transitive, reflexive, and anti-symmetric. Transitivity (if $a \leq b$ and $b \leq c$ then $a \leq c$) follows immediately from Lemma 2.4.1. Reflexivity ($a \leq a$) is obvious by letting m be the identity function. It remains to show anti-symmetry (if $a \leq b$ and $b \leq a$, then $a = b$), i.e. that given $(\mathcal{S}_1, \mathcal{R}_1)$ and $(\mathcal{S}_2, \mathcal{R}_2)$ with $m_1 : \mathcal{S}_1 \to \mathbb{N}^{\mathcal{S}_2}$ and $m_2 : \mathcal{S}_2 \to \mathbb{N}^{\mathcal{S}_1}$ that each satisfy the atomic, delimiting, and permissive conditions, $(\mathcal{S}_1, \mathcal{R}_1)$ and $(\mathcal{S}_2, \mathcal{R}_2)$ are identical up to a change of species names. The atomic condition implies that $|\mathcal{S}_1| \leq |\mathcal{S}_2|$ and $|\mathcal{S}_2| \leq |\mathcal{S}_1|$, thus the numbers of species are equal and in particular m_1 is a bijection from species in $\mathcal{S}_1$ to sets of exactly one species in $\mathcal{S}_2$ (and the same is true for m_2). To simplify notation, we let $n(x) = y$ if $m_1(x) = \{\!|y|\!\}$; n must be a bijection from $\mathcal{S}_1$ to $\mathcal{S}_2$. (If the CRN has sufficient symmetry, it is not necessarily true that $m_2(n(x)) = \{\!|x|\!\}$, for example if both CRNs are $\{A \to C, B \to C\}$ we could have $m_2(n(A)) = \{\!|B|\!\}$.) Since n is a bijection, any reaction that would be trivial after interpretation (by either m_1 or m_2) must be trivial before interpretation, and thus cannot exist. By the delimiting condition for m_1, every reaction in $\mathcal{R}_1$ must have its image under n in $\mathcal{R}_2$; by the permissive condition for m_1, every reaction in $\mathcal{R}_2$ must have its preimage under n in $\mathcal{R}_1$; thus the CRNs are equal up to isomorphism n. □

This result stands in contrast to the definition of bisimulation in transition systems, which is an equivalence relation on states that can be extended to an equivalence relation on systems [52]. We discuss this difference further in Section 2.6.

In Section 2.4 we showed that the translation scheme from [66] is a correct implementation of the single reaction $A + B \to C + D$ according to CRN bisimulation. Intuitively, given a CRN of multiple reactions we should be able to combine the implementations of each such reaction to form a correct implementation of the CRN. In particular, we would like to show that the combined implementation CRN is correct using a condition we can check on each individual reaction's implementation without having to check any property of the combined CRN. Since, as we will see in Section 2.5, the time required to check an interpretation scales much worse than linearly in the size of the implementation CRN, such a modularity condition would be a significant saving in the time required. While it is not in general true that combining two correct implementation CRNs gives a correct implementation of the combined formal CRN, there is a modularity condition which guarantees that the combined CRN is correct.

We consider an implementation CRN $(\mathcal{S}'_1, \mathcal{R}'_1)$ and formal CRN $(\mathcal{S}_1, \mathcal{R}_1)$ with interpretation $m_1 : \mathcal{S}'_1 \to \mathbb{N}^{\mathcal{S}_1}$, and another implementation CRN $(\mathcal{S}'_2, \mathcal{R}'_2)$ and formal CRN $(\mathcal{S}_2, \mathcal{R}_2)$ with interpretation $m_2 : \mathcal{S}'_2 \to \mathbb{N}^{\mathcal{S}_2}$, where both m_1 and m_2 are bisimulations. We assume the interpretations are compatible: for each $x \in \mathcal{S}'_1 \cap \mathcal{S}'_2$, $m_1(x) = m_2(x)$, which implies $m_1(x) \in \mathbb{N}^{\mathcal{S}_1 \cap \mathcal{S}_2}$. We also assume that if the species involved are real molecules (for example, in a DNA Strand Displacement system), the reactions in $\mathcal{R}'_1$ and $\mathcal{R}'_2$ are the only reactions that occur when you combine the implementation species in $\mathcal{S}'_1$ and $\mathcal{S}'_2$; that is, we assume no crosstalk reactions. Whether there is crosstalk in a DSD system can be checked by a reaction enumerator [36, 45], but is beyond the scope of this theory. (Although if crosstalk reactions do occur, but all of them satisfy the delimiting condition, the remainder of this discussion will still hold.) Aside from crosstalk, the main reason for the combined implementation to be incorrect according to bisimulation is a failure of the permissive condition. If some implementation species y in e.g. $\mathcal{S}'_1$ but not in $\mathcal{S}'_2$ has an interpretation that contains a formal species $A \in \mathcal{S}_1 \cap \mathcal{S}_2$, there may be some formal reaction in $\mathcal{R}_2$ with A as a reactant that cannot be implemented from an implementation state where y is the representation of A, in which case the permissive condition is false. (For example, if in Figure 2.7 the reaction $i_{1:1} \to x_A$ was not present, then the interpretation m_1 would still be a correct bisimulation, but the combined interpretation m would fail the permissive condition for where formal reaction $C + A \to B + D$ cannot be implemented from implementation state $\{\!|x_C, i_{1:1}|\!\}$.) If any such species y can, via trivial reactions, "release" any formal species in $\mathcal{S}_1 \cap \mathcal{S}_2$ in its interpretation to implementation species in $\mathcal{S}'_1 \cap \mathcal{S}'_2$, then we would think this problem cannot arise. This condition can be checked individually on each module without checking the combined CRN, and we show that this condition guarantees that the combined implementation is correct according to bisimulation. An example of a modular implementation CRN is shown in Figure 2.7.

Definition 2.4.3 (Modularity Condition)**.** Let m be a bisimulation from $(\mathcal{S}', \mathcal{R}')$ to $(\mathcal{S}, \mathcal{R})$. Let $\mathcal{S}'_0 \subset \mathcal{S}'$ and $\mathcal{S}_0 \subset \mathcal{S}$ be subsets of implementation and formal species, respectively, where $y \in \mathcal{S}'_0 \Rightarrow m(y) \subset \mathcal{S}_0$. We say that m is a *modular interpretation* with respect to the *common (implementation and formal) species* $\mathcal{S}'_0$ and $\mathcal{S}_0$ if for any $x \in \mathcal{S}'$ there is a sequence of trivial reactions $\{\!|x|\!\} \overset{\mathcal{T}}{\Rightarrow} Y + Z$ where $Y \subset \mathcal{S}'_0$ and $m(Z) \cap \mathcal{S}_0 = \emptyset$, i.e. all common formal species in the interpretation of x are extracted as common implementation species.

Theorem 2.4.2. *(Modularity) Let m_1 be a bisimulation from $(\mathcal{S}'_1, \mathcal{R}'_1)$ to $(\mathcal{S}_1, \mathcal{R}_1)$ and m_2 be a bisimulation from $(\mathcal{S}'_2, \mathcal{R}'_2)$ to $(\mathcal{S}_2, \mathcal{R}_2)$ where m_1 and m_2 agree on $\mathcal{S}'_1 \cap \mathcal{S}'_2$. Let $\mathcal{S}' = \mathcal{S}'_1 \cup \mathcal{S}'_2$, $\mathcal{R}' = \mathcal{R}'_1 \cup \mathcal{R}'_2$, $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, and $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, and $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ equal m_1 on $\mathcal{S}'_1$ and m_2 on $\mathcal{S}'_2$. If m_1 and m_2 are both respectively modular bisimulations with respect to the common implementation species $\mathcal{S}'_1 \cap \mathcal{S}'_2$ and common formal species $\mathcal{S}_1 \cap \mathcal{S}_2$, then m is*

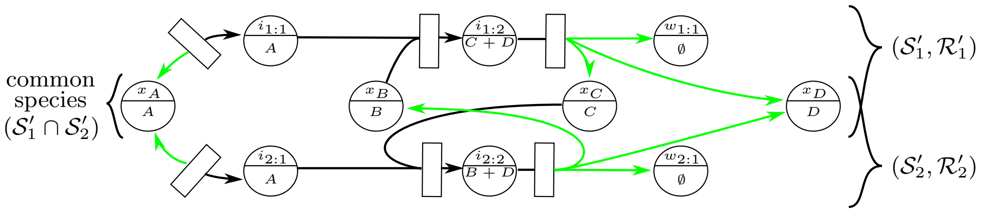


Figure 2.7: An implementation CRN that satisfies the modularity condition. Circles represent implementation species with interpretation given below the line, and boxes with arrows represent reactions, with reactants on one side of the box and products on the other; boxes where arrows point both ways are reversible reactions. Here the top CRN $(\mathcal{S}_1', \mathcal{R}_1')$ is a correct implementation of the formal reaction $A + B \to C + D$, and the bottom CRN $(\mathcal{S}_2', \mathcal{R}_2')$ a correct implementation of $C + A \to B + D$. Green arrows indicate reactions used to satisfy the modularity condition in Definition 2.4.3; for example, $i_{1:2} \overset{\mathcal{T}}{\Rightarrow} Y + Z$ by reaction $i_{1:2} \to x_C + x_D + w_{1:1}$, where $Y = \{\!|x_C + x_D|\!\} \subset \mathcal{S}_1' \cap \mathcal{S}_2'$ and $Z = \{\!|w_{1:1}|\!\}$ has $m(Z) = \emptyset$. Thus the combined implementation CRN is a correct implementation of the combined formal CRN. If the reverse reaction $i_{1:1} \to x_A$ did not exist, $(\mathcal{S}_1', \mathcal{R}_1')$ would still be a correct implementation of $A + B \to C + D$, but the combined CRN would not satisfy the permissive condition, since state $\{\!|x_C, i_{1:1}|\!\}$ cannot implement $C + A \to B + D$ without that reaction.

a bisimulation from $(\mathcal{S}', \mathcal{R}')$ to $(\mathcal{S}, \mathcal{R})$, and m is also modular with respect to $\mathcal{S}_1' \cap \mathcal{S}_2'$ and $\mathcal{S}_1 \cap \mathcal{S}_2$.

Proof. We use the three conditions formulation. The atomic condition for m for each formal species A is satisfied by the species x_A that satisfy it for m_1 or m_2, as appropriate, or possibly both; e.g. if $A \in \mathcal{S}_1$ then there is some species $x_A \in \mathcal{S}_1'$ such that $m_1(x_A) = \{\!|A|\!\}$, which implies that $x_A \in \mathcal{S}'$ and $m(x_A) = m_1(x_A) = \{\!|A|\!\}$. Similarly the delimiting condition for m follows from that for m_1 and m_2: for any implementation reaction $R' \to P'$ in $\mathcal{R}'$, that reaction is in either $\mathcal{R}_1'$ or $\mathcal{R}_2'$ (the proof still holds if in both), and its interpretation in m agrees with its interpretation in either m_1 or m_2 as appropriate, which is either a trivial reaction or a formal reaction in $\mathcal{R}_1$ or $\mathcal{R}_2$, which is thus in $\mathcal{R}$.

For the permissive condition, consider a formal reaction $r = R \to P$ and implementation state S' where $R \leq m(S')$. Either $r \in \mathcal{R}_1$ or $r \in \mathcal{R}_2$; without loss of generality say $r \in \mathcal{R}_1$ (where again, the proof still holds if also $r \in \mathcal{R}_2$). Divide S' into species in the first CRN and species not: let $S' = S_1' + S_2'$, where

$S_1' \subset \mathcal{S}_1'$ and $S_2' \cap \mathcal{S}_1' = \emptyset$. If $m(S_1') \geq R$, then the permissive condition for m_1 applied to reaction r and state S_1' mean $S_1' \overset{r}{\Rightarrow}$, thus $S' \overset{r}{\Rightarrow}$ by the same sequence of reactions ignoring species in S_2'. In the general case, this means the proof is nontrivial only for formal species in R whose implementations in S' are in S_2', and we need to show that those formal species can be "extracted" into an implementation species in $\mathcal{S}_1'$. This is exactly the modularity condition: for each species $x_i \in S_2'$ there is a sequence of trivial reactions by which $x_i \overset{\tau}{\Rightarrow} Y_i + Z_i$, where $Y_i \subset \mathcal{S}_1'$ and $m(Z_i) \cap \mathcal{S}_1 = \emptyset$. In particular, since $R \to P$ is a reaction in $\mathcal{R}_1$, $R \subset \mathcal{S}_1$ and $R \cap m(Z_i) = \emptyset$. We then have $S' \overset{\tau}{\Rightarrow} S_1' + Y + Z$, where $Y = \sum_i Y_i \subset \mathcal{S}_1'$ and $Z = \sum_i Z_i$. Since $R \cap Z = \emptyset$, $R \leq m(S')$, and $m(S') = m(S_1' + Y) + m(Z)$, we have $R \leq m(S_1' + Y)$. Since $S_1' + Y \subset \mathcal{S}_1'$, the permissive condition for m_1 implies $S_1' + Y \overset{r}{\Rightarrow}$, thus $S' \overset{r}{\Rightarrow}$.

That m is modular with respect to $\mathcal{S}_1' \cap \mathcal{S}_2'$ and $\mathcal{S}_1 \cap \mathcal{S}_2$ also follows simply from the same properties for m_1 and m_2, and the fact that if $m_1(r') = \tau$ or $m_2(r') = \tau$ then $m(r') = \tau$. For any $x \in \mathcal{S}'$, there is a sequence of trivial (under m) reactions by which $x \overset{\tau}{\Rightarrow} Y + Z$, for $Y \subset \mathcal{S}_1' \cap \mathcal{S}_2'$ and $m(Z) \cap \mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$: if $x \in \mathcal{S}_1'$, then such a sequence follows from the modularity of m_1, and if $x \in \mathcal{S}_2'$, from the modularity of m_2. □

DNA implementation schemes for arbitrary CRNs such as [66], [57], and [6] typically have a set of common species and for each formal reaction a "module" with additional species and implementation reactions that implement the formal reaction. If the modules have no crosstalk and each one correctly implements its reaction and satisfies the modularity condition, then repeated applications of Theorem 2.4.2 prove that the entire CRN is a correct implementation.

Corollary 2.4.3. *Consider a formal CRN $(\mathcal{S}, \mathcal{R})$ with n reactions $\mathcal{R} = \{r_i\}_{i=1}^n$, and n implementation "module" CRNs $(\mathcal{S}_0' \cup \mathcal{S}_i', \mathcal{R}_i')$ with species $\mathcal{S}_0'$ in common, where any $\mathcal{S}_i'$ is disjoint from any $\mathcal{S}_j'$ for $0 \leq i < j \leq n$. If there are interpretations $m_i : \mathcal{S}_i' \to \mathcal{S}$ for $0 \leq i \leq n$ such that the interpretation $(m_0 \cup m_i)$ is a bisimulation from $(\mathcal{S}_0' \cup \mathcal{S}_i', \mathcal{R}_i')$ to $(\mathcal{S}, \{r_i\})$ and is modular with respect to the common implementation species $\mathcal{S}_0'$ and common formal species $\mathcal{S}$, then $m = \bigcup_{i=1}^n m_i$ is a bisimulation from $(\mathcal{S}_0' \cup \bigcup_{i=1}^n \mathcal{S}_i', \bigcup_{i=1}^n \mathcal{R}_i')$ to $(\mathcal{S}, \mathcal{R})$.*

In particular, the translation scheme from [66] discussed earlier satisfies the condition in Corollary 2.4.3 for $\mathcal{S}_0' = \{x_A \mid A \in \mathcal{S}\}$, i.e. the signal species.

Note that formally, each module contains all signal species—even ones that do not appear in reactions in that module. For example, if the formal CRN has reactions $A+B \to C$ and $A+D \to B$, then the signal species $x_D \in \mathcal{S}'_0$ appears as an implementation species in the module corresponding to $A+B \to C$ but does not appear in any reaction in that module. Although counterintuitive, our theory works fine when some species do not appear in any reactions. Thus Corollary 2.4.3 proves that for any number of formal reactions, the scheme in [66] produces a correct implementation CRN, as long as the DSD reaction enumerator produces exactly the described reactions and no additional crosstalk reactions.

2.5 Checking Bisimulation

We now have a definition of "correct implementation", and can sometimes prove that a particular implementation is or is not correct. We would like to find a general way to check whether any implementation is correct.

We divide "checking bisimulation" into three questions. First, given a formal and implementation CRN and an interpretation, is the interpretation a bisimulation? Second, if (as in most engineered CRN implementations) we have a formal CRN, implementation CRN, and for each formal species A a designated signal species x_A, is there an interpretation which is a bisimulation and has $m(x_A) = \{\!| A |\!\}$? Finally, given a formal CRN, implementation CRN, and no additional information, is there an interpretation which is a bisimulation?

For complexity purposes, we define the size of a CRN $(\mathcal{S}, \mathcal{R})$ as

$$|(\mathcal{S}, \mathcal{R})| = |\mathcal{S}| + \sum_{X \in \mathcal{S}, R \to P \in \mathcal{R}} \left(\lceil \log(R(X)+1) \rceil + \lceil \log(P(X)+1) \rceil \right).$$

This corresponds roughly (up to polynomial factors) to writing the number of species in unary (to cover edge cases involving species not appearing in any reaction), then writing each reaction in the usual chemical notation (e.g. $5X + 3Y \to Z + 2X$). Similarly, for an interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ we define

$$|m| = \sum_{x \in \mathcal{S}'} \sum_{X \in \mathcal{S}} \lceil \log(m(x)(X)+1) \rceil,$$

which corresponds roughly to writing out each interpretation e.g. $m(x_3) = 3A + B$. We find that the complexity of our algorithms is best expressed in terms of three parameters: the *size* $n = |\mathcal{S}| + |\mathcal{R}| + |\mathcal{S}'| + |\mathcal{R}'|$, the number of species and reactions in the CRNs; the *arity* $k = \max_{R \to P \in \mathcal{R}} \sum_{X \in \mathcal{S}} R(X)$, the maximum number of reactants in any formal reaction; and the *max stoichiometry* $s = \max_{(R \to P) \in \mathcal{R} \cup \mathcal{R}', X \in \mathcal{S} \cup \mathcal{S}'} R(X)$. Note that $|(\mathcal{S}, \mathcal{R})| + |(\mathcal{S}', \mathcal{R}')| \leq n^2 \lceil \log(s+1) \rceil$ and $k \leq s|\mathcal{S}|$, so any algorithm whose time or space complexity is (for example) polynomial in some combination of n, $\log s$, $\log k$, and $|m|$ is (for example) polynomial in $|(\mathcal{S}, \mathcal{R})| + |(\mathcal{S}', \mathcal{R}')| + |m|$. We find that some algorithms are less complex when k is bounded by a constant, such as $k = 2$ limiting the formal CRN to bimolecular reactions, and that the possibility that s is *not* bounded by a constant (in particular, when k is $\omega(n)$) affects the technical details of the proofs but not the result.

Checking an Interpretation

First we consider the problem of, given an interpretation, checking whether it is a bisimulation. We use the three conditions on an interpretation, having proved in Theorem 2.4.1 that they are equivalent to bisimulation and trajectory equivalence. Given two CRNs and an interpretation between them, the atomic and delimiting conditions are trivial to check. This leaves only the permissive condition.

Checking the permissive condition means, for each formal reaction $r = (R, P)$ and implementation state S' with $m(S') \geq R$, checking whether S' can reach via trivial reactions some state from which a reaction that is interpreted as r can happen. Although there are infinitely many such implementation states, we can find a finite set that is sufficient for checking the permissive condition. Figure 2.8 shows an example of such a set for the reaction $A + B \to C$ in a hypothetical implementation CRN.

Definition 2.5.1. Given a formal reaction $r = (R, P) \in \mathcal{R}$, we say that an implementation state S' is a *minimal implementation state* for r if $m(S') \geq R$ and there is no S'_0 with $S'_0 < S'$ and $m(S'_0) \geq R$. (Equivalently, S' is a minimal element—in the usual sense for partially ordered sets—of the set $\{S' \mid m(S') \geq R\}$, so we often say S' is "minimal for $m(S') \geq R$".) We use the notation $\mathcal{M}(r)$ for the set of minimal implementation states for a formal reaction r.

Lemma 2.5.1. *Let $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ be a formal and implementation CRN with interpretation m. The interpretation m satisfies the permissive condition if and only if, for every formal reaction r and every $S' \in \mathcal{M}(r)$, $S' \overset{r}{\Rightarrow}$.*

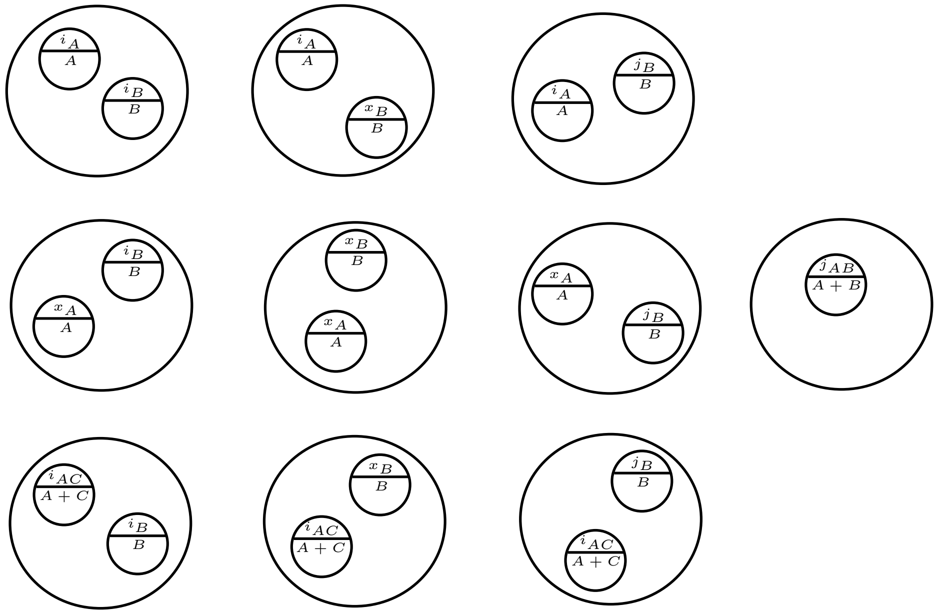


Figure 2.8: Example set of minimal implementation states for the reaction $A + B \to C$. Large circles represent minimal states; circles within minimal states represent implementation species, with their interpretation in formal species given below the line. Assuming that the 7 implementation species shown are the only species whose interpretation contains either an A or a B, any implementation state whose interpretation contains $A + B$ must contain one of the 10 minimal states shown. The set of minimal states does not depend on the implementation *reactions*, so no reactions are shown.

Proof. Clearly, if the permissive condition is satisfied, then for any formal reaction r and implementation state $S' \in \mathcal{M}(r)$, $m(S') \xrightarrow{r}$ so by the permissive condition, $S' \overset{r}{\Rightarrow}$. For the converse, assume for every $r \in \mathcal{R}$ and $S' \in \mathcal{M}(r)$ that $S' \overset{r}{\Rightarrow}$, and consider an arbitrary formal reaction $r = (R, P)$ and implementation state S'' with $m(S'') \xrightarrow{r}$, i.e. $m(S'') \geq R$. There is at least one minimal $S' \in \mathcal{M}(r)$ with $S'' \geq S'$, and by assumption $S' \overset{r}{\Rightarrow}$. Since $S'' \geq S'$, the same sequence of reactions can occur in S'', thus $S'' \overset{r}{\Rightarrow}$. Since this is true for every r and S'', the permissive condition is satisfied. □

Lemma 2.5.2. *Let $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ be a formal and implementation CRN with interpretation m, and $r = (R, P) \in \mathcal{R}$ a formal reaction. Let $n = |\mathcal{S}'|$ be* *the number of implementation species and $k = |R|$ the number of reactants of r. The number of minimal implementation states for r is at most n^k, and all such states can be enumerated in time* $\mathrm{poly}(n^k)$ *and space* $\mathrm{poly}(n, k)$. *When $k \geq n$, in particular the number of minimal implementation states for r is at most $2^{n \log k}$, and they can be enumerated in time* $2^{\mathrm{poly}(n, \log k)}$ *and space* $\mathrm{poly}(n, \log k)$.

Proof. We describe an algorithm to enumerate all implementation states S' minimal for $m(S') \geq R$ given R, then show that it has the desired complexity and correctly enumerates all minimal states.

If $R = \emptyset$, then the only minimal implementation state for $m(S') \geq R$ is $S' = \emptyset$. If not, choose an arbitrary formal species $A \in R$. For each implementation species x with $A \in m(x)$: Construct a multiset of formal species $Q_x = R \backslash m(x)$ by removing $m(x)$ from R, ignoring any species in $m(x)$ but not in R. Apply this algorithm recursively to enumerate all implementation states S'_x minimal for $m(S'_x) \geq Q_x$. For each such S'_x, the state $S' = S'_x + \{\!|x|\!\}$ has $m(S') \geq R$ and may be minimal. Check if it is minimal by, for each $y \in S'$, checking if $m(S' - \{\!|y|\!\}) \geq R$. If none of them are, then S' is minimal; print it and continue enumerating.

We now prove that the algorithm enumerates exactly all minimal states S' with $m(S') \geq R$. Since we check whether S' is minimal before printing it, the algorithm clearly does not enumerate any S' with $m(S') \geq R$ that is not minimal. (Without this check, the algorithm could generate a non-minimal implementation state. For example, let $R = \{\!|A, B|\!\}$ where there are implementation species x_1 with $m(x_1) = \{\!|A|\!\}$ and $m(x_2) = \{\!|A, B|\!\}$. The algorithm could first choose A from R and x_1 with $A \in m(x_1)$, generating $Q_{x_1} = \{\!|B|\!\}$, then when called recursively choose B from Q_{x_1} and x_2 with $B \in m(x_2)$, thus generating the state $S' = \{\!|x_1, x_2|\!\}$ which is not minimal, since $S' > \{\!|x_2|\!\}$ and $m(\{\!|x_2|\!\}) = \{\!|A, B|\!\} \geq R$.) That every enumerated S' has $m(S') \geq R$ can be proven by induction on $|R|$. If $|R| = 0$, i.e. $R = \emptyset$, then $m(\emptyset) = \emptyset$ is trivially true. If not, then for each x with $A \in m(x)$ iterated through, $|Q_x| < |R|$ so by assumption each generated S'_x has $m(S'_x) \geq Q_x$. Then the S' generated from that S'_x has $m(S') = m(S'_x) + m(x) \geq Q_x + (m(x) \wedge R) \geq R$. Finally, to prove that every minimal state is enumerated, we again use induction on $|R|$, with the case $|R| = 0$ having only one minimal state, $S' = \emptyset$, which is generated. When $|R| > 0$, consider an arbitrary state S' minimal for $m(S') \geq R$. Where $A \in R$ is the first formal species chosen by the algorithm, there is at least one $x \in S'$ with $A \in m(x)$, and the algorithm at some point iterates through that x. Consider $Q_x = R \backslash m(x)$ as generated in the algorithm, and $S'_x = S' - \{\!| x |\!\}$. If we can show that S'_x is minimal for $m(S'_x) \geq Q_x$, then by assumption the recursive call to the algorithm generates S'_x, thus the algorithm generates $S' = S'_x + \{\!| x |\!\}$. If S'_x is not minimal for Q_x, then there is some $y \in S'_x$ such that $m(S'_x - \{\!| y |\!\}) \geq Q_x$. However, if so, then $y \in S'$ and $m(S' - \{\!| y |\!\}) = m(S'_x - \{\!| y |\!\}) + m(x) \geq Q_x + (R \wedge m(x)) = R$, thus S' was not minimal for R, creating a contradiction. Thus the algorithm generates every minimal S', completing the proof of correctness.

Finally, we prove that the algorithm takes time $\text{poly}(n^k)$ and space $\text{poly}(n, k)$. Since the algorithm adds one implementation species at each recursion depth and subtracts at least one species from R at each depth, the depth is at most $|R| = k$. Iterating through at most $|\mathcal{S}'| = n$ implementation species at each depth proves a bound of n^k on the number of minimal implementation states and the $\text{poly}(n^k)$ time bound. At any time the algorithm stores one implementation species plus $\text{poly}(n, k)$ information for the Q_x and S'_x's for each recursion depth, proving the space bound.

If $k \geq n$ then instead of removing one copy of one implementation species x at each recursive step, we choose one implementation species x and a number α, set $Q_{\alpha x} = R \backslash \alpha m(x)$, and mark that x cannot be chosen again in the recursive call. To keep only minimal states, we bound α such that $(\alpha - 1)m(x) \wedge R < \alpha m(x) \wedge R$ (i.e. the αth copy of x is not redundant); since we only choose x with $|m(x)| \geq 1$ and $|R| = k$ this implies $\alpha \leq k$. This algorithm has a depth of at most n, making a choice out of k possibilities at each step and keeping track of at most n numbers each bounded by k, which proves the bounds of time $2^{\text{poly}(n, \log k)}$ and space $\text{poly}(n, \log k)$ given when $k \geq n$. (Note that $k \geq n \geq 2 \Rightarrow n \log k \leq k \log n$, so the bounds given for $k \geq n$ are tighter than the bounds given for the general case.) □

Consider applying the algorithm described in Lemma 2.5.2 to the example implementation CRN in Figure 2.8. Here $R = \{\!| A + B |\!\}$, and assume A is chosen first. The algorithm then splits into different branches based on the choice of implementation species that contains A, where each branch may enumerate one or more minimal states; we discuss the branches separately. One branch of the algorithm will choose j_{AB} to contain A, and stop, since $m(j_{AB}) \geq R$, enumerating one minimal state. Other branches will choose each

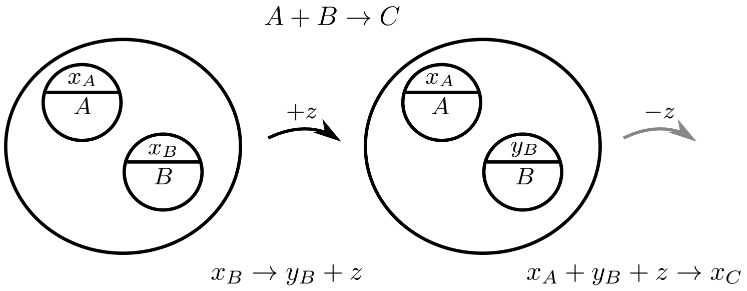


Figure 2.9: Minimal state $S'_0 = \{\!|x_A, x_B|\!\}$ can implement $A + B \to C$ by producing and then consuming a null species z. Because z must be produced from x_B, minimal state $S'_1 = \{\!|x_A, y_B|\!\}$ cannot implement $A + B \to C$.

of i_A, x_A, and i_{AC}. Each of those branches will recursively call the algorithm with $Q_x = \{\!|B|\!\}$; in particular, in the i_{AC} line, $\{\!|A + B|\!\} \setminus \{\!|A + C|\!\} = \{\!|B|\!\}$. Within each of those three branches, the recursive call will again split into branches, with one branch that considers each of i_B, x_B, j_B, and j_{AB} to contain B. The branches that consider j_{AB} will conclude that the resulting minimal states e.g. $\{\!|x_A + j_{AB}|\!\}$ are not minimal, but the other branches will conclude that the resulting states are minimal, enumerating the other 9 minimal states shown in Figure 2.8.

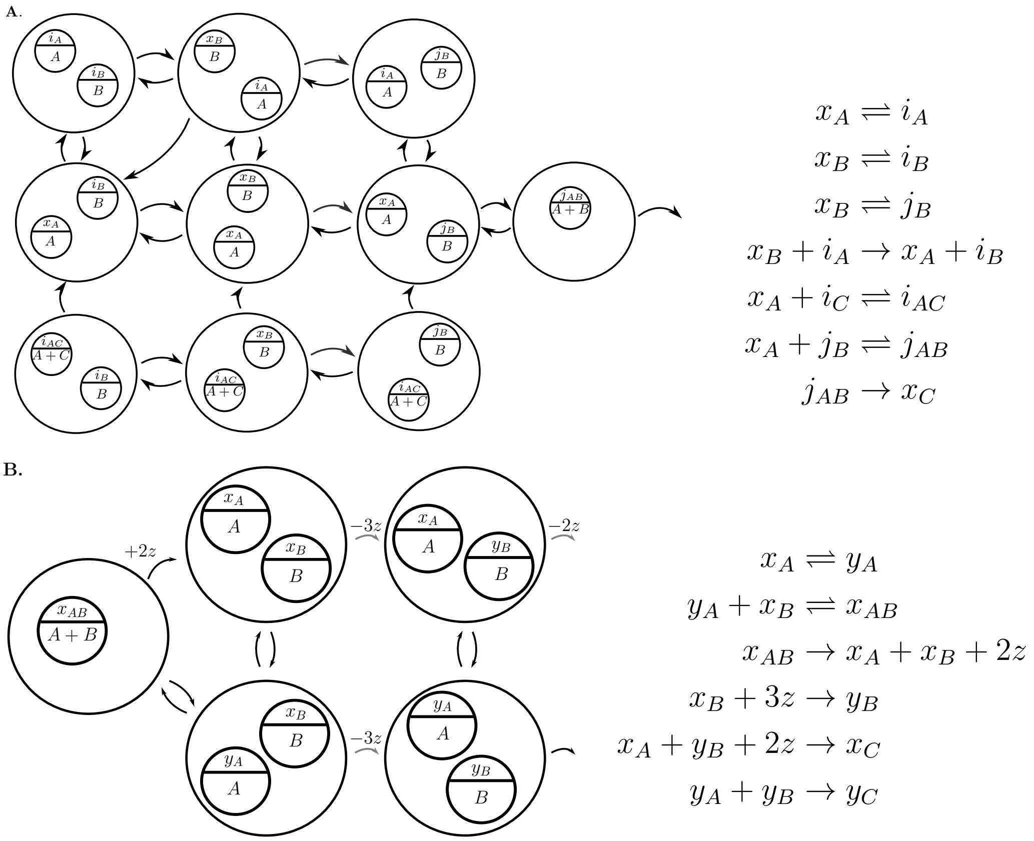
Now we have reduced an infinite number of possible initial implementation states to a finite number of minimal implementation states for each formal reaction. We have to check, for each $S'_0 \in \mathcal{M}(r)$, whether $S'_0 \overset{r}{\Rightarrow}$; equivalently, whether $S'_0 \overset{\tau}{\Rightarrow} T'$ where $T' \geq R'$ for some implementation reaction with $m(R' \to P') = r$. Checking this for *one* S'_0 is the "superset reachability" or "covering" problem, which was proven by Rackoff [58] and Lipton [49] to be EXPSPACE-complete. Surprisingly, we found that the requirement that *every* $S'_0 \overset{r}{\Rightarrow}$ makes the permissive condition checkable in PSPACE, by ruling out complex constructions such as Lipton's proof of hardness in [49].

Intuitively, we will show that if some $S'_0 \overset{r}{\Rightarrow}$ but requires the full complexity of exponential space to determine that, then that complexity will force some other S'_1 with $m(S'_1) \geq R$ to have $S'_1 \overset{r}{\not\Rightarrow}$. Figure 2.9 provides a simplified example of the principle we use. Here we have two minimal states for the formal reaction $r = A + B \to C$. One of them, $S'_0 = \{\!|x_A, x_B|\!\}$ can implement r via the reactions $x_B \to y_B + z$ and $x_A + y_B + z \to x_C$. In doing so, it passes through a non-minimal state $\{\!|x_A, y_B, z|\!\}$, and requires the extra species z to finish implementing r. However, that the extra z is required to implement r

means that the other minimal state $S_1' = \{\!|x_A, y_B|\!\}$ has no way to implement r; so the permissive condition is false, and we don't need to check whether $S_0' \overset{r}{\Rightarrow}$. This idea turns out to be generalizable, and allows us to mostly ignore "null species" with $m(x) = \emptyset$, which among other things prevents the complexity necessary for Lipton's proof in [49]. In particular, we will show that when checking for a pathway by which $S' \overset{r}{\Rightarrow}$ we need only consider the minimal states $\mathcal{M}(r)$ plus a small amount of additional information, and that this can be done in poly(n^k) time and poly(n, k) space, where $n = |\mathcal{S}'| + |\mathcal{R}'|$ and $k = |R|$.

For a simple case, consider an implementation CRN with no species x where $m(x) = \emptyset$, such as the one shown in Figure 2.10A, and consider its graph of minimal states for a formal reaction $r = R \rightarrow P$. If, for every minimal state, there is a path through the graph of minimal states to a reaction that implements r, then the permissive condition is true. In fact, the permissive condition is true if and only if such a path exists for every minimal state. If S_0' is minimal and $S_0' \overset{\tau}{\Rightarrow} S''$ which is not minimal, then there is some minimal S_1' with $S'' > S_1'$, which without null species must have $m(S_1') < m(S'') = m(S_0')$. Here either $S_1' \overset{r}{\Rightarrow}$ and therefore $S_0' \overset{r}{\Rightarrow}$ by reaching S'' and then following the same path by which $S_1' \overset{r}{\Rightarrow}$, or $S_1' \overset{r}{\Rightarrow}$ and the permissive condition is false regardless of whether $S_0' \overset{r}{\Rightarrow}$. Since $m(S_1') < m(S_0')$, the path by which $S_1' \overset{r}{\Rightarrow}$ cannot pass through S_0', so reducing the question of $S_0' \overset{r}{\Rightarrow}$ to $S_1' \overset{r}{\Rightarrow}$ is valid. Effectively, for the purpose of checking the permissive condition, we can pretend S'' is S_1', thus reducing our search for trajectories to a search for paths through the set of minimal states. Where k is the number of reactants in r and n the number of implementation species, we know from Lemma 2.5.2 that the number of minimal states is at most n^k when $k \leq n$ and at most $2^{n \log k}$ when $k \geq n$, both of which are exponential in the size of the CRN as defined at the beginning of this section. Because searching for paths through a graph can be done in space logarithmic in the size of the graph [60], we can check the permissive condition in polynomial space when there are no null species. To generalize this, we show that null species and loops do not make this bound worse.

Now consider an implementation CRN with null species, such as the one shown in Figure 2.10B, and its graph of minimal states for a formal reaction $r = R \rightarrow P$. We can try to apply the same logic as in the case without null species: if a minimal state $S_0' \overset{\tau}{\Rightarrow} S''$ non-minimal with a minimal state $S_1' < S''$, either



$$\begin{aligned} x_A &\rightleftharpoons i_A \\ x_B &\rightleftharpoons i_B \\ x_B &\rightleftharpoons j_B \\ x_B + i_A &\rightarrow x_A + i_B \\ x_A + i_C &\rightleftharpoons i_{AC} \\ x_A + j_B &\rightleftharpoons j_{AB} \\ j_{AB} &\rightarrow x_C \end{aligned}$$

$$\begin{aligned} x_A &\rightleftharpoons y_A \\ y_A + x_B &\rightleftharpoons x_{AB} \\ x_{AB} &\rightarrow x_A + x_B + 2z \\ x_B + 3z &\rightarrow y_B \\ x_A + y_B + 2z &\rightarrow x_C \\ y_A + y_B &\rightarrow y_C \end{aligned}$$

Figure 2.10: Example graphs of minimal states. We draw an arrow from a state S'_1 to a state S'_2 if there is a trivial reaction that can occur in S'_1 (plus some null species) and the resulting state is $\geq S'_2$. Arrows "out" (with no target) represent implementation reactions interpreted as the formal reaction in question and that can occur in the minimal state in question plus some null species. **A.** An example graph for the reaction $A + B \rightarrow C$ in an implementation CRN without null species, producing the same set of minimal states shown in Figure 2.8. Here the permissive condition is true for $A + B \rightarrow C$ if and only if every minimal state has a path to some arrow out, which we can see is true for this graph. Note that the reaction $i_{AC} \rightarrow x_A + i_C$ in state $\{\!|i_{AC} + x_B|\!\}$ would result in $\{\!|x_A + x_B + i_C|\!\}$, which is not minimal but is $> \{\!|x_A + x_B|\!\}$, so the arrow is from $\{\!|i_{AC} + x_B|\!\}$ to $\{\!|x_A + x_B|\!\}$. Since the reverse reaction $x_A + i_C \rightarrow i_{AC}$ is not possible in any minimal state and our algorithms do not need it, as far as we are concerned the reverse reaction is impossible. **B.** An example graph for the reaction $A + B \rightarrow C$ in an implementation CRN with one null species z. Here arrows for reactions that consume and/or produce null species are marked with the number of null species they consume and/or the number they produce. Arrows for reactions that consume null species are grayed out, since they cannot occur in the minimal state in question, but may be relevant if the required null species are produced. Checking the permissive condition for $A + B \rightarrow C$ in this CRN may require more complex techniques.

$S_1' \overset{r}{\Rightarrow}$ and we can pretend S'' is S_1', or $S_1' \overset{r}{\not\Rightarrow}$ and the permissive condition is false anyway. Without null species this was valid because $S'' > S_1'$ implies $m(S'') > m(S_1')$, and thus S_1' cannot reach S_0' via trivial reactions. With null species, on the other hand, it may be that $S'' - S_1'$ contains only null species and it may be that $S_1' \overset{\mathcal{T}}{\Rightarrow} S_0'$; in particular it may be that both $S_0' \overset{r}{\Rightarrow}$ and $S_1' \overset{r}{\Rightarrow}$, but the only path by which $S_1' \overset{r}{\Rightarrow}$ goes through S_0' and the only path by which $S_0' \overset{r}{\Rightarrow}$ goes through S'', creating a loop that will not be found when searching through the graph of minimal states. In fact, this is exactly the case in the CRN shown in Figure 2.10B, for example with $S_0' = \{\!|x_{AB}|\!\}$ and $S_1' = \{\!|x_A + x_B|\!\}$. All minimal states in that CRN can in fact implement $A + B \rightarrow C$, but doing so for e.g. $\{\!|x_A + x_B|\!\}$ requires a "loop" through $\{\!|y_A + x_B|\!\}$ and $\{\!|x_{AB}|\!\}$ to $\{\!|x_A + x_B + 2z|\!\}$, eventually producing enough z for the reactions $x_B + 3z \rightarrow y_B$ and $x_A + y_B + 2z \rightarrow x_C$, which is interpreted as $A + B \rightarrow C$. Since any state with z is a non-minimal state, that path cannot be found by searching through only the graph of minimal states.

In the CRN in Figure 2.10B, the path by which $\{\!|x_A + x_B|\!\} \overset{r}{\Rightarrow}$ involves a "loop" by which $\{\!|x_A + x_B|\!\} \overset{\mathcal{T}}{\Rightarrow} \{\!|x_A + x_B|\!\} + \{\!|2z|\!\}$, i.e. a minimal state can reach a state equal to itself plus some null species. In such a case, that loop can be repeated any number of times to produce any number of z, and when searching for a path, there is no need to keep track of the exact number of z produced: either there are no z, or there are "enough" z produced by a previous loop. Recall the argument we tried to use that failed: if $S_0' \overset{\mathcal{T}}{\Rightarrow} S_1' + Y$, with S_0' and S_1' minimal, then either $S_1' \overset{r}{\Rightarrow}$ and so does S_0', or $S_1' \overset{r}{\not\Rightarrow}$ and the permissive condition is false. Recall that this argument only failed because it may be that $S_1' \overset{r}{\Rightarrow}$ but only by passing through S_0', in a situation similar to the loop in Figure 2.10B. This suggests, which we will show is true, that this example is general: if $S_0' \xrightarrow{\mathcal{T}} S_1' + Y$ for $S_0', S_1' \in \mathcal{M}(r)$, then for each $y \in Y$, either y can be completely ignored when checking the permissive condition, or else the following all hold: $m(y) = \emptyset$, there is some $S_j' \in \mathcal{M}(r)$ such that $S_j' \overset{\mathcal{T}}{\Rightarrow} S_j' + y + \ldots$, and y is only "relevant" after it has been produced in some such loop.

The above discussion allows us to define a graph, which can be both enumerated and searched through in polynomial space, such that paths through the graph correspond to paths by which a given minimal state implements a formal reaction r. The states of this *loopsearch graph* are triples of the form (S', S_0', ζ)

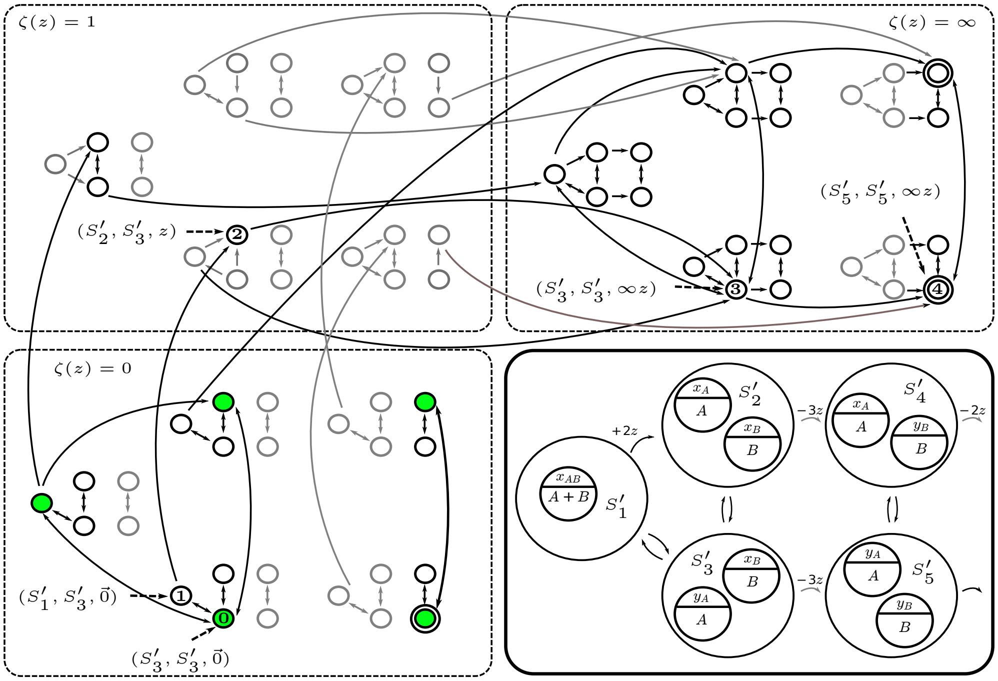


Figure 2.11: Example of the loopsearch graph for the formal reaction $A+B \rightarrow C$ with implementation CRN from Figure 2.10B. The graph of minimal states from Figure 2.10B is reproduced at bottom right, with each minimal state given a name S'_i. In the loopsearch graph, initial vertices $(S'_i, S'_i, \vec{0})$ are filled in green, and terminal vertices represented by doubled circles. Vertices and edges not reachable from any initial vertex are grayed out, as they are not relevant to the theory or algorithms that follow. The permissive condition is true for $A+B \rightarrow C$ if and only if for each initial vertex, there is a path in the loopsearch graph to some terminal vertex. One such path is given by the numbered vertices 0 through 4 from initial vertex $(\{\!|y_A + x_B|\!\}, \{\!|y_A + x_B|\!\}, \vec{0})$ to terminal vertex $(\{\!|y_A + y_B|\!\}, \{\!|y_A + y_B|\!\}, \infty z)$; observe that each of the other four initial vertices can also reach a terminal vertex, so the permissive condition is true for this interpretation.

where ζ maps each null species in $\mathcal{Z}$ to 0, 1, or ∞: in each state we are at or covering one minimal implementation state $S' \in \mathcal{M}(r)$, in the middle of a loop beginning at some other state $S'_0 \in \mathcal{M}(r)$, and each null species in $\mathcal{Z}'$ is either absent, produced previously in the current incomplete loop, or present in infinite copies from a previous loop. An example of such a graph is given in Figure 2.11.

We use the following notation in defining and discussing the loopsearch graph, in the context of a given formal and implementation CRN with interpretation and a specific formal reaction r. Let $S'_0, S'_1 \in \mathcal{M}(r)$, $\zeta \in 3^{\mathcal{Z}}$ and $Z \in 2^{\mathcal{Z}}$. We write $\zeta^{-1}(x) = \{z \in \mathcal{Z} \mid \zeta(z) = x\}$ for $x \in \{0, 1, \infty\}$; in particular, $\zeta^{-1}(\infty)$ is the set of null species that have been produced in previous loops, and are thus "available" for use later. We write $S'_0 \xrightarrow{\zeta} S'_1 + Z$ if there is a trivial reaction that, for some $n \in \mathbb{N}$, can occur in $S'_0 + n\zeta^{-1}(\infty)$, where the resulting state is some S'' containing S'_1 and at least one copy of each null species in Z. Z may be empty, in which case $S'_0 \xrightarrow{\zeta} S'_1$ is the same as $S'_0 \xrightarrow{\zeta} S'_1 + \emptyset$. Following the terminology of [58], to "cover" a state S in a CRN is to be in a state containing S plus possibly some other species.

Definition 2.5.2. Given $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ a formal and implementation CRN, $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ an interpretation, and r a formal reaction, we define the *loopsearch graph* for r. The loopsearch graph is a directed graph with vertex set $\mathcal{M}(r) \times \mathcal{M}(r) \times 3^{\mathcal{Z}}$, where $\mathcal{Z} = \{z \in \mathcal{S}' \mid m(z) = \emptyset\}$, with some vertices designated as "terminal". Here a vertex (S', S'_0, ζ) is interpreted as, "we are covering state S', in the middle of a loop starting and ending at S'_0, with null species present or absent as determined by ζ", except that $S' = S'_0$ is interpreted as "not in the middle of a loop". $\zeta \in 3^{\mathcal{Z}}$ maps each null species z to $\{0, 1, \infty\}$, a coarse-grained representation of the number of copies of z: we only need to remember whether z is not present ($\zeta(z) = 0$), produced during the current loop (1), or produced as many times as necessary in a previous loop (∞). This interpretation suggests the definition of the edges of the loopsearch graph, which is as follows:

- **Reactions outside a loop:** Whenever $S' \xrightarrow{\zeta} S'_1$ and $\zeta^{-1}(1) = \emptyset$, there is an edge from (S', S', ζ) to (S'_1, S'_1, ζ).
- **Reactions inside a loop:** Whenever $S' \xrightarrow{\zeta} S'_1 + Z$, there is an edge from (S', S'_0, ζ) to (S'_1, S'_0, ζ') for each $S'_0 \neq S'_1$, where ζ' is defined as follows:
  - If $\zeta(z) = 1$ or ∞ then $\zeta'(z) = \zeta(z)$.
  - If $\zeta(z) = 0$ and $z \notin Z$ then $\zeta'(z) = 0$.
  - If $\zeta(z) = 0$ and $z \in Z$ then $\zeta'(z) = 1$.
- **Finishing a loop:** Whenever $S'_1 \xrightarrow{\zeta} S'_0 + Z$, there is an edge from (S', S'_0, ζ) to (S'_0, S'_0, ζ'), where $\zeta'(z) = 0$ if $\zeta(z) = 0$ and $z \notin Z$, otherwise $\zeta'(z) = \infty$.

A vertex (S', S', ζ) is designated as "terminal" if $S' + \infty\zeta^{-1}(\infty) \xrightarrow{r}$, that is, if there is some implementation reaction r' with $m(r') = r$ that can occur in S' plus sufficiently many copies of null species z with $\zeta(z) = \infty$.

Some comments on the definition may help give an intuitive understanding of the loopsearch graph. First, note that ζ is monotonic in this graph: for any given $z \in \mathcal{Z}$, $\zeta(z)$ can change from 0 to 1, from 1 to ∞, or from 0 to ∞, but never decrease. A null species z can be produced inside a loop, but the paths we are searching for cannot use z inside the loop where it was first produced; and once that loop ends, z is present in "infinite" copies and will always be so. Second, the loopsearch graph has a repeating substructure that mirrors the structure of the graph of minimal states; compare Figure 2.11 bottom right to the remainder of Figure 2.5.2. Vertices of the form (S', S', ζ) for fixed ζ with $\zeta^{-1}(1) = \emptyset$, with edges from "Reactions outside a loop" in Definition 2.5.2, have exactly the structure of the graph of minimal states, except edges "grayed" in the graph of minimal states may or not be present in the loopsearch graph. Specifically, such grayed edges represent reactions that have null species as reactants (see Figure 2.10B, Figure 2.11 bottom right), and are present in the parts of the loopsearch graph where the null species z that are reactants of the corresponding reaction have $\zeta(z) = \infty$. Vertices of the form (S', S'_0, ζ) for fixed S'_0 and ζ, with edges from "Reactions inside a loop" and "Finishing a loop" in Definition 2.5.2, have a structure very close to the graph of minimal states, differing occasionally when the edge changes ζ. Finally, many of the vertices in the loopsearch graph are unreachable from any "initial vertex" (i.e., vertex of the form $(S', S', \vec{0})$); usually such unreachable vertices, according to the meaning we give them, would contain some sort of contradiction. For example, every vertex of the form (S', S', ζ) with $\zeta^{-1}(1) \neq \emptyset$ will be unreachable in any loopsearch graph; in such a vertex, the form (S', S', ζ) means we are at state S' and not in the middle of a loop, but $\zeta(z) = 1$ means z has been produced in the current, nonexistent loop. Other vertices are unreachable due to less obvious contradictions; in the example in Figure 2.11, where $i \in \{1, 2, 3\}$ and $j \in \{4, 5\}$, vertices of the form (S'_i, S'_j, ζ) are unreachable, because we would be at S'_i in a loop starting at S'_j, but such states S'_i are unreachable from states S'_j; similarly, vertices of the form (S'_j, S'_i, ζ) are unreachable for $\zeta(z) \neq \infty$. Unreachable vertices and edges are shown in grey in Figure 2.11.

Because the edges in the loopsearch graph come from trivial reactions possible at the corresponding states, any path through the loopsearch graph implies the existence of a class of trajectories in the implementation CRN. A segment from (S'_0, S'_0, ζ) to (S'_1, S'_1, ζ) traveling only through vertices of the form (S', S', ζ) with no change in ζ implies that the corresponding sequence of trivial reactions can occur starting from S'_0 plus some null species, including "sufficiently many" copies of $\zeta^{-1}(\infty)$, and ending in a state that covers S'_1. A segment from (S'_0, S'_0, ζ) to (S'_0, S'_0, ζ') traveling only through vertices of the form (S', S'_0, ζ'') implies the existence of a "loop" in the implementation CRN of the form $S'_0 + Z_0 \overset{\tau}{\Rightarrow} S'_0 + Z_1$, where Z_0 includes "sufficiently many" copies of $\zeta^{-1}(\infty)$, and Z_1 includes at least all null species in $(\zeta')^{-1}(\infty)$ that are not in $\zeta^{-1}(\infty)$. Given any number of times for this loop to happen, it can happen that many times starting with sufficiently many copies of $\zeta^{-1}(\infty)$, producing as many copies of $(\zeta')^{-1}(\infty)$ as desired. This logic lets us compose paths made of these segments into trajectories possible in the implementation CRN, by taking each loop as many times as necessary to produce all species necessary for all future segments. In particular, a path from $(S'_0, S'_0, \vec{0})$, where $\vec{0} \in 3^{\mathcal{Z}}$ is the function that maps all null species to 0, to a terminal state of the form (S', S', ζ) implies the existence of a trajectory in the implementation CRN by which $S'_0 \overset{r}{\Rightarrow}$. Thus, if such paths can be found for every minimal S'_0, we know that the permissive condition is satisfied. What we will show is that, if the permissive condition is satisfied, then it is satisfied by trajectories corresponding to paths through the loopsearch graph.

Lemma 2.5.3. *Let $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ be a formal and implementation CRN, with interpretation m. Let $r = (R, P) \in \mathcal{R}$ be a formal reaction and S'_0 an implementation state minimal for $m(S'_0) \geq R$. If the permissive condition is satisfied, then there exists a path through the loopsearch graph described in Definition 2.5.2 from $(S'_0, S'_0, \vec{0})$ to some terminal state, where $\vec{0}(z) = 0$ for all null species z. Conversely, if such paths exist for* every *formal reaction and minimal implementation state, then the permissive condition is satisfied.*

Proof. Given $r = R \to P \in \mathcal{R}$ and $S'_0 \in \mathcal{M}(r)$, assuming the permissive condition is true, we find a path through the loopsearch graph for r from $(S'_0, S'_0, \vec{0})$ to a terminal state. In particular, we show that if the permissive condition is true, then from any (S'_1, S'_1, ζ) where $\zeta^{-1}(1) = \emptyset$, there is a path either to a terminal state (S'_2, S'_2, ζ) for the same ζ, or to another such state

(S_2', S_2', ζ') where $(\zeta')^{-1}(1) = \emptyset$ and $\zeta^{-1}(\infty) \subsetneq (\zeta')^{-1}(\infty)$, from which this process can be repeated. Since $\zeta^{-1}(\infty) \subset \mathcal{Z}$ which is finite, this process must find a terminal state in finitely many steps, namely at most $|\mathcal{Z}|$.

Given arbitrary (S_1', S_1', ζ) with $\zeta^{-1}(1) = \emptyset$, let $Z = \zeta^{-1}(\infty)$ and note that for each $S'' \in \mathcal{M}(r)$ and $\mu \geq 0$, $m(S'' + \mu Z) \geq R$. By the permissive condition, there is a trajectory in the implementation CRN by which $S'' + \mu Z \overset{r}{\Rightarrow}$; for each S'', choose a shortest such path. Construct a new trajectory by starting at $S_1' + \mu Z$, where μ is high enough for this trajectory to be valid, and at each step where we are $\geq$ some $S'' + \mu' Z$, take the first reaction on the chosen shortest path for S''. Continue until the trajectory either takes an implementation reaction r' with $m(r') = r$, or covers the same minimal state S_2' twice.

If the trajectory takes an implementation reaction r' with $m(r') = r$, then by the construction of the loopsearch graph, for each reaction in the trajectory from a state $\geq S_1''$ to a state $\geq S_2''$, there is an edge from (S_1'', S_1'', ζ) to (S_2'', S_2'', ζ). Where S_2' is the minimal state such that r' was taken in a state $\geq S_2'$, since r' was possible that means (S_2', S_2', ζ) is a terminal state. Thus these edges give a path from (S_1', S_1', ζ) to (S_2', S_2', ζ) which is terminal, which is the desired path.

If on the other hand the trajectory covers the same minimal state S_2' twice, then there must be at least one null species $z \notin Z$ produced by a reaction between the first and second times S_2' is covered; otherwise such a path would be a futile loop, implying that for at least one S'' covered in that time there is a μ which gives a shorter path by which $S'' + \mu Z \overset{r}{\Rightarrow}$. Then again by the construction of the loopsearch graph, for each reaction in the trajectory from a state $\geq S_1''$ to a state $\geq S_2''$ *before* the first state $\geq S_2'$, there is an edge from (S_1'', S_1'', ζ) to (S_2'', S_2'', ζ). For each reaction *after* the first state $\geq S_2'$, there is an edge from (S_1'', S_2', ζ_1'') to (S_2'', S_2', ζ_2''), with the first $\zeta_1'' = \zeta$, each new ζ_2'' equalling ζ_1'' except that any null species z produced in the corresponding reaction with $\zeta_1''(z) = 0$ has $\zeta_2''(z) = 1$ ("Reaction inside a loop" in Definition 2.5.2), and the last $\zeta_2'' = \zeta'$ has $\zeta'(z) = \infty$ if any $\zeta''(z) = \infty$ or 1 or if z was produced by the last reaction ($\zeta'(z) = 0$ otherwise), ending in the state (S_2', S_2', ζ') ("Finishing a loop"). Since at least one such $z \notin \zeta^{-1}(\infty)$ must have been produced, this is a path from (S_1', S_1', ζ) to (S_2', S_2', ζ') with $\zeta^{-1}(\infty) \subsetneq (\zeta')^{-1}(\infty)$, which is the desired path.

If such a path through the loopsearch graph from $(S_0', S_0', \vec{0})$ exists for a given formal reaction r and minimal implementation state S'_0, then we show $S'_0 \overset{r}{\Rightarrow}$. We gave this argument informally above. Where states are of the form (S'_i, S'_j, ζ), observe from Definition 2.5.2 that the only edges that changes ζ leave S'_j unchanged (i.e. are in a loop), and the only edges that change S'_j are edges from (S'_1, S'_1, ζ) to (S'_2, S'_2, ζ) leaving ζ unchanged with $\zeta^{-1}(1) = \emptyset$ (i.e. are outside a loop). From that, given a path from $(S'_0, S'_0, \vec{0})$ to some terminal state (S'_f, S'_f, ζ_f), we can divide the path into segments as follows: in states of the form (S'_i, S'_j, ζ), segments will alternate between segments where all states have $S'_i = S'_j$ and ζ is unchanged ("paths"), followed by segments where S'_j is unchanged ("loops"). Where (S'_i, S'_i, ζ_i) is the state at the end of the ith loop and $Z_i = \zeta_i^{-1}(\infty)$, we show by induction on i that for any $\mu \geq 0$, $S'_0 \overset{\mathcal{T}}{\Rightarrow} S'_i + \mu Z_i$. The base case $i = 0$ has $\vec{0}^{-1}(\infty) = \emptyset$, so $S'_0 \overset{\mathcal{T}}{\Rightarrow} S'_0 + \mu\emptyset$ is trivially true. Assuming that $S'_0 \overset{\mathcal{T}}{\Rightarrow} S'_i + \mu Z_i$ for any μ, consider the sequence of trivial reactions corresponding to the edges on the path from (S'_i, S'_i, ζ_i) to $(S'_{i+1}, S'_{i+1}, \zeta_i)$ and the loop from there to $(S'_{i+1}, S'_{i+1}, \zeta_{i+1})$. Those trivial reactions are, by Definition 2.5.2, possible using only null species in Z_i; let μ' be the largest number of a single null species in Z_i used after adding up all reactants of all reactions along the path and loop (ignoring any products of the reactions). Given arbitrary $\mu \geq 0$, let $\mu'' = (\mu+1)(\mu'+1)$, and by assumption, $S'_0 \overset{\mathcal{T}}{\Rightarrow} S'_i + \mu'' Z_i$. Then by following the trivial reactions along the path from S'_i to S'_{i+1}, we use up at most $\mu' Z_i$, so $S'_i + \mu'' Z_i \overset{\mathcal{T}}{\Rightarrow} S'_{i+1} + \mu(\mu'+1)Z_i$. Then by following the trivial reactions along the loop μ times, each loop uses at most $\mu' Z_i$ and produces at least $Z_{i+1} \backslash Z_i$, so $S'_{i+1} + \mu(\mu'+1)Z_i \overset{\mathcal{T}}{\Rightarrow} S'_{i+1} + \mu Z_i + \mu(Z_{i+1} \backslash Z_i)$, completing the induction. That $S'_0 \overset{\mathcal{T}}{\Rightarrow} S'_f + \mu Z_f$ for all $\mu \geq 0$ is a special case of this proof, and since by Definition 2.5.2 that (S'_f, S'_f, ζ_f) is a terminal state means $S'_f + \mu Z_f \xrightarrow{r}$ for some $\mu \geq 0$, this proves that $S'_0 \overset{r}{\Rightarrow}$.

If such paths exist for every formal reaction r and minimal state S'_0 for r, then every minimal state $S'_0 \overset{r}{\Rightarrow}$. Thus as discussed in Lemma 2.5.1 every state with $m(S') \xrightarrow{r}$ has $S' \overset{r}{\Rightarrow}$, satisfying the permissive condition. □

With this preparation, we can now describe algorithms to check the permissive condition. Having shown that the permissive condition is true if and only if certain paths through the loopsearch graph exist, our algorithms will be based on searching for those paths. In general, if a formal reaction $r = R \to P$ has $k = |R|$ reactants and the implementation CRN has $n = |\mathcal{S}'|$ species, there may be order n^k minimal implementation states for r and the trajectories by which any one implements r may have to pass through most or all of them. As that suggests, we will later prove that checking the permissive condition (and thus checking an interpretation in general) is PSPACE-complete. So the first algorithm we present is the loopsearch algorithm, which runs in $\text{poly}(n, k)$ space and $\text{poly}(n^{kn})$ time, which is Algorithm 2.1.

The size (number of vertices) of the loopsearch graph is $|\mathcal{M}(r)|^2 3^{|\mathcal{Z}|}$, at worst exponential in the size of the CRNs, and we have reduced the permissive condition to a question of whether paths between certain pairs of vertices exist in that graph. Savitch's theorem states that we can decide whether such paths exist through a graph of size N in $\log^2 N$ space [60], which given the results so far completes the proof that the permissive condition can be decided in polynomial space; the loopsearch algorithm is just a concrete application of Savitch's result to the loopsearch graph. Specifically, the loopsearch algorithm breaks a path from $(S'_0, S'_0, \vec{0})$ into alternating segments of two types: one type from (S'_i, S'_i, ζ_i) to $(S'_{i+1}, S'_{i+1}, \zeta_i)$ through only states of the form (S', S', ζ_i), and the other type from $(S'_i, S'_i, \zeta_{i-1})$ to (S'_i, S'_i, ζ_i) through only states of the form (S', S'_i, ζ), the same decomposition discussed in the proof of Lemma 2.5.3. Both types of segments can have length no longer than $|\mathcal{M}(r)|$; for the first type, this is obvious, while for the second, we rely on the proof of Lemma 2.5.3 to say that a decomposition exists where no segment covers the same minimal state twice. To search for a path of length $2^0 = 1$, we check each possible edge (trivial reaction) to see if the start and target state are connected; to search for a path of length 2^{i+1}, we check for each possible intermediate state, whether a path of length 2^i exists from the start to the intermediate, and whether a path of length 2^i exists from the intermediate to the target. For a segment of the first type, the possible intermediate states are just (S', S', ζ_i) for $S' \in \mathcal{M}(r)$, while for a segment of the second type, the possible intermediate states are (S', S'_i, ζ) where $S' \in \mathcal{M}(r)$ and for all $z \in \mathcal{Z}$, $\zeta_{i-1}(z) \leq \zeta(z) \leq \zeta_i(z)$. This condition, which reduces the number of ζ to check, relies on a monotonicity of $\zeta(z)$ that follows from the types of edges defined in Definition 2.5.2.

Theorem 2.5.1. *Whether an interpretation is a bisimulation can be checked in polynomial space.*

Proof. We show that the loopsearch algorithm is correct and runs in polynomial space. We proved in Lemma 2.5.3 that the permissive condition is true if and

```
def loopsearch(CRN formal, CRN impl, interpretation m):
  Z = { species x in impl where m(x) is empty }
  for each reaction r in formal:
    Min = minimal_states(impl, m, r)
    k = log(|Min|, base 2)
    for each state S’0 in Min:
      found = False
      for each sets <Y0,Y1,...,Yl> in partitions(Z,|Z|+1),
                               states <S’1,...,S’l> in Min:
        if (for all i in 1,...,l, ...
            reach_with_inf(S’(i-1), S’i, Y1+...+Y(i-1), k) ...
      and reach_with_inf(S’i, S’i + Yi, Y1+...+Y(i-1), k)) ...
      and reach_with_inf(S’l, r, Y1+...+Yl, k):
          found = True
      if not found:
        return False

  # if found for all r and all S’0 ...
  return True

def reach_with_inf(state start, (state or reaction) target, ...
                   (set of species) infinites, integer k):
  # check if start can reach target in at most 2^k reactions
  #  given infinitely many copies of species in infinites
  if k is 0:
    for each trivial reaction r’:
      if r’ takes start to target: return True
    return False
  if k is not 0:
    middles = Min x 2^({z in target | m(z) = 0})
    for each state middle in middles:
      if reach_with_inf(start, middle, infinites, k-1) ...
           and reach_with_inf(middle, target, infinites, k-1):
        return True
    return False
```

Algorithm 2.1: The loopsearch algorithm to check the permissive condition in polynomial space.

only if a loop-segmented path exists for every formal reaction r and every minimal implementation state S'_0 for r, so we need only to show that the loopsearch algorithm finds a loop-segmented path if and only if one exists. Each loop-segmented path implicitly specifies a sequence of l minimal states S'_i, and a sequence of sets of null species Y_i. By removing from Y_i all null species in Y_j for $j < i$ and defining $Y_0 = Z - \bigcup_{i=1}^{l} Y_i$, we get the partition that the loopsearch algorithm searches for while preserving the loop-segmented path. Since the loops and paths in the desired loop-segmented path never repeat a minimal state, they must each have length at most N the number of minimal states, and a path of length 2^j from S'_a to S'_b exists if and only if for some S'_c a path of length 2^{j-1} from S'_a to S'_c and a path of length 2^{j-1} from S'_c to S'_b both exist. The desired loop-segmented path has each loop and path as a path between minimal states with certain null species ignored and the algorithm matches this restriction, so the loopsearch algorithm is correct.

At any point in the loopsearch algorithm, it is storing the following information: a formal reaction r, a minimal state S'_0, a partition of $l+1$ sets of null species Y_i (thus implying that $l \leq z \leq n$), a sequence of l minimal states S'_i, and at most $\lceil \log N \rceil$ triples of minimal states S'_a, S'_b, S'_c in the recursive search algorithm. The at most n^k minimal states can be enumerated in polynomial space (i.e. without storing any states other than the current and next one) as shown in Lemma 2.5.2, and similarly partitions of $z \leq n$ elements can be enumerated in poly(z) space. Also according to Lemma 2.5.2, the number of minimal states is $N \leq n^k$, so the depth of the search $\lceil \log N \rceil$ is poly(n, k). (When $k \geq n$, $N \leq 2^{n \log k}$ so the depth is at most poly($n, \log k$).) To complete the proof, note that as discussed earlier, checking the atomic and delimiting conditions are both trivial given an interpretation, thus whether an interpretation is a bisimulation can be checked in polynomial space. □

We have repeatedly said that the difficulty of checking the permissive condition scales with the number of minimal states for any given formal reaction $r = R \to P$, which typically scales like (and never scales worse than) n^k where $n = |\mathcal{S}'|$ and $k = |R|$. We stated earlier, and will show later, that when k is unbounded, checking an interpretation is PSPACE-complete. However, many CRNs in practice have large numbers of species but small numbers of reactants per (formal) reaction; in particular, almost any interesting behavior—if not any interesting behavior—that can be done with a CRN can be done with

```
def graphsearch(CRN formal, CRN impl, interpretation m):
  for each reaction r in formal:
    Min = minimal_states(impl, m, r)
    # states or r reachable from S'
    table reach(S') = <empty> for each state S' in Min
    # null species producible in a loop at S'
    table prod(S') = <empty> for each state S' in Min
    repeat until reach and prod are both unchanged:
      for each S'1 where reach(S'1) is not r:
        if S'1 + inf prod(S'1) can do r' with m(r') =  r:
          set reach(S'1) to r, continue to next S'1
        for each trivial reaction r' by which
                   S'1 + inf prod(S'1) -> S'2:
          if reach(S'2) is r:
            set reach(S'1) to r, continue to next S'1
          reach(S'1) += reach(S'2)
          if S'1 in reach(S'2):
            prod(S'1) += prod(S'2)
            prod(S'1) += {y in products(r') where m(y) = empty}
    if not all reach(S') is r: # after table no longer changes
      return False
  return True # if all reactions pass without returning False
```

Algorithm 2.2: The graphsearch algorithm to check the permissive condition in time and space polynomial in the number of minimal states.

a CRN with a bound of $k \leq 2$. For those CRNs, we present a graphsearch algorithm which takes poly(n^k) space and time, making it much faster than the loopsearch algorithm when k is small but taking much more space when k is large, as Algorithm 2.2.

For each formal reaction r, the graphsearch algorithm enumerates and creates a table of all implementation states S'_i minimal for r. The algorithm uses this table to store "known information" about which states are reachable from S'_i and iteratively updates this information, continuing until either every $S'_i \overset{r}{\Rightarrow}$ is known or until no further information can be known, in which case some $S'_i \overset{r}{\not\Rightarrow}$. For each S'_i, the algorithm stores whether or not it is known (yet) that $S'_i \overset{r}{\Rightarrow}$. If it is not yet known that $S'_i \overset{r}{\Rightarrow}$, then the algorithm stores, for each minimal state S'_i, whether it is known that $S'_i \overset{\mathcal{T}}{\Rightarrow} S'_j$, and for each $y \in \mathcal{S}'$ with $m(y) = \emptyset$, whether it is known that $S'_i \overset{\mathcal{T}}{\Rightarrow} S'_i + y$. Initially, the only thing known is that $S'_i \overset{\mathcal{T}}{\Rightarrow} S'_i$ for each S'_i.

Given this table, the algorithm goes through repeated "cycles" of updating the known reachabilities until one cycle passes with no changes. In each cycle, the algorithm iterates through each minimal S'_i. For each S'_i, if $S'_i \overset{r}{\Rightarrow}$ is known, the algorithm skips S'_i. If not, where Y_i is the set of all y such that $S'_i \overset{\tau}{\Rightarrow} S'_i + y$ is known, the algorithm checks for each implementation reaction r' with $m(r') = r$ whether it can occur in S'_i given arbitrarily many copies of Y_i. If $S'_i + \infty Y_i \xrightarrow{r'}$ for one of those r', then the algorithm records that $S'_i \overset{r}{\Rightarrow}$ and will skip S'_i in the future. Otherwise, the algorithm iterates through all trivial reactions in the implementation CRN and check whether they can occur in $S'_i + \infty Y_i$. For each reaction that can occur, the algorithm finds the (possibly multiple) minimal state(s) S'_j that are covered by the state after that reaction, and updates the table based on what is known about S'_j:

(i) If $S'_i + \infty Y_i \xrightarrow{\tau} S'_j$ and $S'_j \overset{r}{\Rightarrow}$, then $S'_i \overset{r}{\Rightarrow}$. Otherwise,

(ii) If $S'_i + \infty Y_i \xrightarrow{\tau} S'_j$ and $S'_j \overset{\tau}{\Rightarrow} S'_k$, then $S'_i \overset{\tau}{\Rightarrow} S'_k$.

(iii) If $S'_i + \infty Y_i \xrightarrow{\tau} S'_j + y$ and $S'_j \overset{\tau}{\Rightarrow} S'_i$, then $S'_i \overset{\tau}{\Rightarrow} S'_i + y$.

(iv) If $S'_i + \infty Y_i \xrightarrow{\tau} S'_j$ and $S'_j \overset{\tau}{\Rightarrow} S'_j + y$ and $S'_j \overset{\tau}{\Rightarrow} S'_i$, then $S'_i \overset{\tau}{\Rightarrow} S'_i + y$.

The algorithm terminates when a full cycle passes with no change to the table. At that time, if $S'_i \overset{r}{\Rightarrow}$ is known for every minimal S'_i, the algorithm states that the permissive condition is true; otherwise, the algorithm states that the permissive condition is false.

Theorem 2.5.2. *When the number of reactants in a formal reaction k is constant, whether an interpretation is a bisimulation can be checked in polynomial time.*

Proof. We prove that i) if the graphsearch algorithm returns true, then the permissive condition is true; ii) if the permissive condition is true, then the graphsearch algorithm will return true; and iii) the graphsearch algorithm always terminates in poly(n^k) time.

To prove the first, the graphsearch algorithm is based on deductions from "known" information. The initially known information is that each $S'_i \overset{\tau}{\Rightarrow} S'_i$, which is trivially true (since $\overset{\tau}{\Rightarrow}$ includes the sequence of 0 reactions). There are five deduction rules: the rule that if $S'_i \overset{\tau}{\Rightarrow} S'_i + Y_i$ and $S'_i + \infty Y_i \xrightarrow{r'}$ with

$m(r') = r$ then $S'_i \overset{r}{\Rightarrow}$, and the four listed rules for when $S'_i + \infty Y_i \xrightarrow{\tau} S'_j$. Each of the rules is a valid deduction, so any information deduced by the algorithm will be true. In particular, the algorithm says the permissive condition is true only when every minimal $S'_i \overset{r}{\Rightarrow}$, which by Lemma 2.5.1 implies that the permissive condition is in fact true.

To prove the second, we show that if the permissive condition is true but at a given cycle evaluating the formal reaction r the graphsearch algorithm does not yet know that every minimal $S'_i \overset{r}{\Rightarrow}$, then there is at least one additional fact that it can learn this cycle. The proof is similar, but not identical, to the proof of Lemma 2.5.3. At any given cycle, for each minimal state S'_i let Y_i be the set of all null species such that $S'_i \overset{\tau}{\Rightarrow} S'_i + Y_i$ is known. If the permissive condition is true, then the permissive condition is true in a modified CRN which for each S'_i adds the (trivial under the given interpretation) reaction $S'_i \to S'_i + Y_i$. For each minimal S'_i, there is a path in the modified CRN by which $S'_i \overset{r}{\Rightarrow}$ which is "shortest" in the sense of having the fewest reactions that are *not* any of the added reactions $S'_i \to S'_i + Y_i$; consider for each of those shortest paths the first reaction that is not one of the added reactions. Note that each of those reactions is a reaction, either trivial or implementing r, that the graphsearch algorithm will detect as possible and consider.

For each S'_i, construct a trajectory which starts at S'_i, takes the reaction $S'_i \to S'_i + Y_i$ as many times as necessary to take the first non-added reaction on the shortest path by which $S'_i \overset{r}{\Rightarrow}$, and takes that reaction to reach some S'_j; takes $S'_j \to S'_j + Y_j$ as many times as possible to take the first non-added reaction on the shortest path by which $S'_j \overset{r}{\Rightarrow}$, and takes that reaction; then continues this process. Each such path must eventually, in a number of reactions less than the number of minimal states $N \leq n^k$, either implement r or repeat a minimal state. If any path eventually implements r, and for any minimal state S'_k on that path it is not known that $S'_k \overset{r}{\Rightarrow}$, then the last such S'_k has the reaction $S'_k \xrightarrow{\tau} S'_l$ available and $S'_l \overset{r}{\Rightarrow}$ known, so on this cycle the algorithm will deduce that $S'_k \overset{r}{\Rightarrow}$.

The last possibility is that at least one path must eventually repeat a minimal state, since if all paths implement r and all states on such paths are known to implement r, then all minimal states are known to implement r. For any such loop, that loop will be the entire trajectory for all states on that loop. If some state in that loop is not known to reach some other state in that loop, then at least one such fact will be deduced this cycle: there will be some S'_i where a reaction $S'_i \xrightarrow{\mathcal{T}} S'_j$ is possible and $S'_j \overset{\mathcal{T}}{\Rightarrow} S'_k$ is known but $S'_i \overset{\mathcal{T}}{\Rightarrow} S'_k$ is not known; that fact will be deduced this cycle. If not, then there must be some y with $m(y) = \emptyset$ that is produced along that loop and some S'_i in that loop for which $S'_i \overset{\mathcal{T}}{\Rightarrow} S'_i + y$ is not known; otherwise one of the reactions in the loop is not the first reaction along the shortest path by which the appropriate $S'_j \overset{r}{\Rightarrow}$. If that S'_i is one such that $S'_i \xrightarrow{\mathcal{T}} S'_j + y$, then $S'_j \overset{\mathcal{T}}{\Rightarrow} S'_i$ is known (by assumption that all such facts in this loop are known), and $S'_i \overset{\mathcal{T}}{\Rightarrow} S'_i + y$ will be deduced this cycle. Otherwise, there will be an S'_i such that $S'_i \xrightarrow{\mathcal{T}} S'_j$ is possible, $S'_j \overset{\mathcal{T}}{\Rightarrow} S'_j + y$ is known but $S'_i \overset{\mathcal{T}}{\Rightarrow} S'_i + y$ is not known, and that fact will be deduced this cycle. This covers all the cases, and proves that if the permissive condition is true but not yet proven, then at least one fact will be deduced each cycle until the permissive condition is proven.

To complete the proof, we note that the number of facts is bounded above by $(z+1)n^k + n^{2k}$, where $z \leq n$ is the number of null species, and thus is poly(n^k). Thus, if the permissive condition is true, one of a finite number of facts will be learned each cycle until the permissive condition is proven. Furthermore, the algorithm will terminate one way or another in at most poly(n^k) cycles, thus poly(n^k) time. □

Although polynomial space is inefficient, in the general case we cannot do better. Two results in particular suggest a connection between CRNs and space-bounded Turing machines, the acceptance problem of which is known to be PSPACE-complete [32]; we use this connection to prove that verifying CRN bisimulation is PSPACE-complete. Jones et al. gave a construction to, given a space-bounded Turing machine with m states and tape size n, construct a Petri net (equivalently, a CRN) that directly simulates it, with poly(n, m) species and reactions [41]. Thachuk and Condon extended this connection to reversible CRNs, constructing a CRN that solves the known PSPACE-complete problem QSAT, proving a number of questions about CRNs and DNA strand displacement systems to be PSPACE-complete [69]. In the case of CRN bisimulation, if we have (on the order of) n^k minimal states, it is possible to embed a PSPACE-complete computation in the trivial reactions between those n^k states. Given any space-bounded Turing machine and input, we construct a formal and implementation CRN with interpretation, where the implementation CRN contains the construction of Jones et al. in the trivial

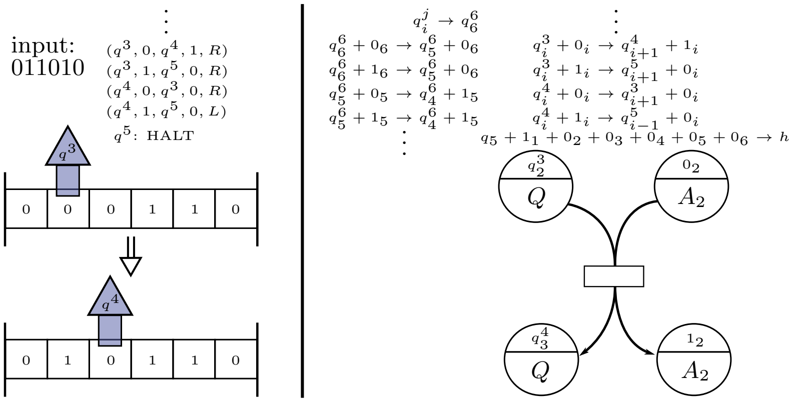

input:
011010
$(q^3, 0, q^4, 1, R)$
$(q^3, 1, q^5, 0, R)$
$(q^4, 0, q^3, 0, R)$
$(q^4, 1, q^5, 0, L)$
q^5: HALT
q^3
0 0 0 1 1 0
q^4
0 1 0 1 1 0
$q_i^j \to q_6^6$
$q_6^6 + 0_6 \to q_5^6 + 0_6$
$q_6^6 + 1_6 \to q_5^6 + 0_6$
$q_5^6 + 0_5 \to q_4^6 + 1_5$
$q_5^6 + 1_5 \to q_4^6 + 1_5$
$q_i^3 + 0_i \to q_{i+1}^4 + 1_i$
$q_i^3 + 1_i \to q_{i+1}^5 + 0_i$
$q_i^4 + 0_i \to q_{i+1}^3 + 0_i$
$q_i^4 + 1_i \to q_{i-1}^5 + 0_i$
$q_5 + 1_1 + 0_2 + 0_3 + 0_4 + 0_5 + 0_6 \to h$
q_2^3
Q
0_2
A_2
q_3^4
Q
1_2
A_2

Figure 2.12: An implementation CRN with a correct interpretation if and only if the corresponding space-bounded Turing machine accepts. The formal CRN has one species Q corresponding to the Turing machine head and one species A_i for each ith tape square; if all are present, they can react. The implementation CRN simulates the space-bounded Turing machine, with Turing machine head state species q_i^j all interpreted as Q and tape square species 0_i and 1_i both interpreted as A_i. All reactions involved in the simulation are thus trivial. If the simulation accepts, the formal reaction can be implemented. At any time the implementation CRN can use q_i^6 to "reset" to the start state on the given input, thus being able to "correctly" simulate the computation from an arbitrary initial implementation state. Thus this interpretation satisfies the permissive condition if and only if the Turing machine accepts.

reactions, plus some additional reactions specific to our case. Our construction is illustrated in Figure 2.12. There is one formal reaction that can only occur in an implementation state corresponding to the accept state of the Turing machine; thus, the state corresponding to the start state can implement that reaction if and only if the Turing machine does in fact accept. The additional reactions ensure that every minimal implementation state can implement the formal reaction if the "start state" can, making the interpretation a CRN bisimulation if and only if the space-bounded Turing machine accepts.

Theorem 2.5.3. *Verifying CRN bisimulation in the general case is PSPACE-complete.*

Proof. We are given a Turing machine with m states with tape alphabet $\{0, 1\}$, and an input x of length n. We assume the states are numbered such that q^0 is the start state and q^{m-1} is the halt state. We assume the Turing machine always halts while never using more space than the length of its input, and when it halts, it does so in state q^{m-1} reading the first square of its tape, with all squares but the first reading 0, and the first square reading 1 to indicate an accepting state and 0 to indicate rejecting. Given that, our formal CRN has $n+2$ species and 1 reaction, $Q + A_1 + \cdots + A_n \rightarrow H$. We construct an implementation CRN with species 0_i and 1_i for each spot on the tape $1 \leq i \leq n$, q_i^j for each tape spot $1 \leq i \leq n$ and state of the Turing machine $0 \leq j \leq m-1$, additional species for a "reset" state q_i^m for $1 \leq i \leq n$, and a halting species h. The implementation CRN contains reactions to simulate the Turing machine, reactions to reset the Turing machine to the start state q^0 on input x, and reactions to check whether a halting state is accepting or rejecting that can implement the formal reaction if and only if it is an accepting state.

To simulate the Turing machine, for each transition of the form, in state j reading symbol $\sigma \in \{0,1\}$, write symbol $\sigma' \in \{0,1\}$, transition to state j', and move (right,left) we have n reactions of the form $q_i^j + \sigma_i \rightarrow q_{i\pm1}^{j'} + \sigma_i'$ for $1 \leq i \leq n$, where the product is $q_{i+1}^{j'}$ if the move is right and $q_{i-1}^{j'}$ if the move is left. (If the move is right, the reaction for q_n^j is instead $q_n^j + \sigma_n \rightarrow q_n^{j'} + \sigma'_n$, and similarly if the move is left the reaction for q_1^j is instead $q_1^j + \sigma_1 \rightarrow q_1^{j'} + \sigma'_1$.)

To reset, we have a reaction $q_i^j \rightarrow q_n^m$ for every q_i^j including $j = m$ and q_1^{m-1} but not including any q_i^{m-1} for $i \geq 2$. We then have reactions $q_i^m + \sigma_i \rightarrow q_{i-1}^m + x_i$ for each $2 \leq i \leq n$ and $q_1^m + \sigma_1 \rightarrow q_1^0 + x_1$, in each case for both $\sigma = 0$ and $\sigma = 1$, where x_i represents the species 0_i if the ith character of the string x is 0 and 1_i if the ith character is 1.

To check whether a halting state is an accepting state, we have a reaction $q_1^{m-1} + 1_i \rightleftharpoons q_2^{m-1}$, reactions $q_i^{m-1} + 0_i \rightleftharpoons q_{i+1}^{m-1}$ for $2 \leq i \leq n-1$, and a reaction $q_n^{m-1} + 0_n \rightarrow h$. Note that by assumption q^{m-1} is the halting state of the given Turing machine and thus has no transitions; so other than these reactions the only implementation reaction with any q_i^{m-1} as a reactant is the reaction $q_1^{m-1} \rightarrow q_n^m$.

We want to check the validity of the interpretation m where $m(q_i^j) = Q$ for $j \neq m-1$, $m(q_i^{m-1}) = Q + \sum_{k=1}^{i-1} A_k$ (so for example $m(q_1^{m-1}) = Q$ and $m(q_3^{m-1}) = Q + A_1 + A_2$), $m(0_i) = m(1_i) = A_i$, and $m(h) = H$. However, we will show in Theorem 2.5.5 that for any CRN constructed this way based on a Turing machine, the only possible valid interpretations are this one up to a permutation of formal species, and either this interpretation is valid or no valid interpretation exists.

We use the three conditions formulation of validity. The atomic condition is always satisfied by $m(q_1^0) = Q$, $m(0_i) = A_i$, and $m(h) = H$, as well as many other ways. The delimiting condition is satisfied since under this interpretation, every reaction mentioned above is trivial except for the reaction $q_n^{m-1} + 0_n \to h$, which is interpreted as the one formal reaction $Q + A_1 + \cdots + A_n \to H$. It only remains to check the permissive condition, and we will show that the permissive condition is true if and only if the given Turing machine accepts the given input x.

The set of minimal states for the one formal reaction $r = Q+A_1+\cdots+A_n \to H$ is exactly the set of states containing either one copy of one q_i^j for $j \neq m-1$ and one copy of either 0_i or 1_i for each $1 \leq i \leq n$, or one copy of some q_i^{m-1} and one copy of either 0_k or 1_k for each $i \leq k \leq n$. Appealing to Lemma 2.5.1, the permissive condition is true if and only if each of those minimal states S' has $S' \overset{r}{\Rightarrow}$. We are particularly interested in the state S'_0 containing q_1^0 and the species x_i for each $1 \leq i \leq n$, where x_i represents either 0_i or 1_i depending on the ith character of x, which is minimal. Any minimal state of the first type with a q_i^j for $j \neq m-1$ can reach S'_0 by the reaction $q_i^j \to q_n^m$ followed by the reset reactions $q_i^m + \sigma_i \to q_{i-1}^m + x_i$ and $q_1^m + \sigma_1 \to q_1^0 + x_1$ in the appropriate order. Any minimal state of the second type with a q_i^{m-1} can reach S'_0 by the reverse reactions $q_{i+1}^{m-1} \to q_i^{m-1} + 0_i$ and $q_2^{m-1} \to q_1^{m-1} + 1_1$, as appropriate, of the reactions to check the halting state, followed by $q_1^{m-1} \to q_n^m$, followed by the rest of the reset reactions as appropriate. Since every minimal state can reach S'_0 via trivial reactions, the permissive condition is true if and only if $S'_0 \overset{r}{\Rightarrow}$.

Note that any state of the implementation CRN with exactly one copy of one q_i^j for $j < m-1$ and for each $1 \leq i \leq n$ exactly one of either 0_i or 1_i corresponds to the Turing machine state where the tape contents of square i is whichever of 0_i or 1_i is present, and the Turing machine is in state j reading square i where q_i^j is the q species present. S'_0 is one such state. Note also that in any such state, the only possible reactions are to either faithfully simulate the next transition of the Turing machine, leading to either another such state or a halting state (i.e. a state which would be a simulating state except that the q species present is q_1^{m-1}), or to start a reset. In a "resetting" state, which is a state where any q_i^m and for each $1 \leq i \leq n$ exactly one of either 0_i or 1_i is present, the only possible reaction is to continue the reset, leading to either another resetting state or eventually to a state that represents a Turing machine state.

If the Turing machine accepts x, then S_0' can faithfully simulate the Turing machine until it halts, which since the input was x, will be the accepting state $q_1^{m-1} + 1_1 + 0_2 + \cdots + 0_n$. From this state, the reactions to check whether a halting state is an accepting state are possible in order, eventually leading to the reaction $q_n^{m-1} + 0_n \to h$, which is interpreted as r; thus $S_0' \overset{r}{\Rightarrow}$ and the permissive condition is true. Conversely, if the Turing machine rejects x, then from S_0' the only possible trajectories are those that faithfully simulate the Turing machine with occasional resets. Some of those trajectories may reach a halting state, but since the Turing machine rejects x, that state will be $q_1^{m-1} + 0_1 + 0_2 + \cdots + 0_n$, and the reaction $q_1^{m-1} + 1_1 \to q_2^{m-1}$ will not be possible. None of these trajectories ever reach the reaction $q_n^{m-1} + 0_n \to h$, thus $S_0' \overset{r}{\not\Rightarrow}$ and the permissive condition is false. □

Finding an Interpretation

We now consider the problem of, given a formal and implementation CRN, can we find an interpretation that is a bisimulation or correctly assert that none exists? It is natural to consider performing an exhaustive depth-first search through the space of possible interpretations, testing each one to see if it satisfies the atomic, delimiting, and permissive conditions using the algorithms described above—thus either finding an interpretation or asserting that none exists. There are two major stumbling blocks to this approach. First, the space of possible interpretations is infinite, and thus we need some way to guarantee that if a valid interpretation exists, there must be one among a defined finite subset of interpretations that we can search. Second, to be useful in practice, the depth-first search must prune aggressively to eliminate fruitless branches.

The reactionsearch algorithm, presented below, addresses both of these challenges. Rather than directly exploring the space of interpretations, the reactionsearch algorithm organizes the depth-first search according to properties that the interpretation must have, effectively proceeding in five stages. First, as a precondition for the permissive condition, the algorithm ensures that every formal reaction has an implementation reaction that interprets to it; second, to satisfy the delimiting condition, the algorithm ensures that every remaining implementation reaction is interpreted as some formal reaction or is trivial; third, to satisfy the atomic condition, the algorithm ensures that every formal species has some implementation species that interprets to it; fourth, any unassigned implementation species are provided an interpretation that respects the assignment of implementation reactions as formal reactions or trivial reactions; and fifth, the permissive condition is tested on any such completed interpretation that is thus found. We will first describe the algorithm itself, and then discuss the lemmas that guarantee that a valid interpretation will be found if one exists.

Often, implementation CRNs are designed with specific interpretations in mind for some species, so it is reasonable to provide such information as an additional constraint on the search. Further, such a formulation enables a natural recursive definition for the algorithm (Algorithm 2.3). Thus, the algorithm takes as input a formal CRN $(\mathcal{S}, \mathcal{R})$, implementation CRN $(\mathcal{S}', \mathcal{R}')$, and a partial interpretation m which, for some (possibly empty) subset of $\mathcal{S}'$, specifies each $m(x) \in \mathbb{N}^{\mathcal{S}}$. The algorithm first constructs a table of, for each formal reaction $r \in \mathcal{R}$ or τ and for each implementation reaction $r' \in \mathcal{R}'$, whether r' can be interpreted as r (in some completion of the partial interpretation m, regardless of what that completion will do to the other reactions). The algorithm then enumerates interpretations by iterating, in an order described below, through all possible assignments of each r' to be interpreted as some r or τ, and enumerating completed interpretations which match that.

After constructing the table, if there is some $r \in \mathcal{R}$ with no r' that *is* interpreted as r (i.e., all species x involved in r' have $m(x)$ specified and $m(r') = r$), the algorithm chooses such an r with the smallest number of r' that *can be* interpreted as that r. (If there is an r with no r' that *can be* interpreted as r, then there is no completion of the partial interpretation that can satisfy the atomic and permissive conditions, so the algorithm returns that fact.) For each r' that can be interpreted as that r, the algorithm enumerates all possible interpretations of each species not yet interpreted by m and involved in r' that make $m(r') = r$. For each enumerated set of interpretations, the algorithm calls itself recursively to enumerate completions of the partial interpretation m with those new interpretations added. If a valid completion is found, the algorithm returns it; otherwise, the algorithm continues with the next partial interpretation or next r'.

If after constructing the table every $r \in \mathcal{R}$ has some r' that *is* interpreted as

```
def reactionsearch(CRN formal, CRN impl,
                   partial interpretation m):
  return complete(formal, impl, m, { })
def complete(CRN formal, CRN impl, partial interpretation m,
                                   assigned reactions k):
  table maybe(r, r’) = True if m does not rule out m(r’) = r ...
        for reaction r in formal or trivial, reaction r’ in impl
  if any r has no r’ or any r’ has no r where maybe(r,r’):
    return False
  let r in formal where no r’ in impl where m(r’) = r ...
      and which minimizes |{r’ in impl where maybe(r,r’)}|
  if such an r exists: for each r’ where maybe(r,r’):
      for each assignment of m’(x) where x in r’ ...
          and m(x) undefined such that (m U m’)(r’) = r:
        out = complete(formal, impl, m U m’, k U {r’})
        if out is an interpretation: return out
    return False # if no r’ is found
  if no such r exists:
    if (m(r’) is known or maybe(trivial, r’)) for all r’:
      for each assignment of, for each formal species A with no
            implementation species x with m(x) = A,
            one unassigned x to have m’(x) = A:
        out = solve_diophantine_equations(impl, m U m’)
        if out is an interpretation and permissive_check(out):
           return out
    if k includes all implementation reactions: return False
    let r’ in impl where r’ is not in k ...
        and which minimizes |{r in formal where maybe(r,r’)}|
    for each r in formal or trivial where maybe(r,r’):
      for each assignment of m’(x) where x in r’ ...
          and m(x) undefined such that (m U m’)(r’) = r:
        out = complete(formal, impl, m U m’, k U {r’})
        if out is an interpretation: return out
    return False # if no r is found
```

Algorithm 2.3: The reactionsearch algorithm to complete a partial interpretation or assert that no completion exists, in polynomial space.

r, the algorithm then finds the r' that has the fewest $r \in \mathcal{R}$ such that r' can be interpreted as r, and which hasn't yet been used for branching. (Again, if there is an r' with no $r \in \mathcal{R}$ *or* τ that r' can be interpreted as, then no completion of the partial interpretation can satisfy the delimiting condition, and the algorithm returns that.) For each such $r \in \mathcal{R}$, the algorithm as above enumerates all possible interpretations of each uninterpreted species involved in r' that make $m(r') = r$ and calls this algorithm recursively for each such partial interpretation. If r' can be interpreted as τ, then an additional recursive branch is explored wherein $m(r') = \tau$ is enforced.

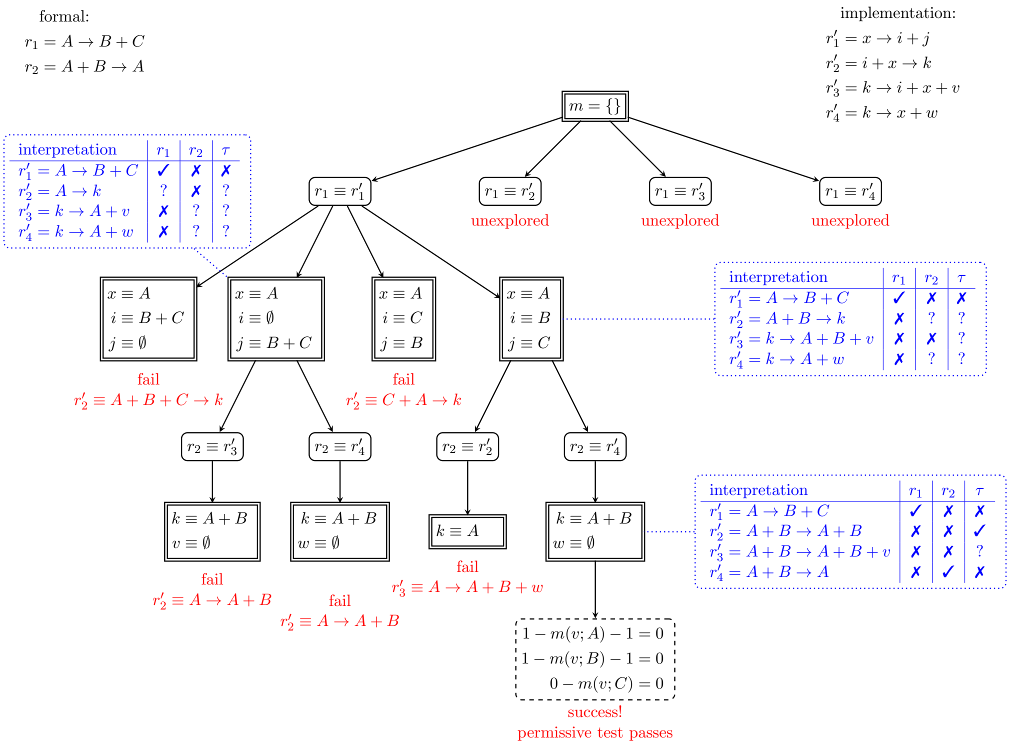
If all $r' \in \mathcal{R}'$ which involve uninterpreted species can be interpreted as τ, then on one branch the algorithm will consider the possibility that all of them are interpreted as τ, which as described above does not involve specifying any interpretations for the remaining uninterpreted species. Since the trivial reaction solver—which as described below will complete the interpretation for the uninterpreted species, if possible—works more efficiently with a partial interpretation which satisfies the atomic condition, the algorithm first ensures that that is the case. For each formal species A for which there is *no* implementation species x_A where the partial interpretation specifies $m(x_A) = A$, the algorithm lists all possible x_A for which, if $m(x_A) = A$ was added to the partial interpretation, all remaining reactions would still be able to be trivial. The algorithm then iterates over all combinations of choices of such x_A for each unimplemented formal species A, and runs the trivial reaction solver for each combination.

To find a completed interpretation in which all remaining r' are interpreted as τ, the trivial reaction solver sets up and solves a system of linear equations in variables $m(x; A)$ for each uninterpreted implementation species x and formal species A, where $m(x; A)$ is the count of A in $m(x)$. For each pair of an implementation reaction r' and a formal species A the algorithm derives one equation regarding various $m(x; A)$ by setting the sum of counts $m(x; A)$ in the (interpreted or uninterpreted) reactants of r' minus that of the products of r' to be 0. For example, if the implementation reaction $x_1 + x_2 \to x_3 + x_4$ should be interpreted as τ, $m(x_1) = A + C$ and $m(x_3) = B + C$ are specified while $m(x_2)$ and $m(x_4)$ are unspecified, then the algorithm will derive the equations $1 + m(x_2; A) - m(x_4; A) = 0$, $m(x_2; B) - 1 - m(x_4; B) = 0$, $m(x_2; C) - m(x_4; C) = 0$, and $m(x_2; D) - m(x_4; D) = 0$ for each other formal species D. Combining such equations for all remaining implementation reactions that are to be interpreted as trivial, we obtain a system of linear Diophantine equations where the variables specify the interpretations of all remaining uninterpreted species. The trivial reaction solver then runs an algorithm described by Contejean and Devie [21] that will find a minimal solution to a system of linear Diophantine equations, if any solution exists; this solution is then used as the interpretation of each remaining implementation species. (A minimal solution for a system of linear Diophantine equations is one such that no other solution has *every* variable being less than or equal to the given solution; thus there may be many minimal solutions.) We will prove in Lemma 2.5.4 that if there is any solution to these equations that satisfies the permissive condition, then the minimal solution returned by this algorithm does. If no solution to the equations exists, then no completed interpretation where all remaining $m(r') = \tau$ is possible, and the algorithm returns that.

Once the reactionsearch algorithm has a completed interpretation, which may be passed in initially, passed in by a recursive call, or found by the trivial reaction solver, it then runs either the loopsearch algorithm or the graphsearch algorithm as described previously, or any other algorithm yet to be discovered, in order to check the permissive condition. If the permissive condition is satisfied, then the given interpretation is valid and the reactionsearch algorithm returns that. If not, the algorithm passes that information to previous recursive calls in order to check further possible interpretations. If any level of recursion in the algorithm has checked all possible completed interpretations without finding a valid one, that level returns that no completion exists. An example of the depth-first search tree explored by the reactionsearch algorithm for a simple pair of CRNs is shown in Figure 2.13.

This completes the description of the reactionsearch algorithm. Now we turn to its correctness and its complexity. For correctness, the exhaustive depth-first search aspect of the algorithm is self-evident; the outstanding issue is whether the trivial reaction solver is guaranteed to find a solution if one exists.

Lemma 2.5.4. *Given a formal CRN $(\mathcal{S}, \mathcal{R})$ and implementation CRN $(\mathcal{S}', \mathcal{R}')$, let $m_0 : \mathcal{S}'' \to \mathbb{N}^{\mathcal{S}}$ be a partial interpretation on some $\mathcal{S}'' \subsetneq \mathcal{S}'$ which satisfies the atomic condition. If there exists any completion $m_1 : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ which agrees with m_0 on $\mathcal{S}''$, is a bisimulation, and is such that every implementation reaction r' involving at least one species not in $\mathcal{S}''$ has $m_1(r') = \tau$, then any*



interpretation	r_1	r_2	τ
$r_1' = A \to B + C$	✓	✗	✗
$r_2' = A \to k$	?	✗	?
$r_3' = k \to A + v$	✗	?	?
$r_4' = k \to A + w$	✗	?	?

interpretation	r_1	r_2	τ
$r_1' = A \to B + C$	✓	✗	✗
$r_2' = A + B \to k$	✗	?	?
$r_3' = k \to A + B + v$	✗	✗	?
$r_4' = k \to A + w$	✗	?	?

interpretation	r_1	r_2	τ
$r_1' = A \to B + C$	✓	✗	✗
$r_2' = A + B \to A + B$	✗	✗	✓
$r_3' = A + B \to A + B + v$	✗	✗	?
$r_4' = A + B \to A$	✗	✓	✗

Figure 2.13: A pictorial illustration of the search tree explored by the reactionsearch algorithm for the given pair of formal and implementation CRNs. Double-lined boxes indicate the new constraints on the partial interpretation at each node of the tree, where $x \equiv A$ is shorthand for $m(x) = A$. Rounded boxes indicate the new constraints on the interpretation of reactions, where $r \equiv r'$ is shorthand for requiring that $m(r') = r$. The dashed box indicates the Diophantine equation set up by the trivial reaction solver upon the first execution where it can successfully find a solution. The green dotted boxes illustrate the table of which implementation reactions may be interpreted as which formal reactions, for the given node in the tree.

minimal solution of the system of equations set up by the trivial reaction solver produces a completed interpretation m which is a bisimulation.

Proof. It is clear that, given m_0 as described, any such m_1 will correspond to a solution of the equations set up by the trivial reaction solver on m_0. It is also clear that any m_1 produced by a solution to the trivial reaction solver equations will satisfy the atomic and delimiting conditions if and only if m_0 does (since m_0 is assumed to satisfy the atomic condition), so we assume that m_0 satisfies the delimiting condition and are concerned only with the permissive condition. We first prove that if some solution to the equations produces an m_1 that satisfies the permissive condition (thus implying that a solution exists), then there is a minimal solution to the equations that produces an interpretation m that also satisfies the permissive condition. For any formal reaction $r = R \to P$ and implementation state S' with $m(S') \geq R$ it is also true that $m_1(S') \geq m(S') \geq R$, because either m_1 is minimal or there is a minimal solution m in which each value is the same or smaller than in m_1. Since m_1 satisfies the permissive condition, there is some implementation trajectory which, under m_1, is interpreted as $S' \overset{r}{\Rightarrow}$. Since m_1 and m agree in their interpretation of every implementation *reaction*, that trajectory under m is also interpreted as $S' \overset{r}{\Rightarrow}$; since r and S' were arbitrary, m satisfies the permissive condition. Since every solution to the trivial reaction equations is $\geq$ some minimal solution, this proves that if there is any solution that produces m_1 that satisfies the permissive condition, some minimal solution (in particular, the one $\leq$ it) produces m that also satisfies the permissive condition.

Having proven that at least one minimal solution produces a valid interpretation m, we show that every minimal solution does. In fact, the statement of the lemma is somewhat misleading: we show that under the above assumptions, for each formal species A that appears as a reactant in at least one formal reaction, the minimal solution to the equations for the counts of A is unique. For a formal species that never appears as a reactant, its counts in the interpretation of an implementation species cannot influence the permissive condition (given a fixed interpretation of every implementation reaction r', which is true by assumption), thus if some solution satisfies the permissive condition then every solution does. Now given m which satisfies the permissive condition, consider a complete interpretation m_2 generated by a distinct minimal solution which differs in at least one formal species that appears as a reactant. Then there is some $x \in \mathcal{S}'$ where $m(x)(A) > m_2(x(A))$ and $A \in R$ for some formal reaction $r = R \to P$ (where $m(x)(A)$ is the count of A in $m(x)$), and in particular choose x to minimize $m(x)(A)$ for the given A. Let $R_1 = R \backslash m(x)$ be the formal species in R not in $m(x)$, and let R_1' be the implementation state obtained by, for each formal species B in R_1, taking (the appropriate count of) the species x_B with $m(x_B) = m_2(x_B) = B$, which exists since the partial interpretation satisfies the atomic condition. Then for $R' = R_1' + x$, we have $m(R') \geq R$, so by the permissive condition there is a sequence of reactions which under m are interpreted as $R' \overset{r}{\Rightarrow}$. Let r'

be the last reaction in that sequence, which means $m(r') = r$, which since all unspecified reactions must be trivial means that r' involves only species in $\mathcal{S}''$ and $m_2(r') = r$ also. By removing r', we get a sequence of reactions $R' \overset{\mathcal{T}}{\Rightarrow} Y' + U'$, where $Y' \subset \mathcal{S}''$ and $U' \cap \mathcal{S}'' = \emptyset$. Then first of all, Y' contains all the reactants of r', and second, m and m_2 agree on every species in Y'. Since that trajectory consists of only trivial reactions under both interpretations, we must have $m(Y' + U') = m(R')$ and $m_2(Y' + U') = m_2(R')$; that is, $m(Y') + m(U') = m(R'_1) + m(x)$ and $m_2(Y') + m_2(U') = m_2(R'_1) + m_2(x)$. In particular, we have $m(U')(A) - m_2(U')(A) = m(x)(A) - m_2(x)(A)$. If $m(U')(A) = 0$, then this implies $m_2(U')(A) < 0$, an obvious contradiction. If $m(U')(A) > 0$, then there is some $x' \in U'$ such that $m(x')(A) > m_2(x')(A)$. But since $m(Y')(A) = R(A) > m(R')(A)$, this means $m(x')(A) \leq m(U')(A) < m(x)(A)$, which is also a contradiction since we assumed x was chosen to minimize $m(x)(A)$ of species for which $m(x)(A) > m_2(x)(A)$. So either way, given a partial interpretation which satisfies the atomic condition, if there is a complete interpretation which satisfies the three conditions in which all remaining reactions are trivial, then the trivial reaction solver will find one by searching for the first minimal solution. □

Theorem 2.5.4. *Given a formal CRN $(\mathcal{S}, \mathcal{R})$, implementation CRN $(\mathcal{S}', \mathcal{R}')$, and a partial interpretation which specifies $m(x)$ for some (possibly empty) set of various $x \in \mathcal{S}'$, whether a complete interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ exists that respects the given partial interpretation and is a bisimulation can be decided in polynomial space. In particular, if such an interpretation exists, then one exists that is polynomial size in that of the two CRNs and the partial interpretation.*

Proof. We prove that the reactionsearch algorithm described above outputs a correct completion of the partial interpretation if one exists and returns false if none exists; that it does so using only polynomial space; and that in particular if a correct interpretation is output then the interpretation is polynomial size in the input.

Most of the algorithm consists of, given a partial interpretation, trying out some number of ways to specify the interpretation of some species not yet specified, then calling the algorithm recursively on each of those more-specified partial interpretations. We will show that, if a correct completed interpretation exists, then at least one of those more-specified partial interpretations can be completed to that interpretation. Since the number of unspecified species and reactions decreases at each recursive call, the algorithm will eventually reach that completed interpretation. In any correct interpretation, for each formal reaction r, the atomic condition implies that there is an implementation state that interprets to the reactants of that reaction, and the permissive condition that that state must be able to eventually reach some r' with $m(r') = r$; in particular, such an r' must exist. So when the algorithm says, “if no r' is known to be interpreted as r, for each possible r' enumerate all possible ways that r' can be interpreted as r,” then if a correct completed interpretation exists, one of those possible ways must be part of it, provided that “enumerate all possible ways” can be done in finite time and in particular in polynomial space. Similarly, by the delimiting condition every r' must have either $m(r') = r$ for some r or $m(r') = \tau$, so when the algorithm considers all those possibilities, one of them must be part of the correct completed interpretation if one exists. On the branch where the algorithm has found all r' with $m(r') \neq \tau$ in the correct interpretation, it will run the trivial reaction solver for all remaining $m(r') = \tau$. At any given time the algorithm only needs to store a single partial interpretation for each of at most $|\mathcal{S}'| + |\mathcal{R}'|$ layers of recursive calls (since each one will specify the interpretation of at least one implementation species or reaction), plus whatever information is needed to “enumerate all possible interpretations” or run the trivial reaction solver or a permissive condition test, all of which we will prove take polynomial space (with the permissive condition tests already proven). So all we have left to prove is that enumerating “all possible interpretations” of each uninterpreted species in some r' such that $m(r') = r$ for a specific r, or enumerating “all possible interpretations” for achieving the atomic condition, can be done in finite time and polynomial space, and that the trivial reaction solver works and takes polynomial space.

We first address enumerating all possible interpretations of each species in a given r' such that $m(r') = r$ for a given r. If $r' = R' \to P'$ and $r = R \to P$, then $m(r') = r$ if and only if $m(R') = R$ and $m(P') = P$. In particular, if in a partial interpretation r' can be interpreted as r, then the interpreted part of $m(R')$ must be $\leq R$ and similarly the interpreted part of $m(P')$ must be $\leq P$. By taking the difference of R minus the partial interpretation of R' and taking all assignments of each formal species in that difference to some uninterpreted species in R', doing the same for P and P', and removing any assignments that self-contradict, we can enumerate all possible partial interpretations where

$m(r') = r$. To clarify, multiple copies of the same formal species in R or P can be assigned to different implementation species, but different copies of the same implementation species in R' and/or P' must be assigned the exact same multiset of formal species, otherwise the interpretation self-contradicts. Since all assignments of at most k copies each of i objects to j boxes can be enumerated in $\text{poly}(i, j, \log k)$ space, this process takes polynomial space. Since the interpretation of any given species involved in r' where $m(r') = r = R \to P$ must be $\leq R$ or $\leq P$, the partial interpretation throughout this entire part of the algorithm is bounded by the size of the formal CRN (except for larger interpretations provided in the initial partial interpretation), and is thus polynomial size.

Lemma 2.5.4 proves that if at the point when the trivial reaction solver is called a valid completion of the interpretation exists (where all remaining reactions are trivial), then the trivial reaction solver will find one and it will be a minimal solution of the given equations. It remains to show that the trivial reaction solver runs in polynomial space and produces a polynomial-size interpretation. The trivial reaction solver runs, for each formal species, the stack-based algorithm given by Contejean and Devie [21] to solve a system of linear Diophantine equations. Where q is the number of variables in the system, i.e. the number of implementation species whose interpretation is not yet specified, Contejean and Devie prove that their algorithm stores at most q states at one time on the stack, each of which is a tuple of q integers. Pottier has proven a bound on the size of those integers, namely, that their sum is at most, in that notation, $(1 + ||A||_{1,\infty})^r$ [54]. There r is bounded above by the number of unknown implementation reactions and $||A||_{1,\infty}$ is the maximum of any individual equation (i.e., unknown implementation reaction) of the sum of coefficients in that equation (coefficients of unknown implementation species in the reaction, or of formal species in the known interpretations involved). The "size" of the given interpretation is bounded by q times the logarithm of the bound on an individual count, i.e. $|m| \leq qr(1 + \log ||A||_{1,\infty})$, where $\log ||A||_{1,\infty}$ itself is the "size" of some combination of the implementation CRN and the current partial interpretation. The implementation CRN is of course part of the input to the algorithm, and we have proven that the partial interpretation up to this point is polynomial in the size of the input; thus the entire algorithm runs in space polynomial in its input, and if a correct interpretation exists, then one exists which is polynomial size. □

In the general case, finding an interpretation turns out to be just as hard as checking an interpretation; in fact, the same space-bounded Turing machine reduction from Theorem 2.5.3 applies.

Theorem 2.5.5. *Whether a bisimulation interpretation exists from a given implementation CRN to a given formal CRN is PSPACE-complete.*

Proof. Theorem 2.5.4, with Theorem 2.5.1 for checking the permissive condition, prove that a bisimulation can be found, or shown that none exists, in polynomial space. To prove completeness, we use the same formal and implementation CRN used in Theorem 2.5.3 and shown in Figure 2.12. Consider an arbitrary Turing machine with m states and tape alphabet $\{0, 1\}$, with start state q^0 and halt state q^{m-1} which on any input, halts without using more space than the length of the input, with the tape reading 10^{n-1} if it accepts and 0^n if it rejects; also consider an input x with length n. Given that, the formal CRN has $n+2$ species and 1 reaction, $Q + A_1 + \cdots + A_n \to H$. The implementation CRN has species 0_i and 1_i for each tape spot i, q_i^j for each tape spot i and each Turing machine state j, additional q_i^m for a "reset" state m and each tape spot, and a halt species h. As described in Theorem 2.5.3, the implementation CRN can simulate the Turing machine with reactions $q_i^j + \sigma_i \to q_{i\pm1}^{j'} + \sigma_i'$; can "reset" the computation to the start state reading string x with reactions $q_i^j \to q_n^m$ (for all q_i^j except q_i^{m-1} for $i > 1$), $q_i^m + \sigma_i \to q_{i-1}^m + x_i$, and $q_1^m + \sigma_1 \to q_0^1 + x_1$; and can check whether the computation has accepted with reactions $q_1^{m-1} + 1_1 \rightleftharpoons q_2^{m-1}$, $q_{i-1}^{m-1} + 0_{i-1} \rightleftharpoons q_i^{m-1}$ for $i > 2$, and $q_n^{m-1} + 0_n \to h$. We showed in Theorem 2.5.3 that the interpretation $m(0_i) = m(1_i) = A_i$, $m(h) = H$, $m(q_i^j) = Q$ for $j \neq m-1$ and $m(q_i^{m-1}) = Q + \sum_{k<i} A_k$ is valid if and only if the given Turing machine accepts the string x, thus proving that checking an interpretation is PSPACE-complete. To prove that finding an interpretation is PSPACE-complete, we show that aside from permutations of the formal species in that interpretation (which are also correct if and only if the Turing machine accepts x), no other interpretation can be correct; therefore, a correct interpretation exists if and only if the Turing machine accepts x.

To prove that only one correct interpretation (up to permutations) is possible, we use the three conditions to eliminate possibilities until only that one remains. First, by applying the delimiting condition to the reactions

$q_i^j \to q_n^m$, either all q_i^j (except q_i^{m-1} for $i > 1$) must have the same interpretation, or some subset of them (including q_n^m) must interpret to H and the rest to $Q + A_1 + \cdots + A_n$. We can quickly eliminate the cases $m(q_n^m) = \emptyset$ and $m(q_n^m) = H$, and prove that $m(0_i) = m(1_i)$ for every i.

If $m(q_n^m) = \emptyset$, then by the atomic condition there must be some state which interprets to exactly $Q + A_1 + \cdots + A_n$ by, for each of those formal species, selecting one implementation species that interprets to exactly one of that formal species and nothing else. For each q_i^j that appears, apply the appropriate sequence of reactions from $q_i^{m-1} \to q_{i-1}^{m-1} + 0_{i-1}$, $q_2^{m-1} \to q_1^{m-1} + 1_1$, and $q_i^j \to q_n^m$ until the only q_i^j that appears is some number of copies of q_n^m. All these reactions must be trivial (or the delimiting condition is violated): each of the q_i^{m-1}-involving reactions are reversible while the formal reaction is not, while if $m(q_n^m) = \emptyset$ then any $q_i^j \to q_n^m$ reaction has product $\emptyset$ while no formal reaction does; thus the resulting implementation state has the same interpretation as the original, namely $Q + A_1 + \cdots + A_n$. Since $m(q_n^m) = \emptyset$, by removing all copies of it we get another implementation state with interpretation $Q + A_1 + \cdots + A_n$ but no copies of q_i^j for any i, j. Since every implementation reaction has some q_i^j as a reactant, no reactions can fire in this state and the permissive condition is violated.

If $m(q_n^m) = H$ then $m(q_i^m) = m(q_1^0) = H$ because the "reset" reactions $q_i^m + x_i \to q_{i-1}^m + x_i$ and $q_1^m + x_1 \to q_1^0 + x_1$ (using the notation x_i for 0_i if the ith symbol of the string x is 0 and 1_i if it is 1, and x_{-i} for 1_i if the ith symbol of x is 0 and 0_i if it is 1; these are the reactions used in the event that the tape already has the correct symbol) must be trivial (since no formal reaction has an H as a reactant), and the only difference between the two sides is the two q's. Since if $m(q_n^m) \neq H$ we know $m(q_i^m) = m(q_1^0) = m(q_n^m)$ anyway, the reset reactions $q_i^m + x_{-i} \to q_{i-1}^m + x_i$ and $q_1^m + x_{-1} \to q_1^0 + x_1$ prove that $m(0_i) = m(1_i)$ for all i: the reaction must be trivial since $m(q_n^m) \neq \emptyset$ and no formal reaction is catalytic. By the atomic condition, there are $n+1$ species $Q, A_1, \ldots, A_n$ which must each have some implementation species that interprets as one copy of that species, and since $m(0_i) = m(1_i)$ the 0_i's and 1_i's can account for at most n of them. Call the remaining species X, and note that by assumption either $m(q_1^{m-1}) = H$ or $m(q_1^{m-1}) = Q + A_1 + \cdots + A_n$. If $m(h) = X$, then the sequence of reactions $q_1^{m-1} + 1_i \to q_2^{m-1}$, $q_{i-1}^{m-1} + 0_{i-1} \to q_i^{m-1}$ for $3 \leq i \leq n$ in order, then $q_n^{m-1} + 0_n \to h$ is a sequence which takes a state whose interpretation contains either an H or *all* of $Q, A_1, \ldots, A_n$ to a state which has no H and only one of $Q, A_1, \ldots, A_n$; this is impossible in the formal CRN, thus at least one implementation reaction on that pathway must be nontrivial and not a formal reaction. The only remaining implementation species are the q_i^j's, and we have already shown that if $m(q_n^m) = H$ then $m(q_i^j)$ for every $j \neq m$ is either H or $Q + A_1 + \cdots + A_n$, and cannot be just X. That leaves only some q_i^{m-1} for $i \geq 2$, but since the reversible reactions must be trivial, $m(q_i^{m-1}) = m(q_1^{m-1}) + \sum_{k<i} m(0_k)$, which must contain either an H or all of $Q + A_1 + \cdots + A_n$, and cannot be just X. Thus it is impossible to have $m(q_n^m) = H$, and we must have $m(q_i^j) = m(q_n^m)$ whenever $j \neq m - 1$.

So far we know that $m(0_i) = m(1_i)$ for all i, $m(q_n^m)$ is neither $\emptyset$ nor H, $m(q_i^j) = m(q_n^m)$ for $j \neq m - 1$, and since the reversible reactions must be trivial $m(q_i^{m-1}) = m(q_n^m) + \sum_{k<i} m(0_i)$. In order to satisfy the atomic condition for $n + 2$ formal species, we need $n + 2$ implementation species with distinct interpretations. Regardless of the interpretations of any other species, $m(q_i^{m-1}) \geq m(q_n^m)$ so no q_i^{m-1} can be interpret as a single formal species other than the one (if any) that q_n^m is interpreted as, so all the q_i^j's can satisfy the atomic condition for at most one formal species. Each pair of 0_i and 1_i must have the same interpretation, so all the 0_i's and 1_i's can satisfy at most n species, and h can satisfy an additional one, but no other implementation species remain. So since we have $n+2$ "categories" of implementation species that can possibly be interpreted as a single formal species and $n + 2$ formal species, each such group must be interpreted as a distinct formal species. Given that, the reaction $q_n^{m-1} + 0_n \to h$ cannot be trivial, so it must be interpreted as $Q + A_1 + \cdots + A_n \to H$; in particular, we must have $m(h) = H$. The remaining constraints say that each 0_i and q_n^m is interpreted as a distinct one of Q or some A_i, that $m(1_i) = m(0_i)$, that $m(q_i^j) = m(q_n^m)$ for $j \neq m - 1$, and that $m(q_i^{m-1}) = m(q_n^m) + \sum_{k<i} m(0_i)$; this is exactly the interpretation given in Theorem 2.5.3, up to a permutation of the formal species Q and the A_i's. Any such interpretation will satisfy the atomic and delimiting conditions, and will satisfy the permissive condition if and only if the given Turing machine accepts the string x, thus finding a correct interpretation is as hard as deciding whether a linear bounded Turing machine accepts a given string, which is PSPACE-complete. □

When the number of formal reactants is bounded by a constant, we showed that whether an interpretation is valid can be checked in polynomial time. Then finding an interpretation is a natural NP problem; we show that it is in fact NP-complete, with a reduction from 3-SAT. An example of this reduction is shown in Figure 2.14.

Theorem 2.5.6. *When the number of reactants in a formal reaction k is bounded by a constant $k \geq 1$, whether a bisimulation interpretation exists is NP-complete.*

Proof. If a valid interpretation exists, Theorem 2.5.4 guarantees that we can find a valid polynomial-size interpretation which can be checked in polynomial time by Theorem 2.5.2.

To prove NP-completeness, given an arbitrary 3-SAT formula we construct a formal and implementation CRN such that a valid interpretation exists if and only if the formula is satisfiable. Our formal CRN has two species C and T and three reactions $C \to T$, $C \to 2T$, and $C \to 3T$. Our implementation CRN has two species s_C and s_T plus for each variable x_i in the 3-SAT formula two species x_i^t and x_i^f. We encode each clause of our 3-SAT formula e.g. $(x_1 \vee \neg x_2 \vee x_3)$ as an implementation reaction e.g. $s_C \to x_1^t + x_2^f + x_3^t$. Formally, we say that a clause is $(l_1 \vee l_2 \vee l_3)$ where each l_j is some x_i or $\neg x_i$, and it is encoded in the implementation reaction $s_C \to v_1 + v_2 + v_3$, where v_j is x_i^t if l_j is x_i, or x_i^f if l_j is $\neg x_i$. We also add the implementation reactions $s_C \to s_T$, $s_C \to 2s_T$, and $s_C \to 3s_T$, and for each x_i the reaction $s_T \rightleftharpoons x_i^t + x_i^f$ which we will show restrict interpretations that satisfy the three conditions to correspond to satisfying assignments of the 3-SAT formula.

We again observe that none of the formal reactions are reversible, so in order to satisfy the delimiting condition the reactions $s_T \rightleftharpoons x_i^t + x_i^f$ must be trivial. Note that all other implementation reactions have exactly s_C as their reactants. The combination of atomic and permissive conditions implies that each formal reaction must have at least one implementation reaction that is interpreted as it; since all three formal reactions have reactants C and all remaining implementation reactions have reactants s_C, any valid interpretation must have $m(s_C) = C$. Now the reactions $s_C \to s_T$ and $s_C \to 2s_T$ cannot both be trivial, since that would imply $m(s_T) = 2m(s_T)$ and thus $m(s_T) = \emptyset$, which from the reactions $s_T \rightleftharpoons x_i^t + x_i^f$ implies $m(x_i^t) = m(x_i^f) = \emptyset$, leaving not enough species to satisfy the atomic condition. Therefore at least one of

3-SAT problem	Formal CRN	Implementation CRN
	$C \rightarrow T$	$s_C \rightarrow s_T$
	$C \rightarrow 2T$	$s_C \rightarrow 2s_T$
	$C \rightarrow 3T$	$s_C \rightarrow 3s_T$
$(x_1 \vee \neg x_2 \vee x_3)$	$(C \rightarrow 2T)$	$s_C \rightarrow x_1^t + x_2^f + x_3^t$
$\wedge(\neg x_1 \vee x_4 \vee \neg x_2)$	$(C \rightarrow 2T)$	$s_C \rightarrow x_1^f + x_4^t + x_2^f$
$\wedge(x_2 \vee \neg x_3 \vee \neg x_4)$	$(C \rightarrow T)$	$s_C \rightarrow x_2^t + x_3^f + x_4^f$
	$(T \rightleftharpoons T)$	$s_T \rightleftharpoons x_1^t + x_1^f$
	$(T \rightleftharpoons T)$	$s_T \rightleftharpoons x_2^t + x_2^f$
	$(T \rightleftharpoons T)$	$s_T \rightleftharpoons x_3^t + x_3^f$
	$(T \rightleftharpoons T)$	$s_T \rightleftharpoons x_4^t + x_4^f$

satisfying assignment	valid interpretation
$x_1 =$ true	$m(x_1^t) = T$, $m(x_1^f) = \emptyset$
$x_2 =$ false	$m(x_2^t) = \emptyset$, $m(x_2^f) = T$
$x_3 =$ false	$m(x_3^t) = \emptyset$, $m(x_3^f) = T$
$x_4 =$ true	$m(x_4^t) = T$, $m(x_4^f) = \emptyset$
	$m(s_C) = C$, $m(s_T) = T$

Figure 2.14: Example formal and implementation CRN corresponding to an instance of the 3-SAT problem. Top left: example 3-SAT instance. Top middle: the formal CRN. Eventual interpretations of the corresponding implementation reactions are given in parentheses. Top right: the implementation CRN corresponding to this 3-SAT instance. Each 3-SAT clause has a corresponding implementation reaction, with auxiliary implementation reactions added. Bottom left: a satisfying assignment for the 3-SAT formula. Bottom right: a valid CRN bisimulation interpretation for the two CRNs, which the reactionsearch algorithm would find. Note the correspondence between satisfying assignment and interpretation: if x_i is true then $m(x_i^t) = T$ and $m(x_i^f) = \emptyset$, otherwise $m(x_i^t) = \emptyset$ and $m(x_i^f) = T$. Such an interpretation is a CRN bisimulation if and only if the corresponding assignment satisfies the 3-SAT formula; if no satisfying assignment exists, then no valid interpretation exists.

those two reactions must be formal; the only way to satisfy that is $m(s_T) = T$, making those two reactions and $s_C \to 3s_T$ interpreted as the three formal reactions respectively, and also satisfying the atomic condition.

Given that $m(s_C) = C$ and $m(s_T) = T$, since the reversible reactions are all trivial we have for each i, $m(x_i^t) + m(x_i^f) = T$. In other words, exactly one of $m(x_i^t)$ or $m(x_i^f)$ is T, and the other is $\emptyset$. Such an interpretation satisfies the atomic condition with $m(s_C) = C$ and $m(s_T) = T$, and satisfies the permissive condition since any state whose interpretation has a C has an s_C, and can do the reactions $s_C \to s_T$, $s_C \to 2s_T$, or $s_C \to 3s_T$ for whichever formal reaction is desired; further, those three reactions and the reversible $s_T \rightleftharpoons x_i^t + x_i^f$ satisfy the delimiting conditions, leaving only the reactions for each clause of the 3-SAT formula. These interpretations have an obvious one-to-one correspondence with assignments to the variables in the 3-SAT formula: x_i is assigned to true if $m(x_i^t) = T$, or to false if $m(x_i^f) = T$. Each reaction $s_C \to v_1 + v_2 + v_3$, corresponding to a clause $(l_1 \vee l_2 \vee l_3)$, will be interpreted as either $C \to \emptyset$, $C \to T$, $C \to 2T$, or $C \to 3T$, depending on how many of l_1, l_2, and l_3 are true in the corresponding assignment. Specifically, the interpretation will satisfy the delimiting condition (none of the clause reactions are interpreted as $C \to \emptyset$) if and only if the assignment satisfies the formula (none of the clauses have no true variables). Thus a valid interpretation exists if and only if the formula has a satisfying assignment, which completes the proof of NP-completeness. □

Using the Modularity Condition

Because finding and checking interpretations are in some cases computationally intractable, any way of reducing the size of the problem would be helpful. Often, a larger implementation CRN can be broken up into a number of smaller modules. The modularity condition of Definition 2.4.3 shows that each module can be checked individually and combined into a correct implementation, as described in Theorem 2.4.2 and Corollary 2.4.3. We show that the reactionsearch algorithm from Section 2.5 can be modified to iterate through a number of correct interpretations such that if any correct interpretation is modular with respect to the given sets of common formal and implementation species, then one of the enumerated interpretations is. We also show that whether an interpretation is modular with respect to given sets of common formal and implementation species can be checked in polynomial time, and present an algorithm to do so. We begin by proving Lemma 2.5.5, which provides the mathematical foundation for the modified reactionsearch algorithm. Then, Lemma 2.5.6 presents an algorithm for testing modularity and establishes its running time. Finally, Theorem 2.5.7 presents the modified reactionsearch algorithm for finding a modular interpretation and establishes its running time. Note that the user will be responsible for identifying the modules and the common species; the modified reactionsearch algorithm will be responsible for finding a valid modular interpretation with respect to those modules, if one exists. Thus, below, the formal CRN and implementation CRN that we discuss will be only those species and reactions relevant to verification of a single module at a time.

Recall that the reactionsearch algorithm for finding an interpretation was based on Lemma 2.5.4 which proved that if a valid completion of an interpretation exists when the trivial reaction solver is called, the trivial reaction solver will find a valid completed interpretation. Our first step is to show that if at the same point a valid *modular* completion exists, then the trivial reaction solver will find one.

Lemma 2.5.5. *Given a formal CRN $(\mathcal{S}, \mathcal{R})$, implementation CRN $(\mathcal{S}', \mathcal{R}')$, and sets of common species $\mathcal{S}_0$ and $\mathcal{S}_0'$, let $m_0 : \mathcal{S}'' \to \mathbb{N}^{\mathcal{S}}$ be a partial interpretation defined on some set of implementation species $\mathcal{S}''$ with $\mathcal{S}_0' \subset \mathcal{S}'' \subsetneq \mathcal{S}'$, where m_0 satisfies the atomic condition. If there exists any completion $m_1 : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ that agrees with m_0 on S'', is a modular bisimulation with respect to $\mathcal{S}_0$ and $\mathcal{S}_0'$, and such that every reaction r' involving at least one species not in S'' has $m_1(r') = \tau$, then any minimal solution of the system of equations set up by the trivial reaction solver produces a completed interpretation m which is a modular bisimulation with respect to $\mathcal{S}_0$ and $\mathcal{S}_0'$. If every formal species in $\mathcal{S}_0$ appears as a reactant in $\mathcal{R}$, then the previous statement holds even if $\mathcal{S}_0' \not\subset \mathcal{S}''$.*

As a remark, we give two conditions of which only one is required for the lemma to hold: $\mathcal{S}_0' \subset \mathcal{S}''$ (the interpretation of the common implementation species is already known) *or* every formal species in $\mathcal{S}_0$ appears in a reaction. In the typical use case of this algorithm, namely a systematic DNA strand displacement implementation to which we want to apply Corollary 2.4.3, the first condition holds but the second does not. In such a system, $\mathcal{S}_0'$ is typically the set of all signal species $\{x_A, x_B, \dots\}$, for which we "know" that $m(x_A) = A$

etc., at least in the sense that any interpretation where that is not true is not interesting. However, in such a system $\mathcal{S}_0 = \mathcal{S}$ is the set of all formal species, in that each module officially contains all formal species of the larger CRN even if only some of them appear in a reaction in any given module.

Proof. First, we show that if some solution to the trivial reaction solver equations produces a modular bisimulation, then some minimal solution does. Let m_1 be the modular bisimulation produced by the solution that exists by assumption, and let m be the interpretation produced by an arbitrary minimal solution $\leq$ that solution; so for all x, $m(x) \leq m_1(x)$. Lemma 2.5.4 shows that m is a bisimulation, so we only need to prove it is modular. Given any x, there is a sequence of trivial reactions (since any two solutions agree on the interpretation of any reaction, the sequence is trivial under both m_1 and m) by which $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$, where $Y \subset \mathcal{S}_0'$ and $m_1(Z) \cap \mathcal{S}_0 = \emptyset$. Since $m \leq m_1$, in particular $m(Z) \leq m_1(Z)$ and $m(Z) \cap \mathcal{S}_0 = \emptyset$, so the same sequence of trivial reactions satisfies the modularity condition for m. Since this applies for all x, m is also modular.

Now that there exists at least one minimal solution which produces a modular bisimulation, let m be that modular bisimulation and m_2 a bisimulation produced by another minimal solution. Given that the interpretation of every reaction is the same, we know we can treat the solution for each formal species separately. For species $B \notin \mathcal{S}_0$, if $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$ for $Y \subset \mathcal{S}_0'$ and $m(Z) \cap \mathcal{S}_0 = \emptyset$ then m_2 disagreeing with m on the count of B in any species will not affect whether $m_2(Z) \cap \mathcal{S}_0 = \emptyset$; if m and m_2 only disagree on species not in $\mathcal{S}_0$ then m_2 is modular. (Since Lemma 2.5.4 proves that if any minimal solution produces a bisimulation then the minimal solution is unique on all formal species that appear as a reactant, if every species in $\mathcal{S}_0$ appears in a reaction then the above completes the proof without requiring $\mathcal{S}_0' \subset \mathcal{S}''$.) If m and m_2 disagree on any $m(x)(A)$ for $A \in \mathcal{S}_0$, then take any such x look at the reactions by which $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$ with $Y \subset \mathcal{S}_0'$ and $m(Z) \cap \mathcal{S}_0 = \emptyset$. Now using the assumption $\mathcal{S}_0' \subset \mathcal{S}''$ so m and m_2 agree on any species in $\mathcal{S}_0'$, in particular $m(x)(A) = m(Y)(A) = m_2(Y)(A)$, which since $m(Z)(A) = 0$ implies $m_2(Z)(A) = m_2(x)(A) - m(x)(A) > 0$. Since each formal species can be taken independently, if $m_2(x)(A) > m(x)(A)$ whenever they disagree then an interpretation m_3 defined as $m_3(x)(A) = m(x)(A)$ and $m_3(x)(B) = m_2(x)(B)$ for $B \neq A$ is also a solution, but $m_3 \leq m_2$ so m_2 is not minimal, a contradiction.

This proves that the minimal solution to the trivial reaction solver is unique and correct for any formal species A that appears as a reactant in a formal reaction (Lemma 2.5.4) or is in $\mathcal{S}_0$ (assuming $\mathcal{S}'_0 \subset \mathcal{S}''$), and any other formal species can't affect whether the interpretation is a modular bisimulation; so if a modular bisimulation exists, then given the conditions, any minimal solution to the trivial reaction equations produces one. □

Given a formal and implementation CRN $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$, an interpretation m which we already know is a bisimulation, and sets of common species $\mathcal{S}_0$ and $\mathcal{S}'_0$, we can check whether m is modular with respect to $\mathcal{S}_0$ and $\mathcal{S}'_0$ using an algorithm similar to the graphsearch algorithm for checking the permissive condition. First, construct a table which for each implementation species $x \in \mathcal{S}'$ stores either that x is "finished", or if it is not finished, stores a list of all $x' \in \mathcal{S}'$ such that $m(x') = x$ and $x \overset{\mathcal{T}}{\Rightarrow} x' + Z$ is known (x "can reach" x') and a list of all $z \in \mathcal{S}'$ such that $x \overset{\mathcal{T}}{\Rightarrow} x + z$ is known (x "can produce" z; this implies $m(z) = \emptyset$). Here "x is finished" means either it is known that $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$ for $Y \subset \mathcal{S}'_0$ and $m(Z) \cap \mathcal{S}_0 = \emptyset$, or it is known that $x \overset{\mathcal{T}}{\Rightarrow} x'_1 + X'$ for $\emptyset < m(x'_1) < m(x)$. We will show later that if all x meet one of those two conditions, then m is modular with respect to $\mathcal{S}_0$ and $\mathcal{S}'_0$. Initialize the table to say that x is finished if $x \in \mathcal{S}'_0$ or $m(x) \cap \mathcal{S}_0 = \emptyset$, otherwise x is known to reach itself and not known to produce anything. Then in cycles, for each x not yet finished where Z_x is the set of all z such that $x \overset{\mathcal{T}}{\Rightarrow} x + z$ is known, for each trivial reaction that can happen in a state with one x and arbitrarily many copies of Z_x (written $x + \infty Z_x \xrightarrow{\mathcal{T}} \dots$), update the table according to the following rules:

(i) If $x + \infty Z_x \xrightarrow{\mathcal{T}} Y$ where every species in Y is finished, then x is finished.

(ii) If $x + \infty Z_x \xrightarrow{\mathcal{T}} x'_1 + X'$ where $\emptyset < m(x'_1) < m(x)$, then x is finished.

(iii) If $x + \infty Z_x \xrightarrow{\mathcal{T}} x' + Z$ where $m(x') = m(x)$ (implying $m(Z) = \emptyset$), then x can reach x' and x can reach any species that x' can reach.

(iv) If $x + \infty Z_x \xrightarrow{\mathcal{T}} x' + Z$ where $m(x') = m(x)$ and x' can reach x, then x can produce any species $z \in (Z \cup Z_{x'})$.

Continue until one cycle (checking every trivial reaction for every unfinished x) passes with no changes to the table. At that time, if every x is finished then m is modular, otherwise m is not modular (with respect to $\mathcal{S}_0$ and $\mathcal{S}'_0$).

Lemma 2.5.6. *Given a formal and implementation CRN $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$, a bisimulation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$, and common species sets $\mathcal{S}_0 \subset \mathcal{S}$ and $\mathcal{S}'_0 \subset \mathcal{S}'$, whether m is modular with respect to $\mathcal{S}_0$ and $\mathcal{S}'_0$ can be checked in polynomial time.*

Proof. We prove that the above algorithm is correct and runs in polynomial time. For correctness, first, the table is initialized with true facts and updated with true deductions: for initialization, if $x \in \mathcal{S}'_0$ then $x \overset{\mathcal{T}}{\Rightarrow} x + \emptyset$, and if $m(x) \cap \mathcal{S}_0 = \emptyset$ then $x \overset{\mathcal{T}}{\Rightarrow} \emptyset + x$, as sequences of 0 reactions fulfill $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$ for $Y \subset \mathcal{S}'_0$ and $m(Z) \cap \mathcal{S}_0 = \emptyset$. The deductions in general follow from that, if $x \overset{\mathcal{T}}{\Rightarrow} x + Z_x$ and $x + \infty Z_x \xrightarrow{\mathcal{T}} S'$, then $x \overset{\mathcal{T}}{\Rightarrow} S'$. The only non-obvious one is that if $x \overset{\mathcal{T}}{\Rightarrow} Y'$ where every species in Y' is finished, then x is finished; this follows from induction on the order in which the algorithm marks species as finished: if every species x' marked as finished before x satisfies one of the two $x' \overset{\mathcal{T}}{\Rightarrow} \ldots$ conditions in the definition of finished, and $x \overset{\mathcal{T}}{\Rightarrow} Y'$ made up of only those species, then x satisfies one of the two conditions. Then we need to prove that if every $x \in \mathcal{S}'$ is finished then every $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$ as desired: the proof is by induction on $|m(x)|$, proving that if $x' \overset{\mathcal{T}}{\Rightarrow} Y' + Z'$ for every x' with $m(x') < m(x)$ and x is finished then $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$. Recall that "$x$ is finished" is defined as, either $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$ for $Y \subset \mathcal{S}'_0$ and $m(Z) \cap \mathcal{S}_0 = \emptyset$, or $x \overset{\mathcal{T}}{\Rightarrow} x'_1 + X'$ for $\emptyset < m(x'_1) < m(x)$. If $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$ then we are done. If $x \overset{\mathcal{T}}{\Rightarrow} x'_1 + X'$ for $\emptyset < m(x'_1) < m(x)$, then every species $x' \in X'$ has $m(x') \leq m(x) - m(x'_1) < m(x)$, so by the induction hypothesis every $x' \in X' \cup \{x'_1\}$ is finished and therefore has $x' \overset{\mathcal{T}}{\Rightarrow} Y_{x'} + Z_{x'}$ as desired, so by combining all those $x \overset{\mathcal{T}}{\Rightarrow} \sum_{x'} Y_{x'} + \sum_{x'} Z_{x'}$, satisfying the modularity condition. This proves that if the algorithm says m is modular, then it is.

To complete the proof that the algorithm is correct, we show that if m is in fact modular, the algorithm will not say it is not. The algorithm terminates when a cycle passes with no change to the table, so we show that if m is modular but at the beginning of a cycle at least one species is not yet finished, then there is some fact not yet in the table that will be learned this cycle. Consider a modified implementation CRN where the reaction $x \xrightarrow{\mathcal{T}} x + Z_x$ for the current Z_x known to be producible at x is added for each x; if m is modular in the original CRN then it is modular in the new CRN, since reactions were only added. Then consider, for each $x \in \mathcal{S}'$, the first reaction on the path with the fewest non-$x \xrightarrow{\mathcal{T}} x + Z_x$ reactions by which $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$ to satisfy the modularity condition; the algorithm will consider these reactions (among others) as possible and update based on them this cycle. Now consider the path obtained by starting from an x not yet finished, at any given state $\{\!|x'|\!\}$ (ignoring null species), taking that first reaction, until either a $Y + Z$ satisfying the modularity condition is reached, or some $x'_1 + X'$ with $\emptyset < m(x'_1) < m(x)$ is reached, or a non-null species is repeated; for this purpose, the reactions $x' \xrightarrow{\mathcal{T}} x' + Z_{x'}$ do not count as repeating a species. (Given finitely many x' with $m(x') = m(x)$, one of those three must eventually happen.) If the path ends in $Y + Z$ or $x'_1 + X'$, then the last x' with $m(x') = m(x)$ along the path which is not yet finished (which may be x) will be marked as finished this cycle. If the path ends by repeating a species, say $x \overset{\mathcal{T}}{\Rightarrow} x'_0 + Z_1 \overset{\mathcal{T}}{\Rightarrow} x'_0 + Z_1 + Z_0$ with $m(x'_0) = x$, then if any species x'_1 in the loop $x'_0 \overset{\mathcal{T}}{\Rightarrow} Z_1 + Z_0$ is not known to reach some other species x'_2 in the loop, for each x'_2 the last such species will become known to reach x'_2 this cycle. If every species in the loop is known to reach every other species in the loop, and x'_0 is known to produce every species in Z_0, then replacing the $x'_0 \overset{\mathcal{T}}{\Rightarrow} x'_0 + Z_0$ loop with the (in our measure of path length, 0-length) reaction $x'_0 \xrightarrow{\mathcal{T}} x'_0 + Z_{x'_0}$ would create a shorter path by which $x \overset{\mathcal{T}}{\Rightarrow} Y + Z$; so there is some $z \in Z_0$ not known to be produced from x'_0. Somewhere in that loop is a reaction $x'_1 \xrightarrow{\mathcal{T}} x'_2 + z + Z_2$, and since every species in the loop is known to reach every other species in the loop (including itself), the last species in the loop before x'_1 which is not yet known to produce z (which may be x'_0) will be known to produce z this cycle. This covers all cases, and completes this part of the proof: if m is modular but the algorithm does not yet known that every species is finished, it will learn at least one new fact each cycle, thus never saying no.

To complete the proof, we show that the algorithm always terminates in polynomial time. Each cycle consists of for each implementation species, for each trivial reaction, checking whether that reaction is possible from that species plus null species, checking properties in the table for each of the produced species, and updating the table, a polynomial number of polynomial-time operations. Since at least one fact must be learned each cycle, the number of cycles is bounded by the number of facts: $n(n + z + 1)$, where n is the number of implementation species and z the number of null species, so the algorithm is guaranteed to terminate in polynomial time. Since the algorithm is guaranteed to terminate, that the algorithm never returns no when m is modular implies that it returns yes, which also completes the proof of correctness. □

To find modular bisimulation interpretations, we modify the reactionsearch algorithm such that after checking the permissive condition it also uses the above algorithm to check the modularity condition. For this to be correct we would need to prove that if a modular bisimulation exists then the algorithm will find it; thankfully this is true.

Theorem 2.5.7. *Given a formal and implementation CRN* $(\mathcal{S}, \mathcal{R})$ *and* $(\mathcal{S}', \mathcal{R}')$*, sets of common species* $\mathcal{S}_0 \subset \mathcal{S}$ *and* $\mathcal{S}'_0 \subset \mathcal{S}'$*, and a partial interpretation which specifies* $m(x)$ *for some set* $\mathcal{S}'' \subset \mathcal{S}'$ *of various* $x \in \mathcal{S}''$*, provided that either* $\mathcal{S}'_0 \subset \mathcal{S}''$ *or every formal species in* $\mathcal{S}_0$ *appears as a reactant in* $\mathcal{R}$*, whether a complete interpretation* $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ *exists that respects the given interpretation and is a modular bisimulation with respect to* $\mathcal{S}_0$ *and* $\mathcal{S}'_0$ *can be decided in polynomial space. In particular, if such an interpretation exists, then the modified* reactionsearch *algorithm will find one that is polynomial size in that of the two CRNs and the partial interpretation.*

Proof. That the modified reactionsearch algorithm outputs only polynomial-size bisimulations and runs in polynomial space is proven in Theorem 2.5.4, so when combined with the modularity checker it will output only polynomial-size modular bisimulations. If a complete modular bisimulation exists, then whatever partial interpretation of it specifies the interpretation of all nontrivial reactions will be considered by the modified reactionsearch algorithm, at which point it will call the trivial reaction solver; so far, the proof is the same as Theorem 2.5.4. Given that partial completion, by Lemma 2.5.5, if a completion of it exists (which it does) then one exists which is a minimal solution of the trivial reaction solver equations. That such an interpretation must be polynomial-size is again proven in Theorem 2.5.4, so it will be found and verified by the permissive and modularity checkers, and returned. □

Given a large formal CRN and implementation CRN along with a user-provided breakdown into modules with defined common species and a partial interpretation on the common implementation species, the modified reactionsearch algorithm may be applied sequentially (or in parallel) to each module; if all modules admit a valid modular interpretation, then a valid interpretation for the full system exists – specifically, the union of all the modules' interpretations, which will be consistent since they share the given interpretation on the common species. Furthermore, if the algorithm fails to find an interpretation for some module, then no valid modular interpretation exists (although it remains possible that a non-modular valid interpretation exists). Note that the choice of modules must be consistent with the requirements of Theorem 2.4.2, for example, every module contains the same set of common species and their intersection contains no other species.

We have shown that finding a modular bisimulation is not significantly harder than finding a bisimulation at all, and in fact finding a modular bisimulation for a large CRN broken into many modules is much easier than trying to find a bisimulation for the whole CRN with no information about its modularity. On a trivial level, any interpretation is modular with respect to common implementation species $\mathcal{S}_0' = \mathcal{S}'$ regardless of the common formal species *or* common formal species $\mathcal{S}_0 = \emptyset$ regardless of the common implementation species. Thus, Theorems 2.5.3, 2.5.5, and 2.5.6 apply, and checking a modular bisimulation is PSPACE-complete in general (but polynomial time in n^k if the largest number of reactants in a formal reaction is k), and finding one is PSPACE-complete in general and NP-complete when the number of reactants in a formal reaction is bounded by a constant. Whether this stays true for more "meaningful" cases of modularity is a more interesting question. For example, the property $\mathcal{S}_0 = \mathcal{S}$, $|\mathcal{S}_0'| = |\mathcal{S}_0|$ and satisfies the atomic condition—every formal species is common, and the common implementation species consist of exactly one species x_A with $m(x_A) = A$ for each formal species A—describes the typical non-history-domain systematic DSD implementation of CRNs, and neither of the CRNs in Theorems 2.5.3, 2.5.5, or 2.5.6 are modular with respect to any sets with that property. Whether there is another hardness proof for that sort of set of common species, or whether checking or finding a modular interpretation is in fact easier in (the worst case of) that subcase, is currently an open question.

2.6 Additional Features of CRN Bisimulation

Bisimulation in Transition Systems

We call this theory "CRN bisimulation" because it is a special case of the theory of weak bisimulation in concurrent systems, adapted to CRNs. In [52], this theory is defined in terms of a *labelled transition system*

$$(\Sigma, \mathcal{T}, \xrightarrow{t} : t \in \mathcal{T})$$

where Σ is a set of *states*, $\mathcal{T}$ a set of *labels*, and for each $t \in \mathcal{T}$ there is a relation $\xrightarrow{t} \subset \Sigma \times \Sigma$, specifying which states can transition to which other states by an *action* of type t. For example, a CRN $(\mathcal{S}, \mathcal{R})$ can be expressed as a labelled transition system; there are multiple ways to do this, but to give one particularly natural way, let $\Sigma = \mathbb{N}^{\mathcal{S}}$ be the set of all states in the usual sense of the CRN, $\mathcal{T} = \mathcal{R}$ so that the labels for transitions are the reactions, and for $r = R \to P \in \mathcal{R}$ the transition relation is $\xrightarrow{r} = \{(S, S - R + P) \mid S \in \mathbb{N}^{\mathcal{S}}, S \geq R\}$. This construction matches the semantics of CRNs as defined in Section 2.3, and is the basis of the connection between our theory of CRN bisimulation and weak bisimulation as defined in [52].

Aside from labelled transition systems, however, the paradigm used by Milner in [52] diverges from the paradigm we use when discussing CRNs. Milner discusses concurrent processes in terms of *agents* and *agent expressions* in a certain language, and defines a single labelled transition system where Σ is the set of *all*, infinitely many (in fact uncountably many) agent expressions, with $\mathcal{T}$ similarly infinite. Strong and weak bisimulation are used to define which agent expressions are in fact “the same agent” in terms of either what actions they can do (strong) or what *observable* (non-τ) actions they can do (weak). The two types of bisimulation are eventually used to define a notion of equality of agent expressions which roughly matches “same sequence of observable actions” while being preserved by each of the combinators in the language used to define an agent expression. For this purpose, the concept of one labelled transition system is particularly useful, and this one labelled transition system has single relations $\sim$, $\approx$, and $=$ defined as “the” strong bisimulation, weak bisimulation, and equality, respectively. In order to get there, however, Milner defines what it means for a relation to be “a” strong or weak bisimulation, then defines “the” strong or weak bisimulation as the largest such relation. For example,

Definition 2.6.1 (Definition 5.5 in [52]). A relation $\leftrightarrow \subset \Sigma \times \Sigma$ is a (weak) bisimulation if $P \leftrightarrow Q$ implies, for all $\alpha \in \mathcal{T}$,

(i) Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \overset{\alpha}{\Rightarrow} Q'$ and $P' \leftrightarrow Q'$

(ii) Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \overset{\alpha}{\Rightarrow} P'$ and $P' \leftrightarrow Q'$

where, as in Section 2.4, $P \overset{\tau}{\Rightarrow} Q \iff P \xrightarrow{\tau}^{*} Q$ and $P \overset{\alpha}{\Rightarrow} Q \iff P \xrightarrow{\tau}^{*} P'' \xrightarrow{\alpha} Q'' \xrightarrow{\tau}^{*} Q$ for $\alpha \neq \tau$. (Our notation is slightly different from Milner's: we use $\overset{\alpha}{\Rightarrow}$ to mean the same thing as Milner's $\overset{\hat{\alpha}}{\Rightarrow}$.) Note the similarity between Definitions 2.6.1 and 2.4.2(III).

In contrast to Milner in [52] comparing two agents (states) of the same labelled transition system, we want to compare two CRNs which we think of as separate (labelled transition) systems. This itself is not a significant difference: given two systems $(\Sigma_1, \mathcal{T}, \xrightarrow{t}_1 \colon t \in \mathcal{T})$ and $(\Sigma_2, \mathcal{T}, \xrightarrow{t}_2 \colon t \in \mathcal{T})$ and a relation $\leftrightarrow \subset \Sigma_1 \times \Sigma_2$ we can consider the system $(\Sigma = \Sigma_1 \cup \Sigma_2, \mathcal{T}, \xrightarrow{t} = \xrightarrow{t}_1 \cup \xrightarrow{t}_2 \colon t \in \mathcal{T})$ with $\leftrightarrow \subset \Sigma_1 \cup \Sigma_2 \subset \Sigma \times \Sigma$, fitting Milner's paradigm with no significant changes. More importantly, our concept of CRN equivalence wants to consider an asymmetric pair of CRNs: one, $(\mathcal{S}, \mathcal{R})$, is the "formal" CRN where $\mathcal{R}$ is the set of "meaningful actions", and another, $(\mathcal{S}', \mathcal{R}')$, is meant to be an implementation of the formal CRN. This means that some natural conditions we want our definition of correctness to have are that every state of the implementation CRN corresponds to one and only one state of the formal CRN; that every state of the formal CRN has *at least* one state of the implementation CRN that implements it; and since we're working with CRNs, which are fundamentally linear, that the sum of any number of implementation states corresponds to the sum of their corresponding formal states. (This linearity condition is also why the undecidability result from [39] doesn't apply to our CRN bisimulation.) It turns out that those conditions on a relation $\leftrightarrow \subset \mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}'}$ are true if and only if $\leftrightarrow$ corresponds to some interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ as defined in Definition 2.4.1:

Lemma 2.6.1. *Let $\leftrightarrow \subset \mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}'}$ be a relation between formal states and implementation states. If for every implementation state S' there is exactly one formal state S such that $S \leftrightarrow S'$ (function) and for every pair of pairs $S_1 \leftrightarrow S_1'$ and $S_2 \leftrightarrow S_2'$ we have $S_1 + S_2 \leftrightarrow S_1' + S_2'$ (linearity), then there is some interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ which, when extended to implementation states $m : \mathbb{N}^{\mathcal{S}'} \to \mathbb{N}^{\mathcal{S}}$, induces that relation: $S \leftrightarrow S' \iff S = m(S')$. Furthermore, for every S there is some S' such that $S \leftrightarrow S'$ (surjectivity) iff m satisfies the atomic condition.*

Proof. Given that the relation $\leftrightarrow$ is a linear function from $\mathbb{N}^{\mathcal{S}'}$ to $\mathbb{N}^{\mathcal{S}}$, we define the interpretation to be $m(x) = S_x$ where S_x is the unique formal state such that $S_x \leftrightarrow \{\!|x|\!\}$. Now, any implementation state S' is some sum of implementation species, $S' = \sum_{x \in \mathcal{S}'} \alpha_x x$, and because we define the interpretation of a state as the sum of interpretations of species, $m(S') = \sum_{x \in \mathcal{S}'} \alpha_x m(x)$. Then by the linearity assumption on $\leftrightarrow$, $m(S') \leftrightarrow S'$. Thus, if $S = m(S')$, then $S \leftrightarrow S'$. Conversely, if $S \leftrightarrow S'$, then $S = m(S')$ because $\leftrightarrow$ is a function.

If we further assume that $\leftrightarrow$ is surjective, then in particular for each formal species A, there must be some S' such that $\{\!|A|\!\} \leftrightarrow S'$, i.e. $m(S') = \{\!|A|\!\}$. Since $m(S')$ is the sum of interpretations of species in S' and an implementation species cannot interpret to fractional or negative formal species, there must be some species $x_A \in S'$ with $m(x_A) = \{\!|A|\!\}$ (and any other species in S' interpret to $\emptyset$). Thus the atomic condition is satisfied. Conversely, if the atomic condition is satisfied, then consider an arbitrary formal state $S = \sum_{A \in \mathcal{S}} \alpha_A A$. Using linearity, let $S' = \sum_{A \in \mathcal{S}} \alpha_A x_A$, so $m(S') = S$, and thus $\leftrightarrow$ must be surjective. □

Since we said that $\mathcal{R}$ should be the set of "meaningful actions", that means our transition systems $(\Sigma_i, \mathcal{T}, \xrightarrow{t}_i)$ should have the same set of labels, and at first glance that set of labels should be $\mathcal{T} = \mathcal{R} \cup \{\tau\}$. In the formal CRN, this is easy: the interpretation of a CRN as a transition system that we previously described has $\mathcal{T} = \mathcal{R}$, which simply means no τ transitions appear. In the implementation CRN, we need to find a correspondence between implementation reactions and formal reactions (or τ), but this is already what the interpretation does: as described in Definition 2.4.1, any interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ induces a map $m : \mathcal{R}' \to (\mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}) \cup \{\tau\}$. (We previously referred to members of $\mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}$ not necessarily in $\mathcal{R}$ as "reactions in the language of the formal CRN".) Since an implementation reaction might be interpreted as a reaction in the language of the formal CRN which is "invalid" i.e. not in $\mathcal{R}$, we take $\mathcal{T} = (\mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}) \cup \{\tau\}$, and when converting the implementation CRN to a transition system we say for $r \in \mathcal{T}$ that $S' \xrightarrow{r} T'$ if $S' \xrightarrow{r'} T'$ and $m(r') = r$ for some $r' \in \mathcal{R}'$. (This means that, formally, which transition systems we are comparing depends on the relation we find between them, a bit of apparent circularity which leads to various differences between CRN bisimulation and the classic definition.) Given this, an interpretation is a CRN bisimulation (satisfies Definition 2.4.2(III)) if and only if it satisfies the Atomic Condition (Definition 2.4.2(II.i)) and the relation on states it induces is a weak bisimulation (Definition 2.6.1). Therefore, a valid interpretation m can be equivalently described as a surjective linear weak bisimulation.

We would like to compare some features of our concept of CRN bisimulation to bisimulation in transition systems as in [52]. To avoid ambiguity, for this discussion we use the phrase "bisimulation relation" to mean a relation between states $\leftrightarrow \subset S \times S$ that satisfies Definition 2.6.1, and "CRN bisimulation" or "bisimulation interpretation" to mean an interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ that satisfies Definition 2.4.2 and therefore induces a bisimulation relation.

The most important difference between a bisimulation relation and a CRN bisimulation is that, as we said earlier, the transition system induced by the implementation CRN is not fully defined until an interpretation is given. For example, if $x_1 \to x_2 \in \mathcal{R}'$, $m(x_1) = A$ and $m(x_2) = B$ then the transition $\{\!|2x_1|\!\} \to \{\!|x_1, x_2|\!\}$ has label $A \to B$, but if $m(x_1) = m(x_2) = A$ then the same transition has label τ. So for example, the fact (Proposition 5.1(4) in [52]) that $\bigcup_{i \in I} \leftrightarrow_i$ is a bisimulation relation if each $\leftrightarrow_i$ is a bisimulation relation has no obvious analog for CRN bisimulations, since there is no obvious way to even define the union of two relations defined on different and "contradictory" transition systems. We don't try.

Milner discusses the relation $\approx = \bigcup\{\leftrightarrow \mid \leftrightarrow \text{ is a bisimulation relation}\}$ [52]. In any given transition system, such a relation exists, is the largest bisimulation relation (which follows from the previous statement about unions), and is an equivalence relation. In the context of comparing agent expressions, this is a useful way of saying, for example, that the result of applying a sequence of transitions to a complex agent expression is another complex agent expression. In the context of a formal CRN $(\mathcal{S}, \mathcal{R})$ and its induced transition system $(\mathbb{N}^{\mathcal{S}}, \mathcal{R} \cup \{\tau\}, \xrightarrow{r})$, the relation $\approx$ exists but is not very useful. In fact the relation is the trivial $S \approx T \iff S = T$ relation except in some particularly degenerate CRNs; for example, if $\mathcal{S} = \{A\}$ and $\mathcal{R} = \{\emptyset \to A\}$ then $S \approx T$ for all S, T. In the context of CRN bisimulation, a formal CRN and implementation CRN where the actions are reactions in the language of the formal CRN or τ, as previously discussed we haven't finished defining the transition system, so $\approx$ is not yet defined without an interpretation. Given an interpretation m, we have a transition system so $\approx$ exists, but it is mostly restricted by m. If m is a CRN bisimulation, then adopting the convention that $m(S) = S$ for formal states $S \in \mathbb{N}^{\mathcal{S}}$, $S \approx T \iff m(S) \approx m(T)$ for any (formal or implementation) states S, T, where the right side can use the definition of $\approx$ on the formal CRN. That is, "the bisimulation" is just m together with any degeneracy in the formal CRN.

Handling Spurious Catalysts

The definition of CRN bisimulation as stated previously has difficulty handling overly detailed enumerations of DNA strand displacement circuits, but a simple extension of bisimulation fixes that problem. As an example, consider the implementation of the reaction $A + B \to C + D$ according to the variant of the scheme by Soloveichik et al. [66] discussed in Section 2.4. Figure 2.15 shows a spurious reaction possible in that scheme: a toehold on the "trigger strand" (what would be t_{CD} if released) in the complex i_A binds to the exposed complementary toehold in another copy of i_A. This spurious binding has no meaningful effect on the DSD system's function: no strand displacement reactions are possible given that binding that should not be possible, and since the binding is reversible it can fall off before any reaction that it would otherwise interfere with. In particular, an analog of the $i_A + x_B \to t_{CD} + w_1$ reaction can still happen in this complex, producing a t_{CD} strand with a spurious binding to an i_A complex (and a normal w_1 waste complex). However, when analyzing the system with bisimulation, we need to interpret each implementation species, including this complex and the result of the reaction. In order for the binding and unbinding reactions to be trivial, we must interpret the $i_A : i_A$ complex as $2A$ and the $t_{CD} : i_A$ complex as $C + D + A$, and they must be trivial in order to satisfy the delimiting condition. Then the reaction $i_A : i_A + x_B \to t_{CD} : i_A + w_1$ is interpreted as $2A + B \to A + C + D$. This is neither trivial nor is it a formal reaction, so by bisimulation as so far defined, the delimiting condition is violated. However, it is "clearly" the reaction $A + B \to C + D$ with a "spurious catalyst" A on the side, and we would like a definition of bisimulation that can confirm this.

Recall that in Definition 2.4.1 we defined three related but distinct interpretations: $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ an interpretation of implementation *species*; $m : \mathbb{N}^{\mathcal{S}'} \to \mathbb{N}^{\mathcal{S}}$ an interpretation of implementation *states*; and $m : \mathbb{N}^{\mathcal{S}'} \times \mathbb{N}^{\mathcal{S}'} \to (\mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}) \cup \{\tau\}$ an interpretation of implementation *reactions*. At the time, we said that the interpretation of species $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ was arbitrary, but the other two were defined unambiguously in terms of that m. While we still want to keep the interpretation of *states* as the sum of the interpretations of species, we can

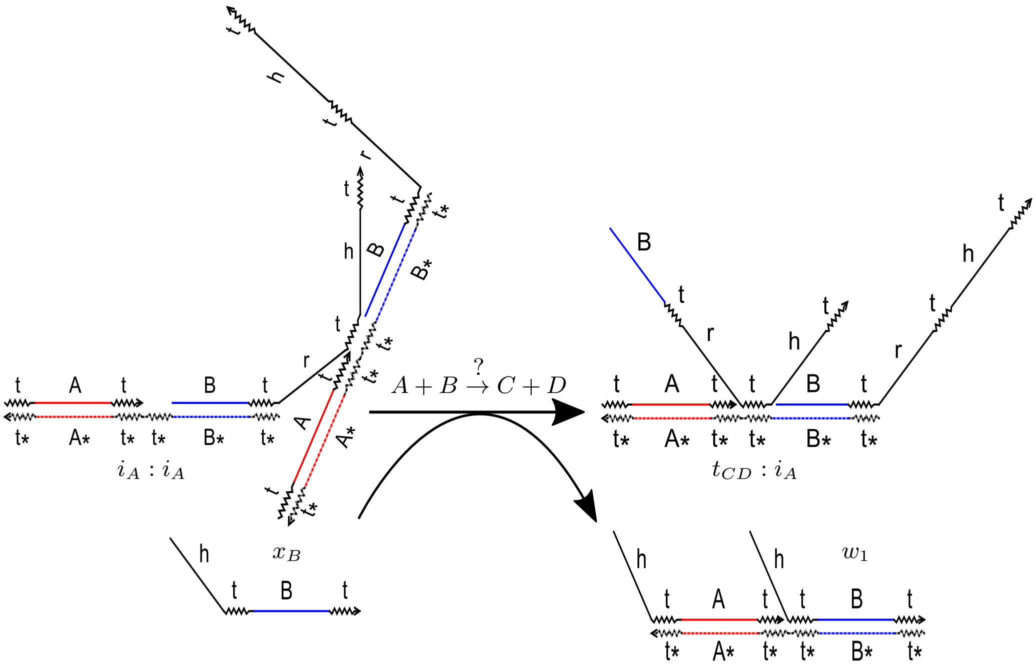


Figure 2.15: An example spurious reaction in a variant of the translation scheme by Soloveichik et al. [66].

loosen the definition of the interpretation of a reaction to allow reactions like $i_A : i_A + x_B \rightarrow t_{CD} : i_A + w_1$ to be interpreted as $A + B \rightarrow C + D$ as intended.

Definition 2.6.2. Let $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ be a formal and implementation CRN with $m : \mathcal{S}' \rightarrow \mathbb{N}^{\mathcal{S}}$ an interpretation of implementation species, which is extended to implementation states as in Definition 2.4.1. An interpretation of reactions $m_\rho : \mathbb{N}^{\mathcal{S}'} \times \mathbb{N}^{\mathcal{S}'} \rightarrow (\mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}) \cup \{\tau\}$ is *consistent with* m if:

(i) If $R' \neq P'$ but $m(R') = m(P')$ then $m_\rho(R' \rightarrow P') = \tau$, and

(ii) If $m(R') \neq m(P')$ and $m_\rho(R' \rightarrow P') = R \rightarrow P$ then there is some $C \in \mathbb{N}^{\mathcal{S}}$ such that $m(R') = R + C$ and $m(P') = P + C$.

Adapting the definition of bisimulation to this new concept of interpretation is straightforward.

Definition 2.6.3. Let $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ be a formal and implementation CRN. Let $m : \mathcal{S}' \rightarrow \mathbb{N}^{\mathcal{S}}$ be an interpretation of implementation species and m_ρ an interpretation of reactions consistent with m. The pair (m, m_ρ) is a CRN bisimulation if m satisfies the atomic condition, m_ρ satisfies the delimiting condition, and the combination satisfies the permissive condition where m is applied to states and m_ρ to reactions.

Most of the important properties of the interpretation of reactions remain true for any m_ρ consistent with m. For example, if $S' \xrightarrow{r'} T'$ and $m_\rho(r') = r = R \to P$ then $m(T') = m(S') - R + P$, as we used in proving Theorem 2.4.1. In fact, once trajectory equivalence and weak bisimulation are redefined to use m_ρ for the interpretation of any reaction, they remain equivalent to the three conditions by the same logic used in that theorem. Less trivially, it turns out that the algorithms for checking or finding a bisimulation discussed in Section 2.5 can be modified to work with the new concept of CRN bisimulation. (For complexity purposes, we assume m_ρ is written by writing the index of $m_\rho(r')$ for each $r' \in \mathcal{R}'$, giving it size $|m_\rho| \leq |\mathcal{R}'| \log |\mathcal{R}| \leq n \log n$.)

Theorem 2.6.1. *Given a formal and implementation CRN $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ with interpretation (m, m_ρ) where m_ρ is consistent with m, the problem of checking whether (m, m_ρ) is a CRN bisimulation is PSPACE-complete in general, and can be checked in polynomial space by the* loopsearch *algorithm as previously described. The* graphsearch *algorithm as previously described also correctly checks whether (m, m_ρ) is a CRN bisimulation, and when the number of reactants in any formal reaction in $\mathcal{R}$ is bounded by some k, the* graphsearch *algorithm runs in* $\mathrm{poly}(n^k)$ *time and space.*

Proof. Both algorithms do not depend on the fact that $m(R' \to P') = m(R') \to m(P')$ or τ, but only depend on being able to find $m(r')$ given r', which is still easy given m_ρ. Similarly, assumptions made by the algorithm such as if $S' \xrightarrow{\tau} T'$ then $m(S') = m(T')$ still hold. Thus, the previous proof of correctness and complexity of the algorithms holds. Similarly, in the completeness proof in Theorem 2.5.3, the given interpretation with $m_\rho(R' \to P') = m(R') \to m(P')$ when $m(R') \neq m(P')$ is consistent with m and is the same as the interpretation in that theorem, and is still correct if and only if the space-bounded Turing machine accepts, so the problem is still PSPACE-complete. □

When we try to find an interpretation, we modify the reactionsearch algorithm as follows. The algorithm takes as input $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ as the previous version, and also takes zero or more *interpretation constraints*. An interpretation constraint is a statement of one of the forms $m(x) = S_x$, $m(x) \geq S_x$, or $m_\rho(r') = r$, where x is an implementation species, S_x a multiset of formal species, r' an implementation reaction, and r a formal reaction. We assume any r' has zero or one $m_\rho(r') = r$ constraints and any x has either exactly one $m(x) = S_x$ constraint or zero or more $m(x) \geq S_x$ constraints but not both; anything else would be redundant or contradictory, and it would be easy to tell which it is. Where the algorithm previously would consider the possibility that $m(r') = r$ and enumerate all possible partial interpretations of uninterpreted species in r', then call itself recursively, the new algorithm enumerates all possible *minimal* (in the usual sense of having no strict subset be valid; e.g. if $m(x_1) \geq A$, $m(x_2) \geq B$ is valid then $m(x_1) \geq 2A$, $m(x_2) \geq B$ is not minimal) partial interpretations of all implementation species x appearing in r' which do not have an $m(x) = S_x$ constraint, and recursively calls itself with the reaction constraint $m_\rho(r') = r$ and the enumerated partial interpretation encoded as $m(x) \geq S_x$ constraints, with an exception: if every species x on one side of r' has an $m(x) = S_x$ constraint, then the enumerated interpretation of the other side is passed as $m(x) = S_x$ constraints.

The algorithm runs the trivial reaction solver when every r' with no $m_\rho(r') = r$ constraint can be trivial. When solving the atomic condition before running the trivial reaction solver, the new algorithm assigns an $m(x) = A$ for each A such that there is no $m(x) = A$ restriction already present, and chooses from all x such that there is no $m(x) = S_x \neq A$ restriction and $m(x) = A$ is not contradicted by any $m(x) \geq S_x$ restriction. When running the trivial reaction solver, the algorithm first assigns to each x with no $m(x) = S_x$ restriction a "base interpretation" $m_0(x) = \bigvee_{m(x) \geq S_x} S_x$, then sets up and solves equations in terms of an "additional interpretation" $m_+(x)$ such that $m(x) = m_0(x) + m_+(x)$. (As in the previous algorithm, it solves separately for each $m_+(x; A)$.) The algorithm sets up an equation for *every* implementation reaction r', even those with an $m_\rho(r') = r \neq \tau$ constraint; if $r' = R' \to P'$ has an $m_\rho(r') = r = R \to P$, then the equations are of the form $m(R') - R - m(P') + P = 0$, expanded in terms of each $m_+(x; A)$ after subtracting m_0 and replacing any species x with an $m(x) = S_x$ constraint. As before, the algorithm searches for only one minimal solution, then tests the permissive condition, since as before if there is any solution that satisfies the permissive condition then there is a unique (for every formal species that appears as a reactant in any reaction) minimal solution and it satisfies the permissive condition.

Theorem 2.6.2. *Given a formal and implementation CRN $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ with zero or more conditions of the form $m(x) = S_x$, $m(x) \geq S_x$, or $m_\rho(r') = r$, the modified* reactionsearch *algorithm as described above correctly finds an interpretation or asserts that none exists. The algorithm runs in polynomial space, and if a correct interpretation exists then the algorithm outputs one that is polynomial size in its inputs. Deciding whether an interpretation from a given implementation CRN to a given formal CRN is PSPACE-complete in the general case, and is NP-complete when the number of reactants in any reaction in $\mathcal{R}$ is bounded by a constant $k \geq 1$.*

Proof. Lemma 2.5.4 still applies to the new trivial reaction solver; the proof as stated applies exactly to the new definition of CRN bisimulation with the exception that the statement $m(Y')(A) = R(A) > m(R')(A)$ needs to become $m(Y')(A) \geq R(A) > m(R')(A)$, which does not affect the remainder of the proof. The proof that the new algorithm is correct then follows the same lines as the proof that the old algorithm is correct in Theorem 2.5.4; if a correct completion of the interpretation exists, then one of the branches of the algorithm will find it.

To prove that finding an interpretation is PSPACE-complete, we use the same reduction from linear bounded Turing machine acceptance to formal and implementation CRN as in Theorem 2.5.5. Even under the expanded definition of correct interpretation, the interpretation $m(0_i) = m(1_i) = A_i$, $m(h) = H$, $m(q_i^j) = Q$ for $j \neq m-1$ and $m(q_i^{m-1}) = Q + \sum_{k<i} A_k$, with $m_\rho(q_n^{m-1} + 0_n \to h) = Q + A_1 + \cdots + A_n \to H$ and $m_\rho(r') = \tau$ otherwise, is the only possibly correct interpretation up to a permutation of formal species Q and the A_i's, and it is correct if and only if the given Turing machine accepts the given input string x. The proof, however, requires some different steps to rule out possibilities that are opened up by a less restrictive definition of interpretation. First, we observe that $m_\rho(q_i^m + x_i \to q_{i-1}^m + x_i) = \tau$, since combining those reactions with $q_i^m \to q_n^m$ gives a loop from $q_n^m + x_1 + \cdots + x_n$ to itself; this is impossible in the formal CRN if any reaction fires, so all reactions involved must be trivial (or violate the delimiting condition). This means that $m(q_i^m) = m(q_n^m)$, and applying the same to $q_1^m + x_1 \to q_1^0 + x_1$ and $q_1^0 \to q_n^m$ we get that $m(q_1^0) = m(q_n^m)$ also. Now, each $q_i^m + x_{-i} \to q_{i-1}^m + x_i$

and $q_1^m + x_{-1} \to q_1^0 + x_1$ reaction shows that for each i either $m(0_i) = m(1_i)$ or $m(x_i) = H$ and $m(x_{-i}) = Q + A_1 + \cdots + A_n$ (the second case was impossible in the old definition when $m(q_i^m) \neq \emptyset$), and similarly from $q_i^j \to q_n^m$ either $m(q_i^j) = Q + A_1 + \cdots + A_n$ or $m(q_i^j) = m(q_n^m)$. Since no formal reaction is reversible, we have $m_\rho(q_1^{m-1} + 1_1 \rightleftharpoons q_2^{m-1}) = m_\rho(q_{i-1}^{m-1} + 0_{i-1} \rightleftharpoons q_i^{m-1}) = \tau$, so $m(q_i^{m-1}) = m(q_1^{m-1}) + m(1_i) + \sum_{2 \leq k < i} m(0_k)$ for $i \geq 2$. This is important because, as in the previous proof, we have $n+2$ formal species and by the atomic condition, each one must have at least one implementation species interpreted as one copy of it and nothing else. Although we have yet to prove that $m(0_i) = m(1_i)$, the above does prove that the two cannot (for the same i) satisfy the atomic condition for two *different* formal species; similarly, no q_i^j can satisfy the atomic condition for a different formal species than q_n^m. This leaves $n+2$ groups of implementation species—q_n^m, x_i for $1 \leq i \leq n$, and h—to $n+2$ formal species, so each of the mentioned implementation species must be interpreted as exactly one copy of a different formal species. Then $m(0_n)$ is either a single formal species not equal to $m(h)$, or is $Q + A_1 + \cdots + A_n$, so $m_\rho(q_n^{m-1} + 0_n \to h) \neq \tau$, so $m_\rho(q_n^{m-1} + 0_n \to h) = Q + A_1 + \cdots + A_n \to H$ and $m(h) = H$. This rules out the possibilities that any of the x_i's or q_n^m are interpreted as H, leaving the only possibility that they are interpreted as some assignment of Q and the A_i's, with $m(0_i) = m(1_i)$, $m(q_i^j) = m(q_n^m)$ for $j \neq m-1$, and $m(q_i^{m-1}) = m(q_n^m) + \sum_{k<i} m(0_k)$. This interpretation satisfies the atomic and delimiting conditions, and satisfies the permissive condition if and only if the given Turing machine accepts x, thus deciding whether a correct interpretation exists is still PSPACE-complete.

The proof of Theorem 2.5.6 applies to the new definition of interpretation without modification, proving that whether a correct interpretation exists is NP-complete when the number of reactants in a formal reaction is bounded by a constant $k \geq 1$. □

2.7 Discussion

Comparing Chemical Reaction Networks on different levels of abstraction is an important tool for systematic programming with CRNs. We showed how to adapt the concept of bisimulation to check whether one CRN is a correct implementation of another. We showed that bisimulation can be used to prove the correctness of some existing CRN implementations, and to identify subtle but real problems with others. We discussed transitivity and modularity, which can be used to simplify a bisimulation proof. We presented different algorithms to check bisimulation which are adapted to different cases. We showed that the condition can be checked in polynomial time with favorable assumptions, is NP-complete with less favorable assumptions, and is PSPACE-complete in the general case.

In the beginning, we mentioned a DNA implementation of the approximate majority CRN [2] that was experimentally demonstrated by Chen et al. [19]. We might consider applying our bisimulation checker to this implementation. The implementation as presented in [19] would be incorrect according to bisimulation, for the same reason the example in Figure 2.5 from Qian et al. [57] fails: outputs of an irreversible reaction are released before an irreversible step is taken, leading to a small probability of such a reaction reversing itself after the products have reacted downstream. Despite this, Chen et al.'s *in vitro* experimental demonstration showed no such problems. While there are a number of explanations for this observation, including that the formal approximate majority CRN is particularly resistant to error [2], it nonetheless raises the question of how serious are the potential errors that may occur in CRN implementations that are not correct according to bisimulation? The answer will depend on the specific formal CRN of interest, as well as the conditions under which it is run. For example, behavior that may be problematic with non-negligible probability in low molecular counts, may have negligible effect in high molecular counts typical of *in vitro* experiments.

Another observation we have is that for typical engineered CRN implementations, at least for DNA strand displacement implementations, either there is a problem in the implementation of one formal reaction; or there is a problem with crosstalk between formal reactions; or there is no problem, and correctness can be proven by the modularity condition. In the case of crosstalk, as we mentioned in Section 2.4, that problem needs to be detected by the reaction enumerator, and is beyond the scope of our bisimulation theory. In the implementation by Chen et al. [19], for example, there are three formal reactions, but the (technically) incorrect behavior can be detected by considering only one of them. In the implementation of the rock-paper-scissors oscillator by Srinivas et al. [67], they use a systematic translation method slightly modified from Soloveichik et al. [66]. After confirming that their method applied to one reaction is correct, using Corollary 2.4.3 we can prove that such a method applied to *any* combination of reactions will be correct according to bisimulation.

The theory and algorithms discussed in this paper have been incorporated by Badelt et al. into the Nuskell compiler, a software package that automatically translates a CRN into a DNA strand displacement circuit and verifies that the result is correct [4]. Nuskell currently contains the loopsearch and graphsearch algorithms for checking the permissive condition as well as an exhaustive search algorithm for the same, the reactionsearch algorithm for finding an interpretation, and the algorithms to check the modularity condition and find a modular interpretation when given a decomposition into modules of an implementation CRN. Badelt et al. use bisimulation to verify a number of translation schemes applied to the rock-paper-scissors oscillator [25, 44, 67], showing that bisimulation algorithms can be used to verify CRN implementations used in practice.

Algorithms such as the graphsearch algorithm and loopsearch algorithm scale better with the number of meaningful species than the number of null species, while engineered CRN implementations generally do not use loops that produce null species. Thus those algorithms will be faster than their worst-case limits in practical cases. For example, the graphsearch algorithm takes at most $(2zn^k+1)n^k = O(n^{2k+1})$ cycles in theory, where n is the number of implementation species, k the largest number of reactants in a formal reaction, and z the number of implementation species with empty interpretation. When there are no null species (or when none can be produced in a loop, as in schemes such as [66]), this becomes at most n^k cycles.

In CRN bisimulation, we require that *every* implementation species has an interpretation as a (possibly empty) multiset of formal species. In contrast, verification methods such as pathway decomposition [64] or serializability [46] both assume that each formal species is represented by one implementation species, while other implementation species are classified into fuels, wastes, and intermediates. Because of this, pathway decomposition and serializability compare formal reactions to implementation pathways which begin and end with (representations of) formal species, while in bisimulation an individual implementation reaction can be interpreted and compared to the formal CRN. An additional consequence, for pathway decomposition, is that correctness guarantees do not apply to implementation states that cannot be reached from initial states representing formal species, whereas bisimulation is more robust in that correctness is asserted in those cases as well. Furthermore, even in the permissive condition, bisimulation requires that there *exist* an implementation pathway which implements a given formal reaction, while pathway decomposition and serializability both require that *all* implementation pathways have properties which may be nontrivial to check. This locality is what allows us to prove the complexity results given, which we suspect are significantly lower complexity than methods that depend on implementation pathways.

However, the use of interpretations instead of pathways means that in some cases CRN bisimulation and pathway decomposition differ on which implementations they consider correct. Bisimulation can easily be adapted to situations where there is no clear single "canonical representation" of a given formal species, while pathway decomposition has difficulty. For example, the implementation in [57] of the *reversible* formal reaction $A+B \rightleftharpoons C+D$ by reversible implementation reactions $\{x_A \rightleftharpoons i_A, i_A+x_B \rightleftharpoons i_{CD}, i_{CD} \rightleftharpoons x_C+i_D, i_D \rightleftharpoons x_D\}$. Bisimulation considers this correct with the obvious interpretation, while pathway decomposition considers the ability to release x_C then reverse without releasing x_D to be an error. On the other hand, bisimulation has trouble handling implementation species with no well-defined interpretation. Shin et al. describe a "delayed choice" phenomenon where an implementation CRN commits to implementing one of two formal reactions before deciding which one, producing an intermediate that cannot be correctly interpreted as either of the reaction's products or their reactants; such implementations are generally considered incorrect according to bisimulation but pathway decomposition often considers them correct [64]. They then propose a hybrid notion of correctness where an implementation CRN is considered correct if it is a correct implementation according to pathway decomposition of some intermediate CRN, and the intermediate CRN is a correct implementation of the formal CRN according to bisimulation [64]. This notion considers correct any implementation that is correct according to either pathway decomposition or bisimulation, plus some others.

One area this theory overlooks is the rates of reactions and the probabilities of reaching certain states. For example, in [66] Soloveichik et al. argue that the concentration of each intermediate is proportional to the product of that of the formal species which we would call its interpretation, and thus the reaction rates are approximately correct. Whether this can be generalized, and whether bisimulation can help this generalization, is an important open question.

Acknowledgments

The authors would like to thank Chris Thachuk, Damien Woods, Dave Doty, and Seung Woo Shin for helpful discussions. We would also like to thank the anonymous reviewers for many helpful suggestions. RFJ and EW were supported by NSF grants 1317694, 1213127, and 0832824. RFJ was supported by Caltech's Summer Undergraduate Research Fellowship program and an NSF graduate fellowship. QD thanks Steve Skiena for his kindness and flexibility.

Chapter 3

VERIFYING POLYMER REACTION NETWORKS USING BISIMULATION

3.1 Perspective

I have mentioned that Chemical Reaction Networks (CRNs) are a useful programming language to describe molecules interacting in solution, and that they can be compiled into DNA strand displacement (DSD) systems which can be implemented physically with DNA strands. In the previous chapter I discussed a method called CRN bisimulation to verify the equivalence of two CRNs, which can be used to confirm that the DSD compilation of a CRN is in fact a correct compilation. However, while “molecules interacting in solution” describes many of the possible behaviors of molecules, it doesn’t describe all of them. In this chapter I discuss one way to go beyond molecules interacting in solution, namely unbounded linear polymers. In particular, I discuss how to describe such systems with a linear Polymer Reaction Network (linear PRN or just PRN) model, how to extend CRN bisimulation to the PRN model as PRN bisimulation, and what this means for potential DSD or other physical implementations of polymer systems.

What are polymers and why are they important? The CRN model assumes a bunch of molecules moving freely in solution, where any molecule can come into contact with any other; this is often referred to as a “well-mixed” system. “Systems with geometry” refers to the broad class of systems where that’s not true. Polymer systems are a specific type of geometry, where individual “monomers” bond to each other, and the structure of the bonds makes monomers more or less likely to interact. DNA, RNA, and proteins are classic examples of polymers, where a small number of monomer (nucleotide or amino acid) types form a huge variety of chains. Importantly, the behavior of biological polymers can’t be understood without the polymer structure—the sequence AGGAGG, for example, has different behavior than other arrangements of 4 G’s and 2 A’s.

Though not commonly thought of this way, the common way of writing numbers is a type of polymer: the number 137 in decimal, or 10001001 in binary,

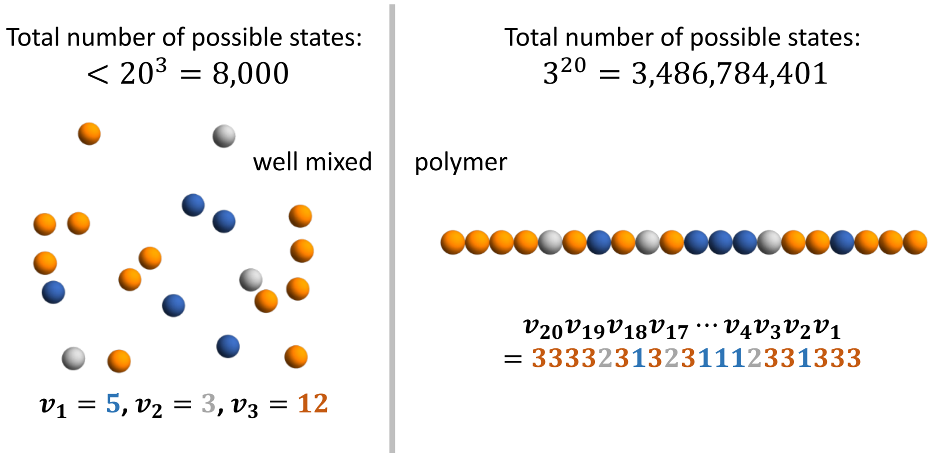

Total number of possible states:
$< 20^3 = 8{,}000$
well mixed
$v_1 = 5, v_2 = 3, v_3 = 12$
Total number of possible states:
$3^{20} = 3{,}486{,}784{,}401$
polymer
$v_{20}v_{19}v_{18}v_{17}\cdots v_4v_3v_2v_1$
$= 33332313231112331333$

Figure 3.1: Scaling of number of states in well-mixed versus polymer systems. Art by Lulu Qian.

is a linear string of monomers such as 0, 1, 3, and 7, and 137 is definitely not the same as 713, nor is 10001001 the same as 00011010 (26 in decimal). Not only that, but common operations such as addition can be thought of as "interactions between adjacent monomers": you add the digits in the same place, then carry over any extra value in the place to their left. The same applies to many computer programs, especially in abstract models of computing such as Turing machines. The fundamental feature of a Turing machine, for example, is a tape made up of squares with different symbols written on them—a linear polymer—with a head moving between adjacent squares and altering the square at its current position. Polymers are fundamental to biology, mathematics, and computation, and are so because they are in a fundamental way more powerful than well-mixed CRNs. There are a number of interesting results illustrating this difference in power [1, 16, 42, 47, 65], but it can be seen starkly in one calculation, illustrated in Figure 3.1. In a well-mixed system where you have 20 total molecules each of which can be one of 3 types, all that matters is the count of each of three species, and the number of possible states is less than $20^3 = 8000$. If instead you have a linear polymer made of 20 monomers, each of which can be one of 3 types, where the same count of species in a different order is a different state, then the number of possible states is $3^{20} = 3,486,784,401$.

This chapter focuses not just on PRNs but on PRN bisimulation, a verification method that extends the CRN bisimulation method discussed in Chapter 2.

If one asks why we would want verification for our polymer systems, then many of the answers will be the same as for CRN bisimulation: systems can have errors, systems large enough to do meaningful tasks can be harder to check and have more subtle errors, and formal verification also serves as a foundation for analysis of systems and guided design. All of this is discussed in the Perspective for, and elsewhere in, Chapter 2. In that case, what's different about PRN bisimulation? In a CRN, there are a finite number of species but an infinite number of possible states; thus, CRN bisimulation took advantage of the structure of CRNs and used an interpretation on the finite set of species that automatically extended to the infinite set of states. In a PRN, the corresponding concept to CRN species are the polymers, of which there are an infinite number made up of a finite number of monomers. So in PRN bisimulation we again take advantage of the structure of PRNs and define an interpretation of each monomer, which can extend to an interpretation of polymers (species) and thus to an interpretation of states.

Our work in this paper is to define the (linear) Polymer Reaction Network model, extend CRN bisimulation to PRN bisimulation with an interpretation suited to the structure of PRNs, and work out the implications of this extension. Like CRN bisimulation, algorithmic checking of an implementation PRN or a DSD implementation is desirable; unlike CRN bisimulation, checking a PRN is much more difficult. We show one example where, using knowledge of how a previously proposed DSD system [57] was meant to implement its specification, we can prove that it is correct according to PRN bisimulation. Currently many steps of that needed our knowledge and input; and even worse, some parts of the task are provably harder or impossible in the most general case. However, we suspect it is possible to find an algorithm that can solve both of those problems in the cases we care most about, and that would be a promising line of investigation.

This chapter also contains the first concrete example in this thesis of my belief that formal verification can give a better understanding, and even guide the design, of the systems it analyzes. The PRN model in general allows a very wide class of reaction mechanisms, but we can define a class of *augmented single-locus PRNs* where each individual reaction mechanism must occur within a bounded region of the polymer, regardless of the size of the polymer it happens to. Intuitively, such mechanisms are "physically realistic" in a way that mechanisms without this property aren't; DSD mechanisms, for example, are single-locus. More importantly, we show that any implementation PRN that is augmented single-locus cannot be a correct implementation, according to PRN bisimulation, of a formal PRN that isn't. Further, any augmented single-locus PRN can be implemented (correctly according to PRN bisimulation) by the right combination of four types of mechanisms, three of which have proposed DSD implementations [56, 57]. So even within the basics of PRN bisimulation theory, we have identified a class of PRNs that is likely easy to implement and a class that is hard or even impossible, and made it much easier to find a candidate DSD implementation for the former class.

The remainder of this chapter is a slightly modified version of the following manuscript, recently submitted to Theoretical Computer Science:

> Robert F. Johnson and Erik Winfree. Verifying polymer reaction networks using bisimulation. *Submitted to Theoretical Computer Science*, 2020.
> Contributions: Work done primarily by RFJ with advice and assistance from EW.

Abstract

The Chemical Reaction Network model has been proposed as a programming language for molecular programming. Methods to implement arbitrary CRNs using DNA strand displacement circuits have been proposed, as have methods to prove the correctness of those or other implementations. However, the stochastic Chemical Reaction Network model is provably not deterministically Turing-universal, that is, it is impossible to create a stochastic CRN where a given output molecule is produced if and only if an arbitrary Turing machine accepts. A DNA stack machine that can simulate arbitrary Turing machines with minimal slowdown deterministically has been proposed, but it uses unbounded polymers that cannot be modeled as a Chemical Reaction Network. We propose an extended version of a Chemical Reaction Network that models unbounded linear polymers made from a finite number of monomers. This Polymer Reaction Network model covers the DNA stack machine, as well as copy-tolerant Turing machines and some examples from biochemistry. We adapt the bisimulation method of verifying DNA implementations of Chemical Reaction Networks to our model, and use it to prove the correctness of the DNA stack machine implementation. We define a subclass of single-locus Polymer Reaction Networks and show that any member of that class can be

bisimulated by a network using only four primitives, suggesting a method of DNA implementation. Finally, we prove that deciding whether an implementation is a bisimulation is Π_2^0-complete, and thus undecidable in the general case, although it is tractable in many special cases of interest. We hope that the ability to model and verify implementations of Polymer Reaction Networks will aid in the rational design of molecular systems.

3.2 Introduction

Background

We consider the problem of how molecules can compute: how do biological systems use their components to compute, and what computing systems can be built with biological or bio-compatible molecules? For relatively small molecules in a well-mixed solution, the well-studied Chemical Reaction Network (CRN) model is a natural way to describe them. Known examples of computation with CRNs include useful small devices such as the approximate majority CRN [2, 19] and the rock-paper-scissors oscillator [25, 44, 67], boolean circuits [50] and neural networks [38], as well as more general results, including deterministic computation of arbitrary semilinear functions [1, 16, 26] and simulation of Turing machines with arbitrarily small error probability [65]. Further, computationally interesting (or uninteresting) CRNs can be "compiled" into physical devices: Soloveichik et al. [66], Qian et al. [57], and Cardelli [6], among others, give schemes to construct a DNA Strand Displacement (DSD) circuit that implements an arbitrary CRN.

One assumption of the CRN model is that the molecules are in a "well-mixed solution": that there is no concept of geometry or spatial organization of the molecules, that any pair of molecules is as likely to interact as any other, and that the only relevant information about the current state is the count (or concentration) of each molecule present. For small circuits like the ones mentioned above, this is quite reasonable. For classic models of computation and for biological systems, however, this assumption doesn't match: Turing machines, DNA/RNA/Proteins, and the cytoskeleton are all fundamental examples and fundamentally geometric. A counting argument suggests why: consider a system with k "types of object" (e.g. chemical species, Turing machine tape symbols) and a state of that system with n total objects. In a well-mixed CRN, the number of possible such states is on the order of (but less than) n^k; in a Turing machine or other geometric system, that number is on the order of k^n. In uniform computation—a single machine built to handle arbitrarily large computations—we have a constant k with n scaling with the size of the computation; so for example, the CRN that simulates Turing machines mentioned above uses around 3^n copies of a given molecule to simulate a Turing machine with n tape squares filled [65].

For such reasons, researchers have begun building molecular computing systems that take advantage of geometry. There are a number of variations on the concept of a DNA walker moving around a surface, often DNA origami, in a programmable way; a particularly complex example is the cargo-sorting robot of Thubagere et al. [71]. Chatterjee et al. have built logic circuits on origami, using a constant number of components regardless of the size of the circuit [15]. In the examples closest to abstract CRNs, Cardelli and Zavattaro discussed a CRN-like model with association and dissociation [9]; Qian et al. proposed a DNA implementation of a generic stack machine [57]; and Qian and Winfree proposed a DNA implementation of CRN-like reactions localized on a surface [56].

Also relevant to this topic are theoretical results on the computational power of well-mixed CRNs, and the difference in power between well-mixed CRNs and geometry-using models. The two most relevant results are the result that well-mixed CRNs that "always eventually" compute the right answer (in a certain sense well-defined in the theorem), can compute exactly the semilinear functions [1, 16]; and the result that the reachability problem for CRNs is decidable by a Turing machine [42, 47]. The reachability problem is in an informal sense the CRN equivalent of the Turing machine halting problem; in particular, any CRN trying to simulate a Turing machine must have some reachable state that involves an error. Thus those CRNs that try to simulate Turing machines can either do so in a non-uniform sense, where a single CRN can simulate a Turing machine with a given bound on its tape size, and a larger CRN must be created to simulate a larger Turing machine tape [41, 50]; or do so uniformly but with some probability of error, and due to the counting argument above, using species counts exponential in the space used by the Turing machine [65]. Polymer systems such as the Biochemical Ground Form [9], the DNA stack machine [57], and Surface CRNs [56], can all simulate Turing machines with no chance of error and using the same amount of space as the Turing machine.

We focus in this paper on verification of polymer systems. Specifically, we focus on the problem of, given an abstract description of a polymer system and a physical system, does the physical system "do the same thing" as the abstract description? For example, we might compare the abstract description of the DNA stack machine to its actual physical implementation [57], and wonder if the properties of a stack machine are preserved. This problem came up in the finite CRN case, where verification methods found subtle errors in some of the proposed CRN compilation schemes. Verification methods developed for finite CRNs include serializability analysis [46], pathway decomposition [64], and our previous work on CRN bisimulation [40]. Each of those methods has various advantages and disadvantages relative to the others, but all are capable of proving relevant correspondences between the behavior of physical CRN implementations and the abstract CRNs, or pointing out implementations that fail to correctly correspond to the abstract CRNs. Focusing on CRN bisimulation, our previous work discusses a method of "interpreting" the chemical species of the physical system as combinations of species of the abstract system, then asking if the possible qualitative behaviors of the two systems are equivalent up to that interpretation. This bisimulation method has a natural adaptation to polymer systems, which we will show.

Structure of the paper

In this paper, we show how CRN bisimulation can be adapted to polymer CRN-like systems, and help design practical such systems. In Section 3.3 we define a model of "linear Polymer Reaction Networks" henceforth referred to as PRNs. PRNs are a special case of CRNs with (usually) infinitely many species and reactions, and the PRN model covers most of the behavior we want while being convenient for discussion of bisimulation. This model is based on species being arbitrary strings over a finite set of "monomers", and a finite set of "reaction schemata" with wildcards from which reactions can be enumerated. Because PRNs are a special case of infinite CRNs, and most of the theorems of CRN bisimulation do not require the CRNs to be finite, CRN bisimulation can with a few new concepts be adapted to PRNs; we show how to do this, and define PRN bisimulation. This PRN bisimulation can be used to verify designs for physical implementations of polymer systems, which we show in Section 3.4 by proving correct an updated version of the DNA stack machine by Qian et al. [57]. Although many of the theorems (such as transitivity and modularity) from our previous work on CRN bisimulation [40] still apply to PRN bisimulation, the algorithms for finding or checking a bisimulation interpretation assume finite CRNs, and in Section 3.5 we show that the corresponding problems are undecidable for polymer systems.

We believe that CRN and PRN bisimulation are not just useful for verifying systems once designed, but can be used as "goalposts" to help guide the design of CRN and PRN implementations. For example, a proof by bisimulation that a certain small class of reaction mechanisms is sufficient to implement any of a larger class of reactions, suggests a design strategy involving implementing that small class of reactions. In Section 3.6 we give an example of such a proof that any of a class of "single-locus reaction schemata", which capture most of what we think of as physically realistic single-step reactions, can be implemented up to PRN bisimulation by a specific set of five reaction primitives. Finally, since our linear PRNs are only one of many reasonable models of a polymer CRN-like system, in Section 3.7 we show how PRN bisimulation might be defined for other such systems, and that most or all of our theorems still apply or can be easily translated.

3.3 Definitions

Multisets and automata

$\mathbb{N}$ is the set of natural numbers, $\{0, 1, 2, \dots\}$. Where A is a set, $\mathbb{N}^A$ is the set of multisets of elements of A, or equivalently, the set of functions from A to $\mathbb{N}$. Where $S \in \mathbb{N}^A$, $X \in A$ we write $S(X)$ for the count of X in S; this is consistent with S as a function $A \to \mathbb{N}$. Addition and scalar multiplication of multisets are defined componentwise. Comparison is also defined, $S \geq T$ means $\forall_X S(X) \geq T(X)$, and $S > T$ if $S \geq T$ and $S \neq T$. As we are only concerned with finite multisets, if A is infinite we use $\mathbb{N}^A$ to mean the set of multisets S with $\sum_{X \in A} S(X) < \infty$. We use the notation $\{\!|\dots|\!\}$ for multisets, e.g. $\{\!|X, Y|\!\}$ or $\{\!|2X, Z|\!\}$.

Where Σ is a set, Σ^* is the set of strings of 0 or more elements of Σ. ε is the empty string.

We use finite automata, stack automata, and Turing machines for various purposes. We generally assume familiarity with them, but give a brief description. A (nondeterministic) *finite automaton* (FA or NFA) is $M = (Q, \Sigma, \delta, q_0, F)$. Q is a set of states, Σ an alphabet, $\delta \subset (Q, \Sigma, Q)$ a transition relation, $q_0 \in Q$ a start state, $F \subset Q$ a set of accepting states. (If δ is a function $(Q, \Sigma) \to Q$, then the automaton is *deterministic* (DFA).) The automaton *accepts* a string $w \in \Sigma^*$ if there is a sequence $q_0 w_1 q_1 \ldots w_n q_n$ with $(q_{i-1}, w_i, q_i) \in \delta$ and $q_n \in F$. A language $L \subset \Sigma^*$ is the language accepted by a finite automaton if and only if it is the language that matches some regular expression, and we often use the two interchangeably [43]. We also use $L(M)$ or $L(e)$ to mean the languages of an NFA or regular expression respectively, so $w \in L(M)$ or $w \in L(e)$ mean w is accepted by M/matches e.

A stack machine is effectively a finite automaton with one or more last-in-first-out memory stacks; where Γ is a stack alphabet, $\delta \subset (Q, \Sigma \cup \{\varepsilon\}, \Gamma \cup \{\varepsilon\}, Q, \Gamma \cup \{\varepsilon\})$ indicates an initial state, symbol to read from the input string, symbol to pop off the stack, target state, and symbol to push onto the stack. Note that any of these three may be empty (ε), i.e. a step can advance the input string and/or pop from the stack and/or push to the stack without doing all three. Acceptance is the same as in the NFA case. Section 3.4 and the DNA stack machine [57] use a variation where the input string is placed on stack 1 and each transition can *either* push *or* pop, but not both. Similarly, a Turing machine is effectively a finite automaton with an unbounded tape of memory, with $\Sigma \subset \Gamma$ and the input string being the initial tape contents. Here $\delta \subset (Q, \Gamma, Q, \Gamma, \{L, R\})$ reads a symbol, writes a symbol, and moves left or right on the tape. We usually deal with deterministic stack and Turing machines.

Chemical Reaction Networks

Definition 3.3.1. A *Chemical Reaction Network* (CRN) is a pair $(\mathcal{S}, \mathcal{R})$, where $\mathcal{S}$ is a set of species and $\mathcal{R} \subset \mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}$ is a set of reactions.

We often use chemical reaction notation to write reactions: $(R, P) = R \to P$. If (R, P) and (P, R) are both reactions, we write $R \rightleftharpoons P$. Consistent with chemical reaction notation, when unambiguous we often identify each species A with the multiset $\{\!|A|\!\}$, so e.g. $A + B$ and $\{\!|A, B|\!\}$ refer to the same object. In general CRNs, each reaction is given a positive real number as a "rate constant", so a reaction is a triple (R, P, k), sometimes written as $R \xrightarrow{k} P$. These rate constants affect the amount each reaction happens in a given time interval and, in the stochastic model, the likelihood of a reaction happening relative to other reactions. The theory of CRN and PRN bisimulation deals with whether a thing *can* happen in CRNs, but not how likely it is or how much time it takes, and for those questions the values of rate constants are irrelevant (as long as they are all positive real numbers). Thus for the purposes of this paper we define reactions as pairs of reactants and products without rate constants. We further assume that no reactions $R \to P$ with $R = P$ exist.

We work with the stochastic model of CRN semantics, where a CRN starts with some count of each species present, and any possible reaction may occur, which changes the counts. Specifically, a CRN will at any point in time be at a state $S \in \mathbb{N}^{\mathcal{S}}$, a multiset of species, and for each reaction $R \to P$ where $S \geq R$ the CRN can transition from state S to state $S - R + P$. Given rate constants, this process is a continuous-time Markov chain with transition rates dependent on the rate constants and the count of reactants in S; when we only care about which transitions are possible, the previous description is equivalent to that continuous-time Markov chain.

Consider a pair of CRNs $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$, where $(\mathcal{S}, \mathcal{R})$ is some abstract CRN and $(\mathcal{S}', \mathcal{R}')$ a more realistic CRN intended to implement $(\mathcal{S}, \mathcal{R})$. We call $(\mathcal{S}, \mathcal{R})$ the *formal CRN* and $(\mathcal{S}', \mathcal{R}')$ the *implementation CRN*. We previously defined a concept of *CRN bisimulation* to check whether the implementation CRN is, in fact, a correct implementation of the formal CRN [40]. (The citation [40] is, in fact, Chapter 2 of this thesis.) CRN bisimulation is based on an *interpretation* of each implementation species as a multiset of formal species, where the implementation is correct if (for some interpretation) from any initial state the possible formal trajectories and interpreted implementation trajectories are equivalent. An example DNA implementation with interpretation is shown in Figure 3.2.

Definition 3.3.2. An *interpretation* is a function $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ from implementation species to multisets of formal species. We extend this linearly from species to states: $m(\sum_{i=1}^{n} a_i X_i) = \sum_{i=1}^{n} a_i m(X_i)$. We also define a *natural interpretation of reactions*: $m(R' \to P') = m(R') \to m(P')$ unless $m(R') = m(P')$, in which case $m(R' \to P') = \tau$ and we say the reaction is *trivial*. For example, if $m(i_{AB}) = A + B$, $m(x_A) = A$, and $m(t_{BC}) = B + C$ then $m(i_{AB} + x_A) = 2A + B$, and $m(i_{AB} \to x_A + t_{BC}) = A + B \to A + B + C$.

In our previous work [40] we considered the possibility that an implementation of a reaction might have "spurious catalysts", i.e. extra species formally present in the interpretation that are not involved in or affected by the intended formal reaction. For example, in a physical DNA-based implementation, an extra strand might bind to some part of the reacting complex without affecting the actual reaction mechanism. This turns out to be a major concern in even abstract polymer systems, so we bring in that definition here.

Definition 3.3.3. Let $(\mathcal{S}, \mathcal{R})$ and $(\mathcal{S}', \mathcal{R}')$ be a formal and implementation CRN with $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ an interpretation of implementation species, which is extended to implementation states as in Definition 3.3.2. An interpretation of reactions $m_r : \mathcal{R}' \to (\mathbb{N}^{\mathcal{S}} \times \mathbb{N}^{\mathcal{S}}) \cup \{\tau\}$ is *consistent with* m if, for each $R' \to P' \in \mathcal{R}'$:

(i) If $m(R') = m(P')$ then $m_r(R' \to P') = \tau$, and

(ii) If $m(R') \neq m(P')$ then $m_r(R' \to P') = R \to P$ for some $R, P, C \in \mathbb{N}^{\mathcal{S}}$ with $m(R') = R + C$ and $m(P') = P + C$.

Naturally, the natural interpretation of reactions given an interpretation m is in fact consistent with m. In general below we abuse notation and use m to refer to all of the interpretation of species, the extension to states, and the chosen interpretation of reactions (natural or otherwise) consistent with the interpretation of species.

We defined correctness of an interpretation in three ways: trajectory equivalence, three conditions, and weak bisimulation. Loosely, trajectory equivalence is what we want "correctness" to imply, the three conditions are easy to define and check, and weak bisimulation is the well-studied theory of which CRN bisimulation is an instance. A theorem from our previous work proves that these three definitions are equivalent, as desired. As notation, $S \xrightarrow{r}$ means reaction r can occur in state S, while $S \xrightarrow{r} T$ means that reaction r takes state S to state T. In the implementation CRN, $S' \overset{r'}{\Rightarrow}$ and $S' \overset{r'}{\Rightarrow} T'$ mean the same for a sequence of zero or more trivial reactions followed by r'. Where S' is an implementation state and r is a formal reaction, $S' \xrightarrow{r} T'$ means "$S' \xrightarrow{r'} T'$ for some r' with $m(r') = r$", and similarly for $S' \overset{r}{\Rightarrow} T'$, $S' \xrightarrow{r}$, and $S' \overset{r}{\Rightarrow}$. In the formal CRN, $S \overset{r}{\Rightarrow} T$ is equivalent to $S \xrightarrow{r} T$.

Definition 3.3.4 (Three notions of correctness)**.** An implementation CRN $(\mathcal{S}', \mathcal{R}')$ is a correct implementation of a formal CRN $(\mathcal{S}, \mathcal{R})$ if a correct interpretation exists. An interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ is correct, in which case we

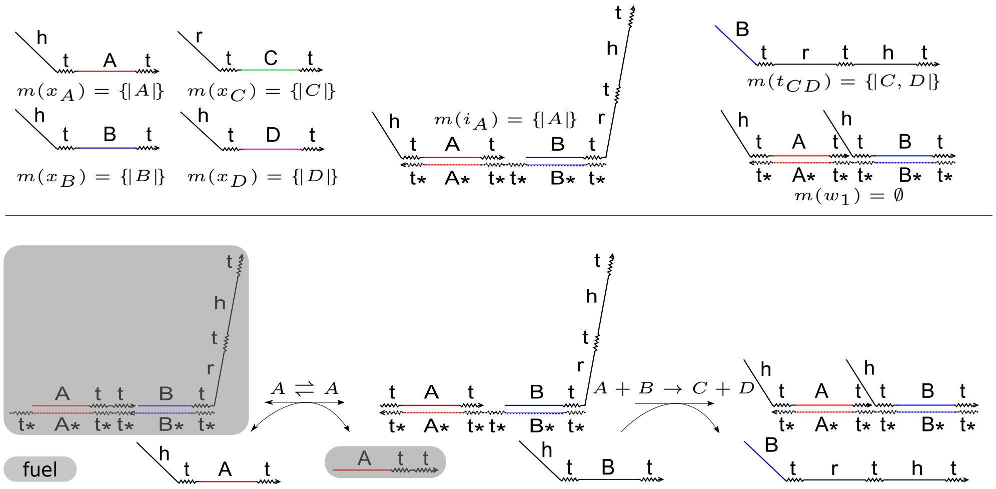


Figure 3.2: An example implementation CRN with interpretation. Various DNA complexes in the implementation system are modeled as species of the implementation CRN with interpretations as multisets of formal species. (Complexes marked as "fuel" are assumed always present and are *not* modeled as species in the implementation CRN [40, 66]. For example, the reaction on the left is enumerated as $x_A \rightleftharpoons i_A$, ignoring the two fuel complexes.) Interpretations of trivial (left) and nontrivial (right) reactions follow from the interpretations of the species involved. Figure adapted from [40].

say m *is a CRN bisimulation*, if any of the following three sets of conditions are true:

I **Equivalence of trajectories**

(i) The set of formal trajectories and interpretations of implementation trajectories are equal.

(ii) For every implementation state S', the set of formal trajectories starting from $m(S')$ and interpretations of implementation trajectories starting from S' are equal.

II **Three conditions on the interpretation**

(i) *Atomic condition:* For every formal species A, there exists an implementation species x_A such that $m(x_A) = \{\!|A|\!\}$.

(ii) *Delimiting condition:* The interpretation of any implementation reaction is either trivial or a valid formal reaction.

(iii) *Permissive condition:* If $S \xrightarrow{r}$ and $m(S') = S$, there exists an implementation reaction r' such that $m(r') = r$ and $S' \overset{r'}{\Rightarrow}$.

III **Weak bisimulation**

(i) For all implementation states S':
if $S' \xrightarrow{r} T'$, then $S \overset{r}{\Rightarrow} T$ where $S = m(S')$ and $T = m(T')$.

(ii) For all formal states S, there exists S' with $m(S') = S$, and for all such S':
if $S \xrightarrow{r} T$, then for some T', $S' \overset{r}{\Rightarrow} T'$ and $m(T') = T$.

Our previous work proved a number of theorems about CRN bisimulation. For this work, the relevant ones are those that do not assume the CRNs involved are finite. In particular, the equivalence of the three definitions of correctness, the transitivity lemma, and the modularity condition will all apply to polymer systems.

Theorem 3.3.1. *The three definitions of correctness, namely trajectory equivalence, the three conditions on the interpretation, and weak bisimulation, are equivalent.*

Lemma 3.3.1. *(Transitivity) If m_2 is a bisimulation from $(\mathcal{S}'', \mathcal{R}'')$ to $(\mathcal{S}', \mathcal{R}')$ and m_1 is a bisimulation from $(\mathcal{S}', \mathcal{R}')$ to $(\mathcal{S}, \mathcal{R})$, then $m = m_1 \circ m_2$ is a bisimulation from $(\mathcal{S}', \mathcal{R}')$ to $(\mathcal{S}, \mathcal{R})$.*

(It is an abuse of notation to write $m_1 \circ m_2$ when m_2 takes $\mathcal{S}'' \to \mathbb{N}^{\mathcal{S}'}$ and m_2 takes $\mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$. Intuitively, we extend m_1 to an interpretation on multisets over $\mathcal{S}'$; formally, for $x \in \mathcal{S}''$, $m(x) = \sum_{y \in \mathcal{S}'} m_2(x)(y) m_1(y)$, where $m_2(x)(y)$ means the count of y in $m_2(x)$ and is a scalar multiplier for the multiset $m_1(y)$.)

Definition 3.3.5 (Modularity Condition)**.** Let m be a bisimulation from $(\mathcal{S}', \mathcal{R}')$ to $(\mathcal{S}, \mathcal{R})$. Let $\mathcal{S}'_0 \subset \mathcal{S}'$ and $\mathcal{S}_0 \subset \mathcal{S}$ be subsets of implementation and formal species, respectively, where $y \in \mathcal{S}'_0 \Rightarrow m(y) \subset \mathcal{S}_0$. We say that m is a *modular interpretation* with respect to the *common (implementation and formal) species* $\mathcal{S}'_0$ and $\mathcal{S}_0$ if for any $x \in \mathcal{S}'$ there is a sequence of trivial reactions $\{\!|x|\!\} \overset{\tau}{\Rightarrow} Y + Z$ where $Y \subset \mathcal{S}'_0$ and $m(Z) \cap \mathcal{S}_0 = \emptyset$, i.e. all common formal species in the interpretation of x are extracted as common implementation species.

Theorem 3.3.2. *(Modularity) Let* m_1 *be a bisimulation from* $(\mathcal{S}'_1, \mathcal{R}'_1)$ *to* $(\mathcal{S}_1, \mathcal{R}_1)$ *and* m_2 *be a bisimulation from* $(\mathcal{S}'_2, \mathcal{R}'_2)$ *to* $(\mathcal{S}_2, \mathcal{R}_2)$ *where* m_1 *and* m_2 *agree on* $\mathcal{S}'_1 \cap \mathcal{S}'_2$. *Let* $\mathcal{S}' = \mathcal{S}'_1 \cup \mathcal{S}'_2$, $\mathcal{R}' = \mathcal{R}'_1 \cup \mathcal{R}'_2$, $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, *and* $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$, *and* $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ *equal* m_1 *on* $\mathcal{S}'_1$ *and* m_2 *on* $\mathcal{S}'_2$. *If* m_1 *and* m_2 *are both respectively modular bisimulations with respect to the common implementation species* $\mathcal{S}'_1 \cap \mathcal{S}'_2$ *and common formal species* $\mathcal{S}_1 \cap \mathcal{S}_2$, *then* m *is a bisimulation from* $(\mathcal{S}', \mathcal{R}')$ *to* $(\mathcal{S}, \mathcal{R})$, *and* m *is also modular with respect to* $\mathcal{S}'_1 \cap \mathcal{S}'_2$ *and* $\mathcal{S}_1 \cap \mathcal{S}_2$.

Polymer Reaction Networks

A polymer reaction network, like a Chemical Reaction Network, will be a set of species and a set of reactions. Unlike a typical CRN, a typical polymer system allows arbitrarily long polymers to be made from its set of monomers, and allows the same "reactions" to occur among monomers regardless of the content of the rest of the polymer. When modeled as a CRN, such a system would typically have an infinite number of both species and reactions. To handle this, we define a Polymer Reaction Network as a finite *species schema* and a finite set of *reaction schemata*, which will then generate the set of species and reactions.

The species of a polymer system are, in general, arbitrarily long polymers made up of a finite set of monomers. While polymers with branches and loops can exist, we wished to avoid the associated complexity. As many of the essential features we wish to study arise in linear polymer systems, we focus on those. (We discuss further the reasons for this in Section 3.7.) Thus our species will be strings over some finite "alphabet" or set of monomers. (We assume that "left" and "right" are distinguished, so that the strings e.g. *abc* and *cba* are different molecules; a *b* monomer in this example would have two distinct binding sites, and the molecules differ in which site is bound to *a* and which to *c*. Again Section 3.7 contains discussion on what can happen if this assumption is not true.)

In a physical system, typically not every string of monomers can actually exist as a polymer; some pairs of monomers will have the appropriate interfaces for binding to each other, and other pairs will not. Similarly, we assume that only some monomers can occur on the left edge of a polymer, which we represent by $\vdash$, and similarly for the right edge $\dashv$; some monomers might not be stable when

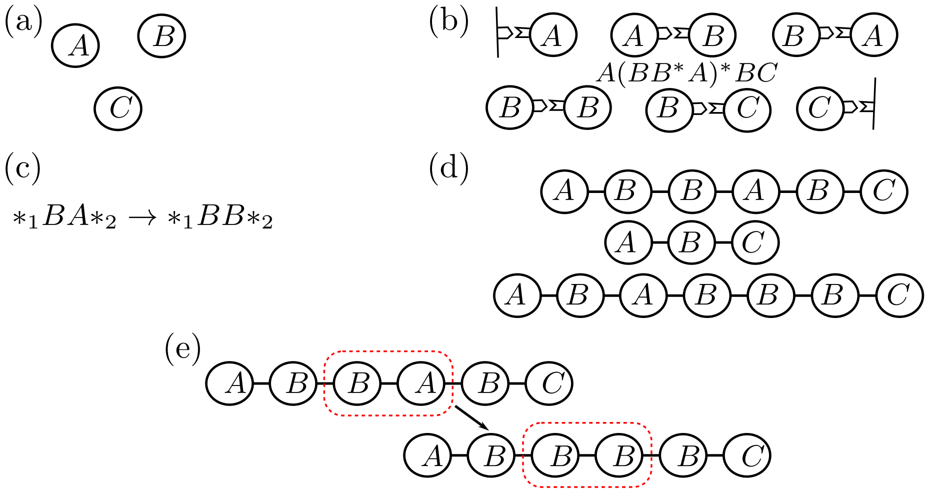


Figure 3.3: A PRN is defined as a set of monomers (a), a regular expression restriction (which may be given as a compatibility relation, b), and a set of reaction schemata (one shown in c). The species of a PRN are the set of all strings of monomers that match the regular expression (three examples shown in d). Reactions are obtained by substituting strings of monomers for wildcards in the reaction schemata such that both sides of the reaction respect the compatibility relation; for example, $*_1 = AB$ and $*_2 = BC$ in $*_1BA*_2 \to *_1BB*_2$ enumerates the reaction $ABBABC \to ABBBBC$ (e).

unbound. We model this by letting the set of species to be restricted to those that match a specified regular expression. We justify this by showing that a more physically meaningful restriction, of only allowing certain monomers to bind to each other and to the edges, is equivalent to a regular expression up to interpretation.

(One might ask why it is necessary to restrict the set of possible polymers at all. To answer that, intrinsic to our notion of CRN bisimulation is that the behaviors of the two CRNs are equivalent from *any* initial state, and we would like to have the same guarantee for PRNs. Many systems would have some polymers that can never exist physically, but if they could exist, would have absurd behavior that breaks the system. Either the regular expression restriction or the local compatibility restriction can solve this problem.)

Given an infinite set of species generated from a finite set of monomers, we would like to specify the possible reactions of those species with a finite set of rules. A reasonable assumption is that there are a finite number of "reaction mechanisms", each of which depends on some features of its context but may be independent of others. To use an example from the stack machine of Qian et al. [57], a 0_2 (symbol 0 on stack 2) unit at the right end of a polymer can react with a query strand Q_2, removing the 0_2 symbol while leaving the rest of that polymer unchanged, and releasing a signal strand, which we call 0_2^f. This reaction depends on the 0_2 symbol being on the end of its polymer, but is independent of what else makes up the remainder of the polymer. We could write this reaction mechanism as $*_1 0_2 + Q_2 \to *_1 + 0_2^f$, where the string $*_1 0_2$ means "any polymer that ends in 0_2". Here $*_1$ is a *wildcard*, which can be filled in by any string, provided that the same wildcard is replaced by the same string in each of its appearances; since there are infinitely many possible strings that can replace $*_1$, this *reaction schema* generates infinitely many reactions. So for example, $\lambda_2 1_2 0_2 1_2 0_2 + Q_2 \to \lambda_2 1_2 0_2 1_2 + 0_2^f$ would match this schema, but $\lambda_2 1_2 0_2 + Q_2 \to \lambda_2 + 0_2^f$ would not. Other mechanisms can also be described with wildcards: $*_1 + P \to *_1 + *_1 + P$ models P catalytically copying an arbitrary string, for example, while $*_1 AB *_2 \to *_1 *_2$ models AB removing itself from anywhere in a polymer. We thus define the reactions of a PRN by a set of *reaction schemata*, each of which is a reaction over strings including wildcards, and generate the reactions of a PRN by substitution into the wildcards of the schemata.

As is usual in CRN notation, we will write $R \rightleftharpoons P$ to represent the pair of reaction schemata $R \to P$ and $P \to R$. This is valid if every wildcard that appears in either R or P appears in both, and if so, observe that the schema is truly reversible: any reaction enumerated by one direction will have its reverse reaction enumerated by the other.

Definition 3.3.6. A *Polymer Reaction Network* is a triple (Σ, e, Ψ) of a monomer set Σ, regular expression e over Σ, and set of reaction schemata Ψ. When the reaction schemata are irrelevant, we refer to the pair (Σ, e) as a *species schema*. A *reaction schema* $\psi \in \Psi$ is a pair (R, P) of multisets of strings over $\Sigma \cup *_{\mathbb{N}}$ where $*_{\mathbb{N}} = \{*_1, *_2, \dots\}$, such that any $*_n$ that appears in either R or P appears at least once in R. Given a PRN, it induces an *enumerated CRN* $(\mathcal{S}, \mathcal{R})$. (We sometimes write $\mathcal{S}(\Sigma, e)$, $\mathcal{R}(\Sigma, e, \Psi)$.) $\mathcal{S}$ is the set of all nonempty strings over Σ that match e. To enumerate $\mathcal{R}$, consider a reaction schema $\psi = (R, P) \in \Psi$, and for each $*_n$ that appears in ψ, choosing a string s_n and substituting s_n for each appearance of $*_n$, to obtain a pair of multisets of strings over Σ. If every string obtained this way (in both R and

P) matches e, then the pair of multisets is a reaction of species in $\mathcal{S}$; $\mathcal{R}$ is the set of all such reactions.

Definition 3.3.7. An *augmented PRN* is a triple (Σ, e, Ψ) of a monomer set Σ, regular expression e over Σ, and set of augmented reaction schemata Ψ. An *augmented reaction schema* $\psi \in \Psi$ is a reaction schema (R, P) together with, for each $*_n$ that appears in the schema, a regular expression e_n over Σ. The enumerated CRN $(\mathcal{S}, \mathcal{R})$ has $\mathcal{S}$ enumerated as usual, while $\mathcal{R}$ is the set of reactions enumerated as for an unaugmented PRN with the restriction that in a schema ψ, each string s_n substituted for $*_n$ must match e_n.

We do not discuss augmented PRNs much until Section 3.6, where among other results we show that an augmented PRN can be implemented, up to PRN bisimulation, by a non-augmented PRN.

Consider a particular type of mechanism that restricts on which strings over Σ are valid polymers: only some pairs of monomers have the complementary binding sites necessary to bind. We might also assume that only some monomers are stable on the left edge of a polymer, and only some monomers are stable on the right. We can write this as a relation $\rho \subset \Sigma_\epsilon$, where $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$, $\epsilon \notin \Sigma$. Then $a\rho b$ means ab can bind in that order in a polymer; similarly $\epsilon\rho a$ ($a\rho\epsilon$) means a can occur on the left (right) end of a polymer. Such a relation cannot be more powerful than a regular expression, and up to an interpretation (as defined below), we show that it is as powerful as a regular expression.

Definition 3.3.8. Given a monomer set Σ with notation $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$, $\epsilon \notin \Sigma$, a *compatibility relation* is a relation $\rho \subset \Sigma_\epsilon \times \Sigma_\epsilon$. Given a monomer set Σ and compatibility relation ρ, the set of enumerated species $\mathcal{S}(\Sigma, \rho)$ is the set of all $w = x_1 \ldots x_n \in \Sigma^*$ such that all of $\epsilon\rho x_1$, $x_i \rho x_{i+1}$, and $x_n \rho \epsilon$. As we show below that any compatibility relation can be described by a regular expression, a PRN (augmented or not) can be given as (Σ, ρ, Ψ) instead of (Σ, e, Ψ).

Lemma 3.3.2. *For any regular expression e over an alphabet Σ, there is a monomer set Σ', compatibility relation ρ', and interpretation $\pi : \Sigma' \to \Sigma$ such that a string $x = x_1 \ldots x_n \in \Sigma^* \in L(e)$ if and only if there is a species $x' = x'_1 \ldots x'_n \in \mathcal{S}(\Sigma', \rho')$ with $\pi(x'_i) = x_i$ for $1 \leq i \leq n$. This construction can be done with $\Sigma' = \Sigma$, $\forall_x \pi(x) = x$ if and only if e has the property that for*

$x \in \Sigma$, $u_1, v_1, u_2, v_2 \in \Sigma^*$, if $u_1 x v_1$ and $u_2 x v_2$ match e then so does $u_1 x v_2$; if so, we say that e is local*. Conversely, given any monomer set Σ with compatibility relation ρ there is a local regular expression e with $\mathcal{S}(\Sigma, e) = \mathcal{S}(\Sigma, \rho)$. All of these transformations can be computed in polynomial time.*

Proof. Consider a nondeterministic finite automaton M that recognizes strings that match e. Where Q is the set of states of M, let $\Sigma' = \Sigma \times Q$ and let $\pi((x, q)) = x$. Let $((x_1, q_1), (x_2, q_2)) \in \rho'$ if and only if M can transition from state q_1 to state q_2 by reading x_2, let $(\vdash, (x, q)) \in \rho'$ if and only if M can transition from its start state q_0 to q by reading x, and $((x, q), \dashv) \in \rho'$ if and only if q is an accept state of M. Valid polymers correspond exactly to accepting computation paths of M on their interpretations.

If e is local, then for any states q_i, q_j both of which have incoming transitions labeled x, either one of those transitions is not reachable on any string and can be removed, or given locality, the set of strings accepted after reaching q_i and q_j are the same. In that case, an equivalent automaton has q_i and q_j collapsed into one state. Repeating this process constructs a finite automaton that recognizes e where for each state x there is at most one state q_x with incoming transitions labeled x (it may be that $q_x = q_y$ for $x \neq y$). Applying the above construction to this new finite automaton and labeling the monomer (x, q_x) as x gives the desired (Σ, ρ).

Given (Σ, ρ), a finite automaton as above can be easily constructed: for each monomer x a state q_x, with $q_x \xrightarrow{y} q_y \iff x \rho y$, $q_0 \xrightarrow{x} q_x \iff \epsilon \rho x$, and q_x accepts if and only if $x \rho \epsilon$. As above, valid polymers correspond exactly to accepting computation paths of M. □

The concept of local regular expressions is the $k = 2$ case of *strict locally testable languages* [85], a connection which may be interesting to explore further.

PRN Examples

Because a PRN is an infinite CRN, we can simulate a PRN using the semantics of a stochastic CRN, such as Gillespie's algorithm [34]. Although the number of reactions in a PRN may be infinite, if at a given time there are a finite number of polymers each of finite length, then at that time there will be a finite number of reactions possible. The only difficulty is that the set of possible reactions in general must be re-computed every time a new species is produced, preventing various methods of optimizing the Gillespie algorithm that are possible for a finite CRN. However, implementing the basic Gillespie algorithm is possible, which has been done in Visual DSD for polymer-like systems created from DNA strand displacement networks [45]. For the examples in Figure 3.4, we assume that each reaction schema has a rate constant, that every reaction enumerated from that schema has that schema's rate constant, and that the same reaction enumerated multiple times (from different schemata and/or different substitutions into the same schema) has as its rate constants the sum of schema rate constants from all of its enumerations (which is guaranteed to be finite). Other methods of specifying rate constants are possible.

Figure 3.4 gives examples that showcase various relevant features of the Polymer Reaction Network model. We discuss some of those examples in further detail here.

Example 3.3.1 (Dynamic instability)**.** Shown in the upper left of Figure 3.4, this PRN has 5 monomers, $\Sigma = \{D, T, S, D_f, T_f\}$; regular expression restriction $SD^*T^* \mid D_f \mid T_f$ equivalent to the compatibility relation ρ shown; and 6 reaction schemata.

We give an example PRN that describes a model of dynamic instability, inspired by but not necessarily agreeing with biological microtubules. Our model, in English, is as follows: A polymer is a seed S followed by any number of D monomers then any number of T monomers; those two types of monomers can also exist free-floating in solution, represented by D_f and T_f. A free T_f monomer can attach onto the right end of a polymer as a T; T monomers can convert into D monomers starting from the left end of a polymer; and D monomers can fall off of either end, but fall off slowly from the left and very rapidly from the right. In solution, free D_f converts back into T_f, to complete the cycle.

In the compatibility relation, first observe the patterns $(\vdash, D_f)$, $(D_f, \dashv)$ and similarly $(\vdash, T_f)$, $(T_f, \dashv)$, with no other occurences of D_f or T_f; the result of this is that D_f and T_f can exist as monomers but can't polymerize. Effectively, D_f and T_f are CRN species. The only other presence of $\vdash$ is $(\vdash, S)$, so any polymer must start with S, and given $(S, \dashv) \in \rho$, can end immediately. Otherwise,

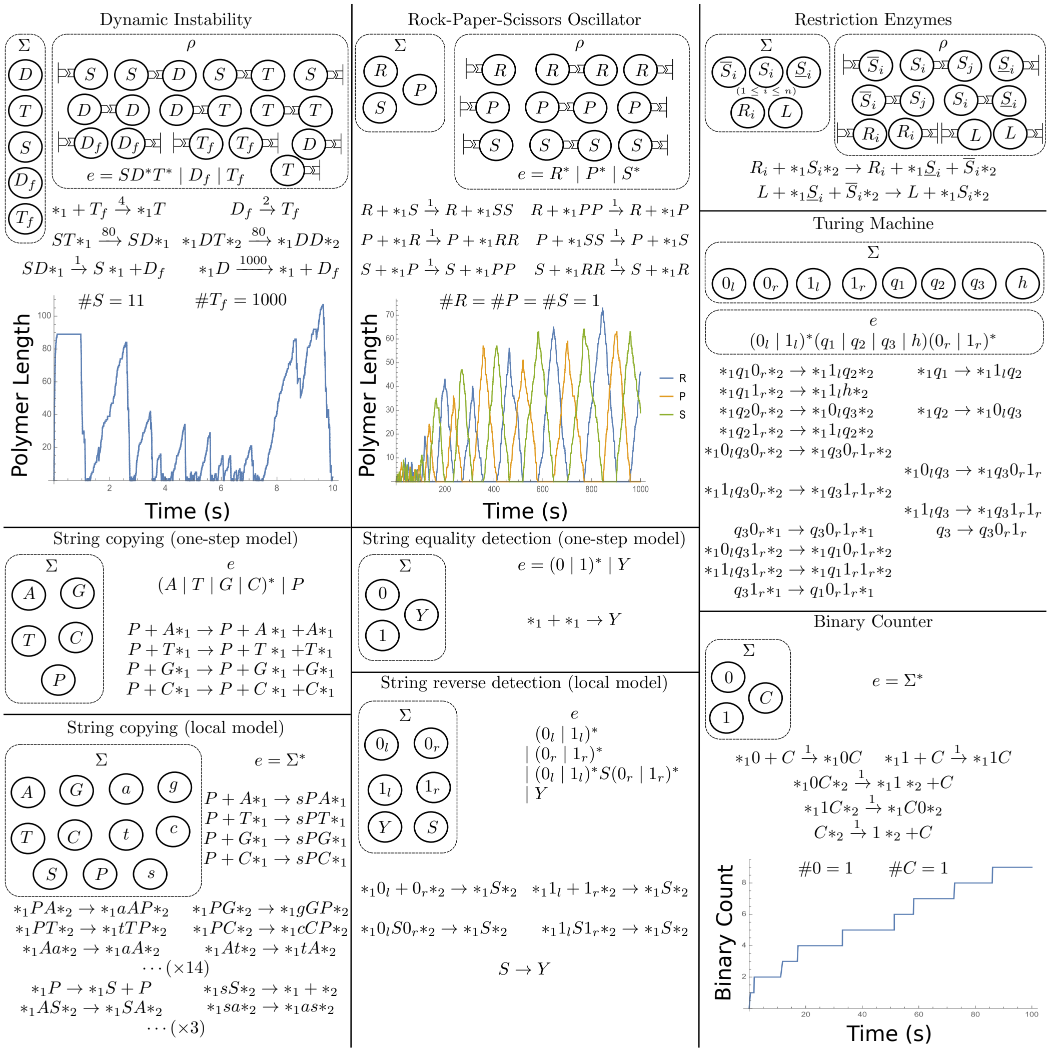

Dynamic Instability
Σ
D T S D_f T_f
ρ
S D T D_f T_f
$e = SD^*T^* \mid D_f \mid T_f$
$*_1 + T_f \xrightarrow{4} *_1T$
$D_f \xrightarrow{2} T_f$
$ST*_1 \xrightarrow{80} SD*_1$
$*_1DT*_2 \xrightarrow{80} *_1DD*_2$
$SD*_1 \xrightarrow{1} S*_1 + D_f$
$*_1D \xrightarrow{1000} *_1 + D_f$
$\#S = 11$
$\#T_f = 1000$
Polymer Length
100
80
60
40
20
2
4
6
8
10
Time (s)
Rock-Paper-Scissors Oscillator
Σ
R P S
ρ
$e = R^* \mid P^* \mid S^*$
$R + *_1S \xrightarrow{1} R + *_1SS$
$R + *_1PP \xrightarrow{1} R + *_1P$
$P + *_1R \xrightarrow{1} P + *_1RR$
$P + *_1SS \xrightarrow{1} P + *_1S$
$S + *_1P \xrightarrow{1} S + *_1PP$
$S + *_1RR \xrightarrow{1} S + *_1R$
$\#R = \#P = \#S = 1$
Polymer Length
70
60
50
40
30
20
10
200
400
600
800
1000
Time (s)
R
P
S
Restriction Enzymes
Σ
$\overline{S}_i$ S_i $\underline{S}_i$
$(1 \leq i \leq n)$
R_i L
ρ
$\overline{S}_i$ S_i S_j $\underline{S}_i$ R_i L
$R_i + *_1S_i*_2 \to R_i + *_1\underline{S}_i + \overline{S}_i*_2$
$L + *_1\underline{S}_i + \overline{S}_i*_2 \to L + *_1S_i*_2$
Turing Machine
Σ
0_l 0_r 1_l 1_r q_1 q_2 q_3 h
e
$(0_l \mid 1_l)^*(q_1 \mid q_2 \mid q_3 \mid h)(0_r \mid 1_r)^*$
$*_1q_10_r*_2 \to *_11_lq_2*_2$
$*_1q_1 \to *_11_lq_2$
$*_1q_11_r*_2 \to *_11_lh*_2$
$*_1q_20_r*_2 \to *_10_lq_3*_2$
$*_1q_2 \to *_10_lq_3$
$*_1q_21_r*_2 \to *_11_lq_2*_2$
$*_10_lq_30_r*_2 \to *_1q_30_r1_r*_2$
$*_10_lq_3 \to *_1q_30_r1_r$
$*_11_lq_30_r*_2 \to *_1q_31_r1_r*_2$
$*_11_lq_3 \to *_1q_31_r1_r$
$q_30_r*_1 \to q_30_r1_r*_1$
$q_3 \to q_30_r1_r$
$*_10_lq_31_r*_2 \to *_1q_10_r1_r*_2$
$*_11_lq_31_r*_2 \to *_1q_11_r1_r*_2$
$q_31_r*_1 \to q_10_r1_r*_1$
String copying (one-step model)
Σ
A G T C P
e
$(A \mid T \mid G \mid C)^* \mid P$
$P + A*_1 \to P + A*_1 + A*_1$
$P + T*_1 \to P + T*_1 + T*_1$
$P + G*_1 \to P + G*_1 + G*_1$
$P + C*_1 \to P + C*_1 + C*_1$
String equality detection (one-step model)
Σ
0 Y 1
$e = (0 \mid 1)^* \mid Y$
$*_1 + *_1 \to Y$
String copying (local model)
Σ
A G a g T C t c S P s
$e = \Sigma^*$
$P + A*_1 \to sPA*_1$
$P + T*_1 \to sPT*_1$
$P + G*_1 \to sPG*_1$
$P + C*_1 \to sPC*_1$
$*_1PA*_2 \to *_1aAP*_2$
$*_1PG*_2 \to *_1gGP*_2$
$*_1PT*_2 \to *_1tTP*_2$
$*_1PC*_2 \to *_1cCP*_2$
$*_1Aa*_2 \to *_1aA*_2$
$*_1At*_2 \to *_1tA*_2$
$\cdots (\times 14)$
$*_1P \to *_1S + P$
$*_1sS*_2 \to *_1 + *_2$
$*_1AS*_2 \to *_1SA*_2$
$*_1sa*_2 \to *_1as*_2$
$\cdots (\times 3)$
String reverse detection (local model)
Σ
0_l 0_r 1_l 1_r Y S
e
$(0_l \mid 1_l)^*$
$\mid (0_r \mid 1_r)^*$
$\mid (0_l \mid 1_l)^*S(0_r \mid 1_r)^*$
$\mid Y$
$*_10_l + 0_r*_2 \to *_1S*_2$
$*_11_l + 1_r*_2 \to *_1S*_2$
$*_10_lS0_r*_2 \to *_1S*_2$
$*_11_lS1_r*_2 \to *_1S*_2$
$S \to Y$
Binary Counter
Σ
0 C 1
$e = \Sigma^*$
$*_10 + C \xrightarrow{1} *_10C$
$*_11 + C \xrightarrow{1} *_11C$
$*_10C*_2 \xrightarrow{1} *_11*_2 + C$
$*_11C*_2 \xrightarrow{1} *_1C0*_2$
$C*_2 \xrightarrow{1} 1*_2 + C$
$\#0 = 1$
$\#C = 1$
Binary Count
8
6
4
2
20
40
60
80
100
Time (s)

Figure 3.4: These example Polymer Reaction Networks demonstrate various features of the PRN model. Shown for each PRN is the monomer set Σ; the regular expression e describing the set of polymers and/or, if e is local, the equivalent compatibility relation ρ; and the reaction schemata. Additionally for some PRNs, rate constants are assigned to each reaction schema and a sample stochastic simulation, as described above, from the given initial conditions is shown. Which of these PRNs are useful and/or interesting is left as an exercise to the reader.

(S, D), (D, D), (D, T), (T, T), and $(T, \dashv)$ allow for one or more D then one or more T, while (S, T) and $(D, \dashv)$ allow the possibility of 0 D's or 0 T's, respectively. Thus the set of possible polymers is, as claimed in Figure 3.4, represented by the regular expression $SD^*T^* \mid D_f \mid T_f$.

The reaction schemata then correspond to the above description of the system's behavior; for example, $*_1 + T_f \to *_1T$ is a free T_f attaching to the right edge of a polymer, while $*_1D \to *_1 + D_f$ is a D falling off the right edge. Recall that in enumerating reactions, a string can only be substituted for a given wildcard if doing so respects ρ *in both the reactants and the products.* Thus in the reaction schema $*_1 + T_f \to *_1T$, while for example $*_1 = D_f$ would make valid reactants $D_f + T_f$, the product D_fT does not respect ρ, and $D_f + T_f \to D_fT$ is not a reaction in this PRN. (This reaction schema is the only one in this figure for which this consideration is relevant. Replacing $*_1 + T_f \to *_1T$ with either the one schema $S *_1 + T_f \to S *_1 T$ or the three schemata $S + T_f \to ST$, $*_1D + T_f \to *_1DT$, and $*_1T + T_f \to *_1TT$ would give the same set of reactions without taking advantage of this technicality, and the generalization of this is discussed further elsewhere in this paper.)

The graph shown is from a Mathematica simulation of this PRN as discussed previously, from an initial state with 11 copies of (the length-1 polymer) S and 1000 copies of T_f. Mathematica was instructed to track the length of one individual S polymer, and the plot shows that one polymer's length over time.

Example 3.3.2 (Copy-tolerant Turing machine)**.** Shown in the middle right of Figure 3.4, this PRN has 8 monomers; a regular expression restriction $(0_l \mid 1_l)^*(q_1 \mid q_2 \mid q_3 \mid h)(0_r \mid 1_r)^*$; and 15 reaction schemata corresponding to the 6 transition rules of a particular 3-state Turing machine.

The PRN shown in Figure 3.4 simulates a particular 3-state Busy Beaver Turing machine with transition rules shown in Table 3.1. This Turing machine, from state q_1 on a blank (all-0) initial tape, halts after 14 steps with 6 1's on the tape [48]. Similarly, in the PRN the polymer q_1 will, after 14 unimolecular reactions, become the polymer $1_l1_l1_lh1_r1_r1_r$ (and any polymer with q_1 preceded by any number of 0_l and followed by any number of 0_r will have a similar trajectory).

This PRN is an instance of a general method of simulating Turing machines with linear PRNs, using unimolecular reaction schemata corresponding to

	q_1	q_2	q_3
0	q_2,1,R	q_3,0,R	q_3,1,L
1	h,1,R	q_2,1,R	q_1,1,L

Table 3.1: The transition rules of the Turing machine whose polymer implementation is shown in Figure 3.4.

the transition rules of the Turing machine; for example, the reaction schema $*_1 q_2 1_r *_2 \rightarrow *_1 1_l q_2 *_2$ corresponds to the rule "in state 2, reading 1, write 1, move right, and transition to state 2". Transition rules reading a 0 require an additional reaction schema for the right edge of the tape, assuming blank spaces are treated as 0, and transition rules moving left require multiple reaction schemata for a 0, 1, or left edge of the tape to the left of the current square; this causes the 6 transition rules of the 3-state, 2-symbol Turing machine to require 15 reaction schemata.

A state of a Turing machine tape is any number of tape symbols, followed by a Turing machine head state, followed by any number of tape symbols; for a 2-symbol 3-state Turing machine this can be described by the regular expression $(0 \mid 1)^*(q_1 \mid q_2 \mid q_3 \mid h)(0 \mid 1)^*$, but this regular expression is not local and, if we wanted to physically implement the restriction, we could not do so using only nearest-neighbor interactions. Applying the construction from Lemma 3.3.2 requires 0 and 1 to each have two monomers representing them, leading to the regular expression $(0_l \mid 1_l)^*(q_1 \mid q_2 \mid q_3 \mid h)(0_r \mid 1_r)^*$ shown in Figure 3.4. (One could imagine some creative methods to physically implement the nonlocal regular expression, such as having a q_i monomer destabilize 0 and 1 monomers to its right and left in different ways. We would argue that such creative solutions are best modeled by treating "0 destabilized by a q on its right" and "0 destabilized by a q on its left" as distinct monomers, since they would be physically distinct and have different behaviors; and this is equivalent to the $0_l, 0_r$ model.) As this regular expression is local, it can be implemented by a compatibility relation ρ containing pairs $(\vdash, 0_l)$, $(\vdash, 1_l)$; (a, b) for $a, b \in \{0_l, 1_l\}$; (a, q) for $a \in \{0_l, 1_l\}$, $q \in \{q_1, q_2, q_3, h\}$; (q, a) for $q \in \{q_1, q_2, q_3, h\}$, $a \in \{0_r, 1_r\}$; (a, b) for $a, b \in \{0_r, 1_r\}$; $(0_r, \dashv)$, and $(1_r, \dashv)$. The generalization to a Turing machine with any number of states and/or tape symbols is obvious.

We previously mentioned that well-mixed CRNs can simulate Turing machines with arbitrarily small probability of error but using species counts exponential in the space of the Turing machine [65], and provably cannot simulate Turing machines deterministically [1, 16, 47]. Polymer systems such as Qian et al.'s stack machine [57] and Cardelli et al.'s Biochemical Ground Form (BGF) [9] are already known to be able to simulate Turing machines deterministically, in some cases with no time or space slowdown. The PRN shown in Figure 3.4, if it can be implemented, has a feature that the DNA stack machine and the BGF register machine do not: because a Turing machine tape is encoded in a single polymer and its transitions are all unimolecular reactions, multiple copies of the Turing machine can coexist in the same solution without interfering with each other.

Example 3.3.3 (String copying)**.** The middle left and lower left of Figure 3.4 show respectively a one-step nonlocal and a multi-step local model of string copying with PRNs. The one-step model has monomers $\Sigma = \{A, T, G, C, P\}$, local regular expression restriction $(A \mid T \mid G \mid C)^* \mid P$, and four reaction schemata to copy, in one step catalyzed by P, any string that starts with A, T, G, or C. The local model takes a string made of monomers A, T, G, C and transcribes the corresponding string of a, t, g, c, using monomers P, s, and S to copy one monomer at a time and eventually split the two strings.

The string copying PRNs illustrate an interesting feature of wildcards using models inspired by, but not accurate to the mechanisms of, DNA and RNA polymerases. The one-step model can be thought of as an abstraction of the *result* of DNA polymerase: where P exists only as a monomer and any string over A, T, G, C is possible under e, the PRN has four reaction schemata of the form $P + x{*_1} \to P + x *_1 + x{*_1}$ for $x \in \{A, T, G, C\}$; note that given e, this implies that $*_1$ must be made of A, T, C, G. (The reaction schema $P + *_1 \to P + *_1 + *_1$ would have allowed the reaction $2P \to 3P$, which is certainly not what we wanted and is known to go to infinity in finite time.) Because when enumerating reactions from a schema, each instance of a given wildcard is substituted by the same string, the result of these schemata is to copy any string made of A, T, G, C, catalyzed by P. However, the idea that a second copy of an arbitrarily long string can be produced in one step is not physically realistic, and while this PRN may represent the *result* of DNA polymerase, it certainly does not represent its mechanism.

The local model of string copying is a more realistic PRN that accomplishes the result of RNA polymerase, i.e. given a string over A, T, G, C it creates an additional copy of the corresponding string over a, t, g, c, catalyzed by P.

(While this is a more physically realistic mechanism with the same result as RNA polymerase, it is not in fact the mechanism of RNA polymerase, because that mechanism cannot be modeled with only linear polymers.) This concept of local mechanisms as "physically realistic" in a sense that many exotic uses of wildcards are not, is formalized in the concept of single-locus PRNs in Section 3.6.

PRN Bisimulation

Because a PRN is an infinite CRN, we can extend the definition of CRN bisimulation from CRNs to PRNs, but doing so requires an infinite interpretation. To finitely express an infinite interpretation, we build an interpretation of species from an interpretation of monomers. The obvious thing to do is to have the interpretation of a polymer be the concatenation of interpretations of its monomers, but that would not allow interpreting one implementation polymer as a multiset of formal polymers (as is possible in the finite case). We therefore require that our interpretation be built from two finite functions, μ and π, defined on the implementation monomers. Here $\pi(x)$ is the contribution of the monomer x to the polymer it is contained in and $\mu(x)$ is a multiset of additional, free-floating species represented by x. We sometimes say that *x polymerizes as* $\pi(x)$ and *carries* $\mu(x)$. Because in PRNs every species is thought of as a polymer, even monomers that never "polymerize", in such cases we will typically encode the interpretation in π and leave μ empty.

Definition 3.3.9. Given a formal PRN (Σ, e, Ψ) and implementation PRN (Σ', e', Ψ'), a *polymer interpretation* is a pair (π, μ) of functions $\pi : \Sigma' \to (\Sigma \cup \{+\})^*$ and $\mu : \Sigma' \to \mathbb{N}^{\mathcal{S}}$. These functions *induce* an interpretation $m : \mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ defined by

$$m(x_1 \dots x_n) = \pi(x_1) \dots \pi(x_n) + \sum_{i=1}^{n} \mu(x_i).$$

The symbol $+$ is interpreted as breaking a polymer, matching the notation for separate CRN species: if the π-interpretation of a polymer reads as e.g. $AB + CD$, then it is interpreted as separate species AB and CD (plus its μ-interpretation). For example, if $\pi(x) = AB + CD$ and $\mu(x) = EF + 2GH$ then $m(xx) = AB + CDAB + CD + 2EF + 4GH$. If an implementation species x carries nothing, $\mu(x) = \emptyset$, and if it polymerizes as nothing, $\pi(x) = \varepsilon$, the empty string. Note that an empty-string polymer is not considered to be a species,

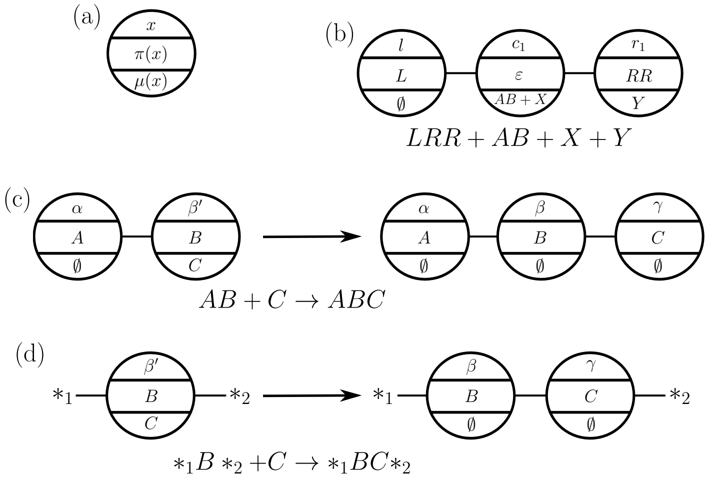


Figure 3.5: Features of a polymer interpretation. An implementation monomer x has a pair of interpretations $\pi(x)$ and $\mu(x)$ (a). An implementation polymer (e.g. lc_1r_1) is interpreted by concatenating π-interpretations and adding μ-interpretations (b). Given an interpretation of species, interpretations of states and reactions (c) follow as in CRN bisimulation [40], Definitions 3.3.2 and 3.3.3. Intuitively, interpretations of reaction schemata can follow from interpretations of monomers (d, see also Theorem 3.3.3), but this does not always work as expected.

so if all $\pi(x_i) = \varepsilon$ then $m(x_1 \ldots x_n) = \sum_{i=1}^{n} \mu(x_i)$. As in CRN bisimulation it is possible for a given $m(x_1 \ldots x_n) = \emptyset$, in this case if all $\pi(x_i) = \varepsilon$ and all $\mu(x_i) = \emptyset$.

If this interpretation preserves validity of species (which is not necessarily true if the compatibility relations are incompatible), adapting the definition of bisimulation is straightforward with one snag. Consider an interpretation where $\pi(x) = X$ and $\mu(x) = Y$ while $\pi(z) = Z$ and $\mu(z) = \emptyset$, in which case the reaction scheme $*_1x*_2 \to *_1z*_2$ intuitively should be interpreted as $*_1X *_2 + Y \to *_1Z*_2$. However, substituting x for $*_1$ and ε for $*_2$ yields $xx \to xz$, which would be interpreted as $XX + 2Y \to XZ + Y$, which cannot be obtained by substituting any two strings into the given formal reaction scheme. Avoiding this requires using the spurious-catalyst extension of CRN bisimulation from [40], in which an implementation reaction whose interpretation has catalysts can be labeled as a formal reaction without some or all of those catalysts. Naturally, we assume any species present in μ of some monomer in a wildcard are spurious catalysts.

An interpretation can "preserve validity of species" in either of two ways. The obvious way is if $x \in \mathcal{S}'$ guarantees $m(x) \subset \mathcal{S}$:

Definition 3.3.10. Let (Σ, e) and (Σ', e') be a formal and implementation species specification, with π-interpretation $\pi : \Sigma' \to (\Sigma \cup \{+\})^*$. Introduce notation $e^+ = \varepsilon \cup (e(+e)^*)$, i.e. a string matches e^+ if it is empty or a +-separated list of strings that each match e. We say that π satisfies the *Compatibility Condition* if $x_1 \dots x_n$ matching e' implies $\pi(x_1) \dots \pi(x_n) \in L(e^+)$.

In this case, given that any μ by assumption is a function $\Sigma' \to \mathbb{N}^{\mathcal{S}}$, the induced m will in fact be a CRN interpretation $\mathcal{S}' \to \mathbb{N}^{\mathcal{S}}$ as desired, and asking whether m is a CRN bisimulation is well-defined.

If π does not satisfy the compatibility condition, the structure of the implementation reaction schemata may still make the system well-behaved. For example, consider the proposed DNA implementation of Qian and Winfree's Surface CRNs [56] in the one-dimensional case as an implementation of a given PRN. The DNA implementation has no restrictions on which signals can be adjacent, so physically $e' = (\Sigma')^*$. If e is nontrivial and π maps onto all formal monomers, there will be stable implementation molecules interpreted as formally invalid polymers. However, an implementation could be designed such that any reaction involving only valid polymers will produce only valid polymers; thus invalid states are not reachable from valid initial states. We capture this concept as follows:

Definition 3.3.11. Let (Σ, e) be a formal species specification and (Σ', e', Ψ') an implementation PRN, with π-interpretation $\pi : \Sigma' \to (\Sigma \cup \{+\})$. We say that π satisfies the *Consistency Condition* if, for any reaction $R' \to P'$ enumerated from a schema in Ψ, if all $x \in R'$ has $\pi(x) \in L(e^+)$ then all $y \in P'$ has $\pi(y) \in L(e^+)$.

If (π, μ) is a polymer interpretation where π satisfies the consistency condition, then let $\mathcal{S}'_0 = \{x \in \mathcal{S}' \mid \pi(x) \in L(e^+)\}$. Naturally, m restricted to $\mathcal{S}'_0$ will be a function $\mathcal{S}'_0 \to \mathbb{N}^{\mathcal{S}}$; the consistency condition implies that "restricting" the enumerated implementation CRN to $\mathcal{S}'_0$ is well-defined. That is, the CRN $(\mathcal{S}'_0, \mathcal{R}'_0)$ where $\mathcal{R}'_0$ is the set of reactions with all reactants and products in $\mathcal{S}'_0$ contains every reaction with all reactants in $\mathcal{S}'_0$. Then m is a CRN interpretation from that CRN to the enumerated formal CRN, and asking whether m is a CRN bisimulation is well-defined.

Definition 3.3.12. Let (Σ, e, Ψ) and (Σ', e', Ψ') be a formal and implementation PRN, with polymer interpretation (π, μ) and induced CRN interpretation m. Let $\mathcal{S} = \mathcal{S}(\Sigma, e)$ and $\mathcal{S}' = \mathcal{S}(\Sigma', e')$.

We say (π, μ) is a *PRN bisimulation* if π satisfies the compatibility condition and m is a CRN bisimulation. We say (π, μ) is a *PRN bisimulation up to reachability* (from valid initial states) if π satisfies the consistency condition and m (restricted to $\mathcal{S}'_0$ as defined above) is a CRN bisimulation.

In our previous work we proved that CRN bisimulation was equivalent to up-to-interpretation trajectory equivalence [40], and that result holds for infinite CRNs and thus for PRNs. Because of this, we use CRN bisimulation (with either the compatibility condition or the consistency condition) as the definition of PRN bisimulation, despite the fact that the atomic, delimiting, and permissive conditions now each refer to an infinite set of objects. We could, instead, have defined similar conditions on the polymer structure of a PRN, and showed that those conditions imply the three conditions of CRN bisimulation, just as a polymer interpretation induces a CRN interpretation. Such conditions would capture most of the typical PRN implementations, while missing some edge cases that nevertheless satisfy CRN bisimulation. Although not the definition of PRN bisimulation, one such set of sufficient conditions is useful for proving that common implementations satisfy PRN bisimulation.

Theorem 3.3.3. *Let (Σ, e, Ψ) and (Σ', e', Ψ') be a formal and implementation PRN with polymer interpretation (π, μ). Assume either π satisfies the compatibility condition and m is the induced CRN interpretation, or the system satisfies the consistency condition and m is the CRN interpretation restricted to formally valid species and reactions. If (π, μ) satisfies the following three conditions, then m is a CRN bisimulation (and thus a PRN bisimulation or PRN bisimulation up to reachability depending on whether it satisfies the compatibility or consistency condition):*

1. Polymer Atomic Condition*: For each formal monomer X there is a canonical implementation monomer x_0 with $\pi(x_0) = X$ and $\mu(x_0) = \emptyset$. e and e' are local and equivalent to compatibility relations ρ and ρ' respectively, where for all formal monomers X, Y with canonical implementation monomers x_0, y_0 respectively, $(X, Y) \in \rho \Rightarrow (x_0, y_0) \in \rho'$ (including $X = \vdash = x_0$ or $Y = \dashv = y_0$).*

2. Polymer Delimiting Condition*: For each reaction schema in the implementation PRN, each wildcard appears the same number of times in the products as it does in the reactants, and syntactically replacing each non-wildcard monomer with its π and μ either produces equal expressions for the reactants and products or produces a formal reaction schema.*

3. m as a CRN interpretation satisfies the permissive condition.

Proof. The polymer atomic condition implies the atomic condition: any formal polymer can be built up from its corresponding implementation monomers. The polymer delimiting condition implies the delimiting condition: any implementation reaction enumerated from a schema will be interpreted as trivial or as a formal reaction enumerated from the "syntactically interpreted" formal reaction schema. (This last statement requires either the compatibility condition to imply that the resulting formal reaction is valid, or the consistency condition for an implementation reaction in the restricted subsystem to imply the same.) □

Note that the above conditions are sufficient, but not necessary, for PRN bisimulation. In some sense they describe a "natural" or "polymer" way to satisfy the conditions of PRN bisimulation. However, a pair of PRNs with a compatible or consistent interpretation may happen to satisfy the conditions of CRN bisimulation, and thus PRN bisimulation, without satisfying the stronger polymer conditions.

3.4 Verifying the DNA Stack Machine

We show the use of the Polymer Reaction Network model, and PRN bisimulation, by analyzing an existing DNA strand displacement system that uses polymers. Specifically, we analyze the system proposed by Qian et al. to implement arbitrary stack machines using DNA strand displacement [57]. This system uses a reversible addition primitive to add units representing stack symbols onto a growing stack, and uses a systematic CRN implementation for state transitions. The reversible addition primitive can grow polymers of unbounded length (as desired for a stack machine), and thus the system cannot be modeled as a Chemical Reaction Network. Modelling the DNA stack machine as a Polymer Reaction Network allows us to check whether the strand displacement system is a correct bisimulation of an abstract stack machine.

We show that the obvious interpretation on the DNA stack machine, with a correction for irreversible reactions, is a bisimulation between the DNA strand displacement system and the set of abstract reactions discussed in the original stack machine paper.

To prove that two systems are (or are not) equivalent using PRN bisimulation, we need to find an interpretation (or consider all potential interpretations for the negative case), check the compatibility or consistency condition (the stack machine as we model it will satisfy the compatibility condition), then check the atomic, delimiting, and permissive conditions. All of that assumes the two systems are a formal and implementation PRN; if not, we need to model each system as a linear PRN. For the stack machine, the formal system is a linear PRN and the implementation a DSD system; we use reaction enumeration as described below to describe it as an implementation PRN. To take advantage of the modularity condition from our previous work [40], we will add an additional step of dividing both systems into modules before checking the three conditions of CRN bisimulation. Thus the steps are as follows: enumerate the reaction schemata of the DSD system as an implementation PRN; construct an interpretation; check the compatibility condition; modularize; check the atomic, delimiting, and permissive conditions for each module.

When enumerating a DSD system without polymers as a CRN, every new DNA complex is a new species in the CRN. With polymers, on the other hand, most DNA complexes are polymers made out of monomer subunits, which requires identifying which patterns of DNA strands are the monomers. The naive approach, of having each strand be a monomer, would work with branched polymers, but fails in our linear polymer model given strands with more than two binding domains. We do not currently have a way to automate this. Therefore we describe below, and in Figures 3.6 and 3.7, which DNA complexes we choose as monomers in Σ', after which e' is a local regular expression generated from a ρ' where monomer complexes can bind if they have complementary long domains and Ψ' is determined by the enumerated set of strand displacement reactions. Even given a set of monomers, current reaction enumeration algorithms cannot automatically enumerate polymer reaction schemata, although this is potentially a useful area for future work. Instead we given an implementation PRN below that we claim is the result of applying Peppercorn's condensed semantics [36] to the DSD stack machine

[57], and invite the reader to confirm that this is the case. For the six-state three-stack machine in Figure 4(a) of Qian et al. [57], the resulting implementation PRN is (Σ', e', Ψ') as follows:

$$\begin{aligned}
\Sigma' = \{\ & 0_1, 0_1^f, 1_1, 1_1^f, \lambda_1, \lambda_1^f, 0_2, 0_2^f, 1_2, 1_2^f, \lambda_2, \lambda_2^f, 0_3, 0_3^f, 1_3, 1_3^f, \lambda_3, \lambda_3^f,\\
& 0_1^+, 0_1^-, 1_1^+, 1_1^-, \lambda_1^+, \lambda_1^-, 0_2^+, 0_2^-, 1_2^+, 1_2^-, \lambda_2^+, \lambda_2^-, 0_3^+, 0_3^-, 1_3^+, 1_3^-, \lambda_3^+, \lambda_3^-,\\
& Q, Q_1, Q_2, Q_3, I_1^Q, I_2^Q, I_3^Q, S_1, S_2, S_3, S_4, S_5, S_6,\\
& I_1^{1012Q}, I_2^{1012Q}, I_3^{1012Q}, I_4^{1012Q}, I_1^{1114Q}, I_2^{1114Q}, I_3^{1114Q}, I_4^{1114Q},\\
& I_1^{1\lambda16}, I_2^{1\lambda16}, I_3^{1\lambda16}, I_4^{1\lambda16},\\
& I_1^{2Q302}, I_2^{2Q302}, I_3^{2Q302}, I_4^{2Q302}, I_1^{3Q103}, I_2^{3Q103}, I_3^{3Q103}, I_4^{3Q103},\\
& I_1^{4Q512}, I_2^{4Q512}, I_3^{4Q512}, I_4^{4Q512}, I_1^{5Q113}, I_2^{5Q113}, I_3^{5Q113}, I_4^{5Q113},\\
& w_1, w_2, w_{1012Q}, w_{1114Q}, w_{1\lambda16}, w_{2Q302}, w_{3Q103}, w_{4Q512}, w_{5Q113}\}
\end{aligned}$$

$$\begin{aligned}
e' = {} & \lambda_1(0_1 \mid 1_1)^*(0_1^+ \mid 0_1^- \mid 1_1^+ \mid 1_1^- \mid \varepsilon) \mid \lambda_1^+ \mid \lambda_1^- \mid 0_1^f \mid 1_1^f \mid \lambda_1^f\\
& \mid \lambda_2(0_2 \mid 1_2)^*(0_2^+ \mid 0_2^- \mid 1_2^+ \mid 1_2^- \mid \varepsilon) \mid \lambda_2^+ \mid \lambda_2^- \mid 0_2^f \mid 1_2^f \mid \lambda_2^f\\
& \mid \lambda_3(0_3 \mid 1_3)^*(0_3^+ \mid 0_3^- \mid 1_3^+ \mid 1_3^- \mid \varepsilon) \mid \lambda_3^+ \mid \lambda_3^- \mid 0_3^f \mid 1_3^f \mid \lambda_3^f\\
& \mid Q \mid Q_1 \mid Q_2 \mid Q_3 \mid I_1^Q \mid I_2^Q \mid I_3^Q \mid S_1 \mid S_2 \mid S_3 \mid S_4 \mid S_5 \mid S_6\\
& \mid I_1^{1012Q} \mid I_2^{1012Q} \mid I_3^{1012Q} \mid I_4^{1012Q} \mid I_1^{1114Q} \mid I_2^{1114Q} \mid I_3^{1114Q} \mid I_4^{1114Q}\\
& \mid I_1^{1\lambda16} \mid I_2^{1\lambda16} \mid I_3^{1\lambda16} \mid I_4^{1\lambda16}\\
& \mid I_1^{2Q302} \mid I_2^{2Q302} \mid I_3^{2Q302} \mid I_4^{2Q302} \mid I_1^{3Q103} \mid I_2^{3Q103} \mid I_3^{3Q103} \mid I_4^{3Q103}\\
& \mid I_1^{4Q512} \mid I_2^{4Q512} \mid I_3^{4Q512} \mid I_4^{4Q512} \mid I_1^{5Q113} \mid I_2^{5Q113} \mid I_3^{5Q113} \mid I_4^{5Q113}\\
& \mid w_1 \mid w_2 \mid w_{1012Q} \mid w_{1114Q} \mid w_{1\lambda16} \mid w_{2Q302} \mid w_{3Q103} \mid w_{4Q512} \mid w_{5Q113}
\end{aligned}$$

We give the reaction schemata in Ψ' in multiple groups based on their intended function. The reaction schemata that implement pushing and popping onto the stack are, for each stack $i \in \{1, 2, 3\}$ and symbol $x \in \{0, 1\}$, where λ indicates the bottom of the stack:

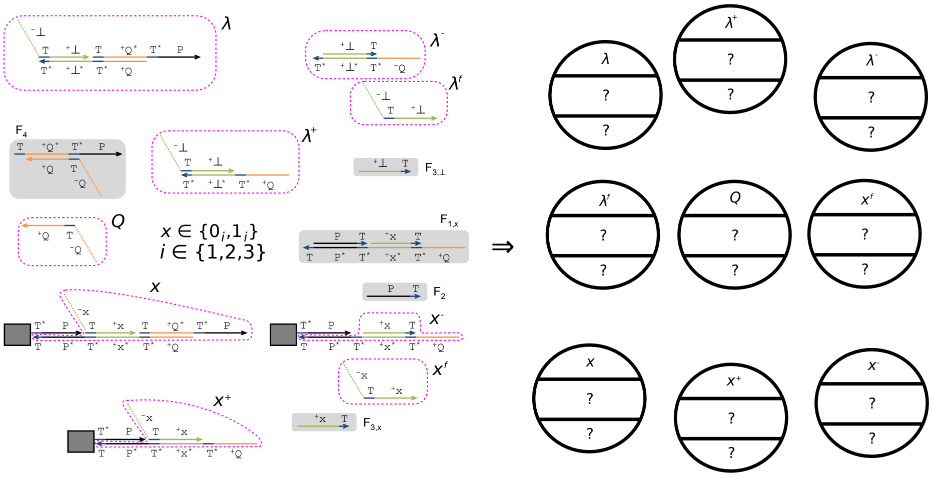


Figure 3.6: Choice of monomers in Qian et al.'s DNA stack machine [57].

$$*_1 \rightleftharpoons *_1 x_i^- \tag{3.1}$$

$$*_1 x_i^- + x_i^f \rightleftharpoons *_1 x_i^+ \tag{3.2}$$

$$*_1 x_i^+ \rightleftharpoons *_1 x_i + Q_i \tag{3.3}$$

$$\lambda_i^- + \lambda_i^f \rightleftharpoons \lambda_i^+ \tag{3.4}$$

$$\lambda_i^+ \rightleftharpoons \lambda_i + Q_i \tag{3.5}$$

For each stack i, interchangeability of Q is implemented by:

$$Q_i \rightleftharpoons I_i^Q \tag{3.6}$$

$$I_i^Q \rightleftharpoons Q \tag{3.7}$$

The irreversible stack machine transitions as shown in Figure 1 of Qian et al. [57] are incorrect according to CRN bisimulation, as discussed in our previous work: releasing the output species before the first irreversible step allows the reaction to reverse itself, producing a small probability of formally incorrect pathways [40]. As one would expect, this would be incorrect according to PRN bisimulation as well. Qian and Winfree have come up with a corrected version of the DSD mechanism (unpublished), and we give the reaction enumeration of the corrected version below. The stack machine transitions of the form

$S_i + A \rightarrow S_j + B$, where A and B are either free stack symbols x_k^f or Q (which correspond to the seven classes of I^{iAjB} monomers), are implemented by:

$$\begin{aligned} S_i &\rightleftharpoons I_1^{iAjB} && (3.8) \\ I_1^{iAjB} + A &\rightleftharpoons I_2^{iAjB} && (3.9) \\ I_2^{iAjB} &\rightarrow I_3^{iAjB} + S_j + w_1 && (3.10) \\ I_3^{iAjB} &\rightleftharpoons I_4^{iAjB} + B && (3.11) \\ I_3^{iAjB} &\rightarrow w_{iAjB} + w_2 && (3.12) \end{aligned}$$

The formal PRN that describes the stack machine is given in Figure 4(d) of Qian et al. [57]. Adapted to our notation, the formal PRN is (Σ, e, Ψ) as follows:

$$\begin{aligned} \Sigma = \{ \; & 0_1, 0_1^f, 1_1, 1_1^f, \lambda_1, \lambda_1^f, \lambda_1^-, 0_2, 0_2^f, 1_2, 1_2^f, \lambda_2, \lambda_2^f, \lambda_2^-, 0_3, 0_3^f, 1_3, 1_3^f, \lambda_3, \lambda_3^f, \lambda_3^-, \\ & Q, Q_1, Q_2, Q_3, S_1, S_2, S_3, S_4, S_5, S_6 \} \end{aligned}$$

$$\begin{aligned} e = \; & \lambda_1(0_1 \mid 1_1)^* \mid \lambda_2(0_2 \mid 1_2)^* \mid \lambda_3(0_3 \mid 1_3)^* \\ & \mid 0_1^f \mid 1_1^f \mid \lambda_1^f \mid \lambda_1^- \mid 0_2^f \mid 1_2^f \mid \lambda_2^f \mid \lambda_2^- \mid 0_3^f \mid 1_3^f \mid \lambda_3^f \mid \lambda_3^- \\ & \mid Q \mid Q_1 \mid Q_2 \mid Q_3 \mid S_1 \mid S_2 \mid S_3 \mid S_4 \mid S_5 \mid S_6 \end{aligned}$$

With Ψ containing the reaction schemata:

$$\begin{aligned}
S_1 + 0_1^f &\to S_2 + Q \\
S_1 + 1_1^f &\to S_4 + Q \\
S_1 + \lambda_1^f &\to S_6 + \lambda_1^f \\
S_2 + Q &\to S_3 + 0_2^f \\
S_3 + Q &\to S_1 + 0_3^f \\
S_4 + Q &\to S_5 + 1_2^f \\
S_5 + Q &\to S_1 + 1_3^f
\end{aligned}$$

$$Q \rightleftharpoons Q_i \mid i \in \{1,2,3\}$$

$$\begin{aligned}
*_1 + x_i^f &\rightleftharpoons *_1 x_i + Q_i \mid x \in \{0,1\}, i \in \{1,2,3\} \\
\lambda_i^- + \lambda_i^f &\rightleftharpoons \lambda_i + Q_i \mid i \in \{1,2,3\}
\end{aligned}$$

Theorem 3.4.1. *There exists a polymer interpretation (π, μ), from the implementation PRN as described above to the formal PRN as described above, that is a PRN bisimulation.*

The proof of Theorem 3.4.1, including constructing π and μ, is the remainder of this section. While the proof is given as though for this specific stack machine, it is in fact general and should apply to any instance of the DNA stack machine [57].

Given the formal and implementation PRNs, since we are proving that the implementation is correct, the next step is to construct an interpretation. For the stack machine, as with most engineered implementation systems, the rationale behind the construction suggests a natural interpretation which, if the implementation is correct at all, will be a valid PRN bisimulation. When we give this interpretation, recall the notation $m(x) = (A; B)$ as a shorthand for $\pi(x) = A$, $\mu(x) = B$ where x is an implementation monomer, A a string of formal monomers, and B a multiset of formal species. Here the natural interpretation is as follows:

1. A monomer x that appears in both the formal and implementation PRNs, such as $x = S_3$ or $x = 0_2$, has $m(x) = (x; \emptyset)$. Note that this covers all formal monomers.

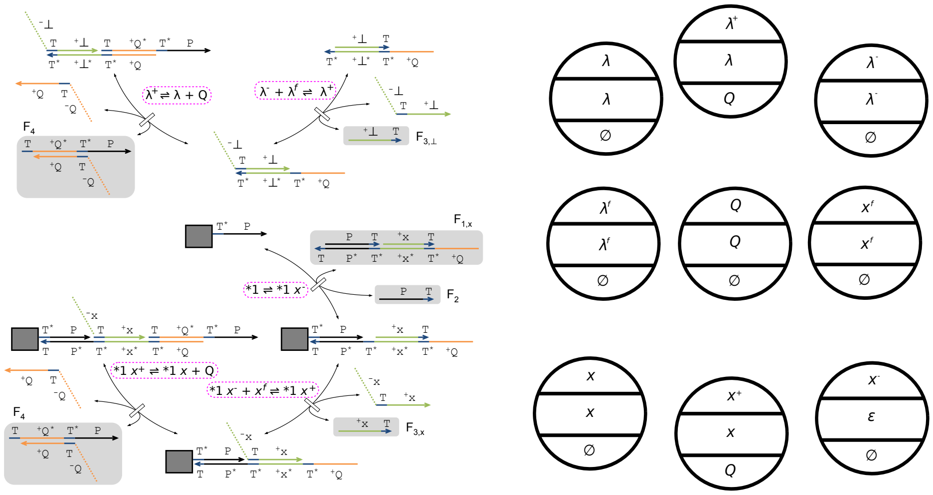


Figure 3.7: Enumeration of reaction schemata and interpretation of the monomers in Qian et al.'s DNA stack machine [57] as enumerated in Figure 3.6.

2. For monomers involved in pushing and popping from the stack, for each stack $i \in \{1, 2, 3\}$, $m(s_i^-) = (\varepsilon; \emptyset)$ for $s \in \{0, 1\}$ and $m(s_i^+) = (s_i; Q)$ for $s \in \{0, 1, \lambda\}$. (The case $m(\lambda_i^-) = (\lambda_i^-; \emptyset)$ is covered as λ_i^- is a formal monomer.)

3. For intermediates I_k^{iAjB} in the stack machine transitions, implementing $S_i + A \to S_j + B$ for $A, B \in \{x_1^f, x_2^f, x_3^f, Q\}$, we have $\pi(I_k^{iAjB}) = \varepsilon$ while $\mu(I_1^{iAjB}) = S_i$, $\mu(I_2^{iAjB}) = S_i + A$, $\mu(I_3^{iAjB}) = B$, and $\mu(I_4^{iAjB}) = \emptyset$. Similarly for the interchange of Q, $m(I_i^Q) = (\varepsilon; Q)$. Each $m(w_{\ldots}) = (\varepsilon; \emptyset)$.

Given an interpretation, we check the compatibility condition or the consistency condition to see if CRN bisimulation is even definable. In this case, the stronger, compatibility condition holds: the π-interpretation of any valid implementation species is a valid formal species or ε. Start with e', which describes all valid implementation species: 3 regular expressions describing stack polymers, and a number of species that exist as monomers (i.e., length-1 polymers). The stack polymer expressions are of the form $\lambda_i(0_i \mid 1_i)^*(0_i^+ \mid 0_i^- \mid 1_i^+ \mid 1_i^- \mid \varepsilon)$; since $\pi(0_i^+) = \pi(0_i) = 0_i$, $\pi(1_i^+) = \pi(1_i) = 1_i$, $\pi(0_i^-) = \pi(1_i^-) = \varepsilon$, and $\pi(\lambda_i) = \lambda_i$, any implementation polymer matching this expression will have its π-interpretation match the $\lambda_i(0_i \mid 1_i)^*$ subexpression of e, and thus be a valid formal polymer. The monomers are I-type species whose π-interpretation is

ε; formal species Q, Q_i, S_i, x_i^f, and λ_i^- whose π-interpretations are themselves and which appear in e as valid formal length-1 polymers; and the λ_i^+ species whose π-interpretation λ_i matches the $\lambda_i(0_i \mid 1_i)^*$ subexpression of ρ. This covers all cases in e', proving that the compatibility condition holds.

Given the compatibility condition, the induced interpretation m (see Definition 3.3.9) is in fact a CRN interpretation, and is a PRN bisimulation if and only if it is a CRN bisimulation (Definition 3.3.12). Thus the last thing we need to do is show that m satisfies the atomic, delimiting, and permissive conditions (Definition 3.3.4.II). Again, for a PRN treated as an infinite CRN, an algorithmic way of doing this is generally infeasible; in fact we will show in the next section that checking the permissive condition in the general case for linear PRNs is undecidable. To check an engineered system, one would typically rely on the intent of the designers being formalizable into a proof of correctness. The stack machine was designed in subsystems, each of which correctly implements one formal reaction or formal reaction schema, which when combined form a correct implementation of the formal PRN. The modularity condition from Theorem 3.3.2 covers exactly this case: we will only need to prove each module correct, and the correctness of the whole system will follow. Thus, the next step is to divide the enumerated formal and implementation CRNs into modules.

To specify each module we must specify a set of formal species, formal reactions, implementation species, implementation reactions, and sets of "common" formal and implementation species, each subsets of their respective sets of species. The modularity condition expects each module to have an interpretation; since we already have m defined, each module's interpretation is just m restricted to its implementation species. For each stack i, we have one module consisting of: all formal species matching $\lambda_i(0_i \mid 1_i)^* \mid \lambda_i^- \mid \lambda_i^f \mid 0_i^f \mid 1_i^f \mid Q_i$, the formal reaction $\lambda_i^- + \lambda_i^f \rightleftharpoons \lambda_i + Q_i$, and all formal reactions enumerated from the two reaction schemata $*_1 + x_i^f \rightleftharpoons *_1 x_i + Q_i \mid x \in \{0, 1\}$ (recall that a reversible reaction schema is a shorthand for two irreversible schemata); all implementation species matching $\lambda_i(0_i \mid 1_i)^*(0_i^+ \mid 0_i^- \mid 1_i^+ \mid 1_i^- \mid \varepsilon) \mid \lambda_i^+ \mid \lambda_i^- \mid 0_i^f \mid 1_i^f \mid \lambda_i^f \mid Q_i$, and all implementation reactions enumerated from reaction schemata of type (1) through (5) for stack i; and the common formal species being all formal species, while the common implementation species are all those with the same name as a formal species. For each stack i, we have a *separate* module containing formal species Q and Q_i and reactions $Q \rightleftharpoons Q_i$; implementation species Q, Q_i, and I_i^Q and reactions of type (6) and (7) for stack i; and $\{Q, Q_i\}$ is again both the set of common formal species and of common implementation species. For each formal reaction of the form $S_i + A \rightarrow S_j + B$, we have a module consisting of those formal species and that formal reaction; implementation species S_i, S_j, A, B, and all I_k^{iAjB} species, and reactions of type (8) through (11) for this formal reaction; and again, all formal species are common and all implementation species with the same name as a formal species are common. In this three-stack, six-state, seven-transition stack machine, this gives 13 modules, shown in Figure 3.8; we prove below that each of those 13 modules satisfies the atomic, delimiting, permissive, and modularity conditions.

Recall the polymer atomic and polymer delimiting conditions from Theorem 3.3.3. The argument that the whole system satisfies the atomic condition starts similarly to the polymer atomic condition: each formal monomer x has an implementation monomer with the same name and with $m(x) = (x; \emptyset)$. Then observing that e is a subexpression of e', any formal species $w \in \mathcal{S}$ (i.e., string matching e) will also match e', thus $w \in \mathcal{S}'$ and will have $m(w) = \{\!|w|\!\}$. Because each module, for each formal species it contains, also contains the implementation species with the same name, each module satisfies the atomic condition.

The whole system satisfies the polymer delimiting condition, which we prove by going through the types of implementation reaction schemata. The reader can verify that schemata of types (1) and (3), and reactions of types (5), (7), (8), (9), and (11) are all trivial (for example, type (3) is syntactically interpreted as $*_1 x_i + Q_i \rightleftharpoons *_1 x_i + Q_i$); schemata of type (2) are syntactically interpreted as $*_1 + x_i^f \rightleftharpoons *_1 x_i + Q_i$; reactions of type (4) as $\lambda_i^- + \lambda_i^f \rightleftharpoons \lambda_i + Q_i$; type (6) as $Q_i \rightleftharpoons Q$; and type (10) as the appropriate $S_i + A \rightarrow S_j + B$. All of those nontrivial syntactically interpreted reactions or schemata appear in the formal PRN, so the polymer delimiting condition is satisfied, which proves that the whole system satisfies the delimiting condition. Again, each module, for each nontrivial implementation reaction it contains, also contains the corresponding formal reaction, so each module satisfies the delimiting condition.

The permissive condition is where the division into modules matters. To prove the permissive condition we will have to check each formal reaction within each

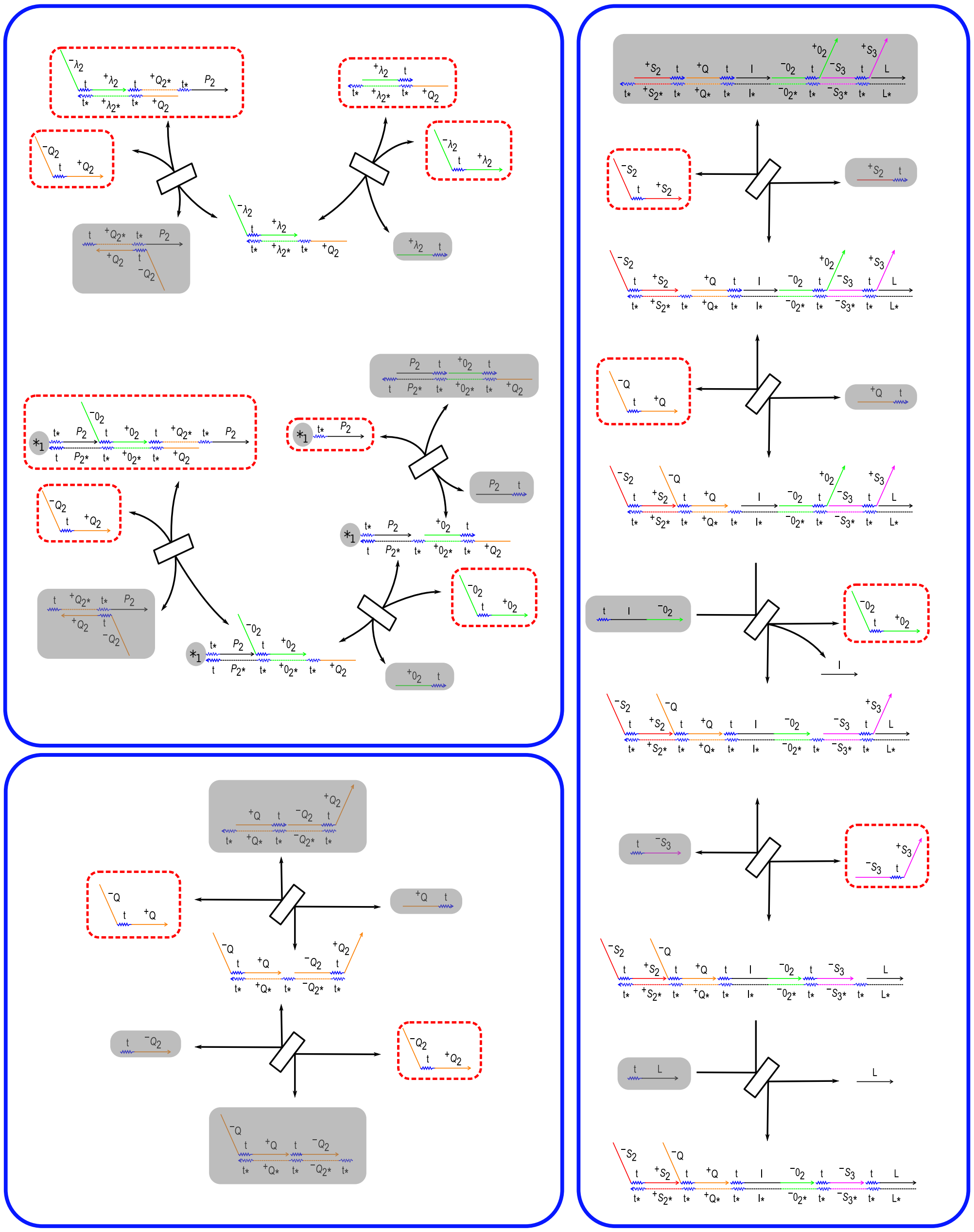


Figure 3.8: Examples of the three types of modules. Top left: the stack module for stack 2. Bottom left: the Q exchange module for stack 2. Right: the stack machine transition module for formal reaction $S_2 + Q \rightarrow S_3 + 0_2^f$. Common species are outlined in red dashed lines. DNA complexes covered in gray boxes are fuel complexes, which are not included in the implementation PRN.

module; since no formal reaction appears in multiple modules, modularity does not increase how much we have to check, and since the size of each implementation module is smaller than the whole implementation CRN, we have less to check per reaction, thus less overall. As discussed in our previous work on CRN bisimulation, we prove the permissive condition by showing that for each formal reaction, for each *minimal* implementation state whose interpretation contains all the formal reactants, that reaction can be implemented; since if every minimal implementation state can do something, then every implementation state can do the same thing [40].

To treat the simple cases first, consider the formal reactions that are not schemata (i.e. have no wildcards). A formal reaction of the form $S_i + A \to S_j + B$ appears only in its own module, in which the minimal implementation states containing $S_i + A$ in their interpretation are $\{\!|S_i + A|\!\}$, $\left\{\left|I_1^{iAjB} + A\right|\right\}$, and $\left\{\left|I_2^{iAjB}\right|\right\}$. These states implemement $S_i + A \to S_j + B$ by, respectively, forward reactions of type (8) then (9) then (10); (9) then (10); or just (10); (8) and (9) are trivial reactions followed by (10) which is interpreted as $S_i + A \to S_j + B$. As an edge case, if $A = B$ (as is the case for $S_1 + \lambda_1^f \to S_6 + \lambda_1^f$), then any of the above three states with A replaced by I_3^{iAjB} is also a minimal state. Such a state implements $S_i + A \to S_j + B$ by the forward reaction of type (11) followed by the appropriate sequence mentioned above. Similarly, the formal reactions $Q \rightleftharpoons Q_i$ each appear only in their own modules, in which the minimal states for $Q \to Q_i$ are $\{\!|Q|\!\}$ and $\left\{\left|I_i^Q\right|\right\}$, and the only minimal state for $Q_i \to Q$ is $\{\!|Q_i|\!\}$. Those three states implement the appropriate formal reaction respectively by forward (6) followed by forward (7); just forward (7); and just reverse (7).

The remaining formal reactions are the $\lambda_i^- + \lambda_i^f \rightleftharpoons \lambda_i + Q_i$ reactions and the reactions enumerated from the $*_1 + x_i^f \rightleftharpoons *_1 x_i + Q_i$ schemata, all of which exist in the stack modules. Each of the three stack modules thus contains infinitely many reactions: in stack module $i \in \{1, 2, 3\}$, for each $w \in \{0, 1\}^*$ and $x \in \{0, 1\}$, where w_i is w made up of 0_i's and 1_i's, $*_1 + x_i^f \rightleftharpoons *_1 x_i + Q_i$ enumerates a pair of reactions $\lambda_i w_i + x_i^f \rightleftharpoons \lambda_i w_i x_i + Q_i$.

There is only one minimal implementation state for $\lambda_i^- + \lambda_i^f \to \lambda_i + Q_i$, $\left\{\left|\lambda_i^- + \lambda_i^f\right|\right\}$, which implements the formal reaction by a forward reaction of type (4). $\lambda_i w_i + x_i^f \to \lambda_i w_i x_i + Q_i$ has four minimal states, namely x_i^f plus any one of $\lambda_i w_i$, $\lambda_i w_i 0_i^-$, $\lambda_i w_i 1_i^-$, $\lambda_i w_i' y_i^+$ if $w = w'y$ for $y \in \{0, 1\}$, or λ_i^+ if

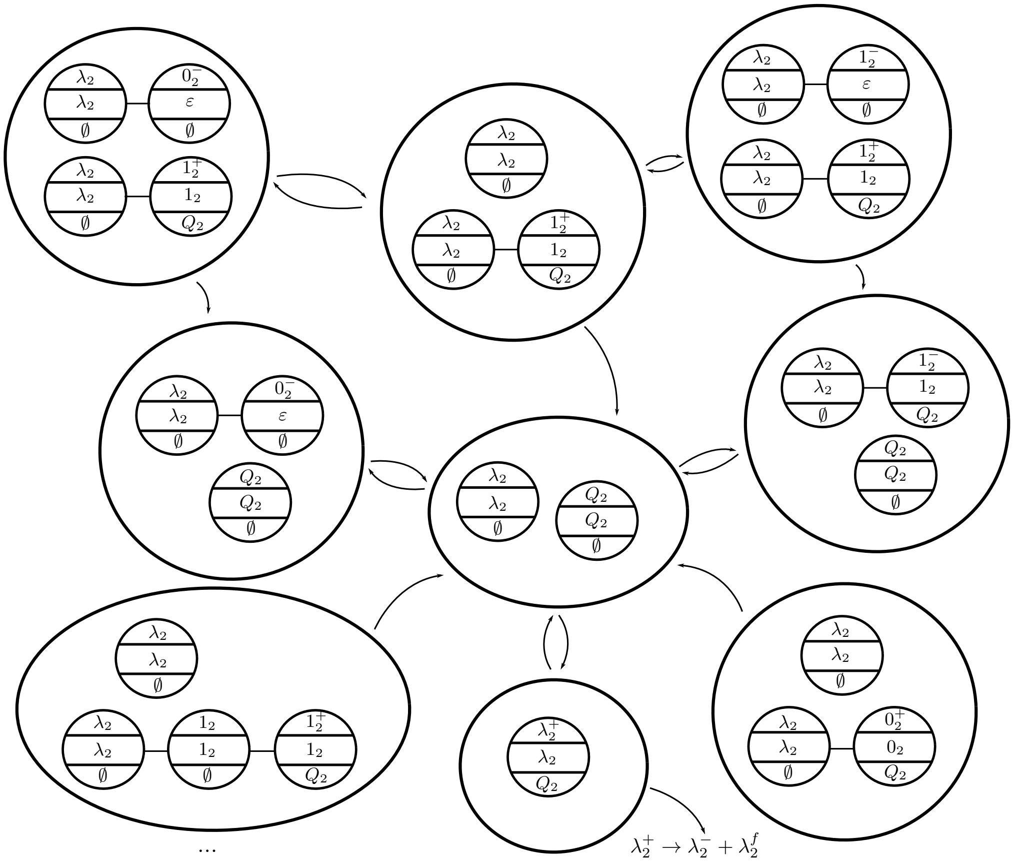


Figure 3.9: The "minimal states argument" from our previous work on CRN bisimulation [40] is often the most effective way to prove the permissive condition. Here we show some of the minimal impementation states (within the stack 2 module) in which the formal reaction $\lambda_2 + Q_2 \to \lambda_2^- + \lambda_2^f$ should be able to occur. Arrows between states represent trivial implementation reactions; the arrow with no target represents the implementation reaction $\lambda_2^+ \to \lambda_2^- + \lambda_2^f$, which is interpreted as the desired formal reaction. As in the finite CRN bisimulation case, reversible reactions from a minimal state to a non-minimal state may be shown as irreversible arrows between minimal states, e.g. $*_1 1_2^+ \rightleftharpoons *_1 1_2 + Q_2$ taking $\lambda_2 + \lambda_2 1_2 1_2^+$ to (a non-minimal state containing) $\lambda_2 + Q_2$. Unlike the finite CRN case, here we have infinitely many minimal states (for example, every state of the form $\lambda_2 + \lambda_2 w 1_2^+$, $w \in \{0_2, 1_2\}^*$), so the permissive condition cannot be verified by simply checking for paths in this graph; however, the argument given in the text based on this graph can prove it.

$w = \varepsilon$. These states implement the formal reaction as follows: $\lambda_i w'_i y_i^+$ becomes $\lambda_i w_i + Q_i$ by a reaction of type (3); λ_i^+ becomes $\lambda_i + Q_i$ by a reaction of type (5); $\lambda_i w_i (1-x)_i^-$ becomes $\lambda_i w_i$ by a reverse reaction of type (1); $\lambda_i w_i$ becomes $\lambda_i w_i x_i^-$ by a forward reaction of type (1), all of the so-far-mentioned reactions being trivial; and $\lambda_i w_i x_i^- + x_i^f$ implements the formal reaction by a reaction of type (2). The reverse reactions are slightly more complex, because the Q_i can be provided by any implementation species in the module whose interpretation *contains* Q_i, namely Q_i itself, λ_i^+, or any $\lambda_i u_i y_i^+$ for $u \in \{0,1\}^*$, $y \in \{0,1\}$. The minimal states for $\lambda_i + Q_i \to \lambda_i^- + \lambda_i^f$ are then either λ_i^+ by itself, or one of λ_i, $\lambda_i 0_i^-$, or $\lambda_i 1_i^-$ plus one of any species providing Q_i (other than λ_i^+, in which case the state would not be minimal). Similarly, the minimal states for $\lambda_i w_i x_i + Q_i \to \lambda_i w_i + x_i^f$ are either $\lambda_i w_i x_i^+$ by itself, or one of $\lambda_i w_i x_i$, $\lambda_i w_i x_i 0_i^-$, or $\lambda_i w_i x_i 1_i^-$ plus one of any non-$\lambda_i w_i x_i^+$ species providing Q_i. These states implement the formal reaction as follows: any species providing Q_i releases the implementation Q_i by a reaction of type (3) or (5) as appropriate; any 0_i^- or 1_i^- "falls off" by a reverse reaction of type (1); free Q_i joins λ_i by a reverse reaction of type (5) or $\lambda_i w_i x_i$ by a reverse reaction of type (3); and finally the formal reaction is implemented by a reverse reaction of type (2) or (4) as appropriate. This covers all modules, and proves that within each module, the permissive condition is satisfied; but does not prove that the permissive condition is satisfied for the whole system.

From our previous work, the modularity theorem proves that, if each module satisfies the permissive condition and the modularity condition, then the whole system satisfies the permissive condition (and the modularity condition) [40]. So the last step is to prove that each module satisfies the modularity condition: each implementation species can "decompose", via trivial reactions, into one multiset of common implementation species and another multiset of implementation species whose interpretation contains no common formal species. For common implementation species, and for implementation species containing no common formal species, this decomposition is already done. The non-common implementation species are the I_k^{iAjB} intermediates, I_i^Q intermediates, λ_i^+, λ_i^-, λ_i^f and x_i^f monomers, and stack polymers matching $\lambda_i(0_i \mid 1_i)^*(0_i^+ \mid 0_i^- \mid 1_i^+ \mid 1_i^- \mid \varepsilon)$, but of those only I_k^{iAjB} for $k \neq 4$, I_i^Q, λ_i^+, and species matching $\lambda_i(0_i \mid 1_i)^*(0_i^+ \mid 1_i^+)$ contain common formal species. Each of those species decomposes as follows: I_1^{iAjB} to S_i via reverse reaction (8); I_2^{iAjB} to $A + S_i$ via reverse reactions (9) and (8); I_3^{iAjB} to $I_4^{iAjB} + B$ via reaction

(11); I_i^Q to Q via reaction (7); λ_i^+ to $\lambda_i + Q_i$ via reaction (5); and a species of the form $\lambda_i w_i x_i^+$ to $\lambda_i w_i x_i + Q_i$ via a reaction enumerated from reaction schema (3). This satisfies the modularity condition, meaning that the permissive condition will be satisfied when the initial implementation state combines species from different modules. This completes the proof that the given (π, μ) is a PRN bisimulation from the DNA stack machine to its formal description, which implies that the two systems will have the same set of trajectories from any initial state. Since it is intuitive that the formal system, when started in the appropriate initial states, simulates an abstract stack machine, so does the DNA system.

3.5 Hardness Results

Having defined a concept of correctness of an implementation of a polymer network, we would like to be able to algorithmically check, given two polymer networks and an interpretation, whether that interpretation is a bisimulation. Knowing that polymer networks are capable of Turing-universal computation, we might suspect that to be impossible. A next best thing would be if bisimulation or non-bisimulation was recursively enumerable: either that any correct interpretation would have a proof of correctness, or that any incorrect interpretation would have a proof of incorrectness. Unfortunately, neither one is the case. We show that verifying our notion of bisimulation for PRNs is equivalent to the uniform halting problem, which given a Turing machine, asks if every possible configuration of the Turing machine will eventually lead to a halting configuration [37]. This problem is in the class Π_2^0, the complement of the second level of the arithmetic hierarchy, which is the class of all languages $L = \{x \mid \forall_y \exists_z \phi(x, y, z)\}$, where ϕ is a decidable predicate. Since each level of the arithmetic hierarchy strictly contains the previous levels, a Π_2^0-complete problem cannot be recursively enumerable, nor can its complement [43]. Since the uniform halting problem is Π_2^0-complete [37], so is PRN bisimulation. It is also interesting to note that the atomic condition, which is trivial to check for finite CRNs, becomes PSPACE-complete for Polymer Reaction Networks, proven by reduction from the problem of checking whether a regular expression describes the language of all strings [51, 68].

Lemma 3.5.1. *Given a formal species schema (Σ, e), implementation species schema (Σ', e'), and interpretation (π, μ), the problem of checking the atomic condition and that of checking the compatibility condition are both PSPACE-*

complete. If one or both are required to be local, or equivalently, given as a compatibility relation (Σ, ρ) *and/or* (Σ', ρ')*, then the atomic condition remains PSPACE-complete. The compatibility condition does not depend on the implementation schema; it is PSPACE-complete if the formal schema is given as a regular expression, but can be checked in polynomial time if the formal schema is known to be local.*

Proof. The problem of, given a pair of regular expressions (e_1, e_2) over Σ, deciding whether the language of e_1 is contained in that of e_2 is PSPACE-complete. To check the atomic condition in polynomial space, $e_1 = e(+e)^*$ and e_2 is the expression of π-interpretations of strings matching e' using only monomers $x \in \Sigma'$ with $\mu(x) = \emptyset$, union with the (finite) set of all A such that some x_0 has $\pi(x_0) = \varepsilon$ and $\mu(x_0) = \{\!|A|\!\}$, and $ux_0v \in L(e')$ where all $x \in uv$ has $m(x) = (\varepsilon, \emptyset)$. Similarly to check the compatibility condition, $e_2 = e$ and e_1 is the expression of π-interpretations of strings matching e', this time regardless of their μ-interpretations. Completeness for both conditions uses $\Sigma' = \Sigma$, $m(x) = (x, \emptyset)$, and e and e' are e_1 and e_2 respectively for the atomic condition and in the reverse order for the compatibility condition.

If one or both species schemata are given as local regular expressions or compatibility relations, then we recall Lemma 3.3.2, that given any regular expression there is a compatibility relation on an implementation monomer set and a π-interpretation under which they allow the same set of strings. Here we use that deciding whether a regular expression e matches all strings over Σ is also PSPACE-complete. So the atomic condition, given formal species schema (Σ, ρ) with $\rho = \Sigma_\epsilon \times \Sigma_\epsilon$ (allowing all strings) and implementation schema (Σ', ρ') and π-interpretation implementing e according to Lemma 3.3.2, is true if and only if e matches all of Σ^*. The compatibility condition, when the formal schema is allowed to be nonlocal, is true for (Σ, Σ^*) implementing (Σ, e) if and only if e matches Σ^*. When the formal schema is given with a compatibility relation (Σ, ρ), and the implementation schema is (Σ', e') which may be local or not, define ρ'_ε such that $x\rho'_\varepsilon y$ if any $uxvyw \in \mathcal{S}(\Sigma', e')$, $u, v, w \in (\Sigma')^*$, with $\pi(v) = \varepsilon$; the cases $x = \epsilon$ and $y = \epsilon$ correspond to $vyw \in \mathcal{S}(\Sigma', e')$ and $uxv \in \mathcal{S}(\Sigma', e')$, respectively, with the same restrictions on u, v, w. This can be computed in polynomial time with reachability questions on the nondeterministic finite automaton associated with e'. It is in general not true that $\mathcal{S}(\Sigma', e') = \mathcal{S}(\Sigma', \rho'_\varepsilon)$, but where $\pi(x)_1$ and $\pi(x)_{-1}$ are the first and last characters, respectively, of $\pi(x)$, it is true that the compatibility condition is true if and only if $x\rho'_{\varepsilon}y \Rightarrow \pi(x)_{-1}\rho\pi(y)_1$, with the convention $\pi(\epsilon)_1 = \pi(\epsilon)_{-1} = \epsilon$. □

Theorem 3.5.1. *The problem of, given a formal PRN* (Σ, e, Ψ)*, implementation PRN* (Σ', e', Ψ')*, and interpretation* (π, μ)*, deciding whether that interpretation is a bisimulation is* Π^0_2*-complete.*

Proof. Weak bisimulation is the statement that *for all* pairs of related states and steps in one of the two states *there exists* a corresponding sequence of steps in the other state, which is naturally a Π^0_2 statement. (In PRN bisimulation this description applies to both the delimiting and permissive conditions, while the atomic condition is decidable in PSPACE by Lemma 3.5.1.) To prove completeness, we reduce from the uniform halting problem: given a Turing machine, is is true that from any combination of state and tape contents, the Turing machine halts? Since PRNs can simulate Turing machines, we show that the condition that, for all states of a PRN, a given reaction can happen is equivalent to the condition that, for all configurations of a Turing machine, the Turing machine will halt. In the case of PRN bisimulation, the above condition corresponds to the permissive condition, in an implementation PRN where the delimiting condition is true. Since the uniform halting problem is Π^0_2-complete [37], so is bisimulation.

Given a Turing machine M with alphabet $\{0, 1, b\}$ (where b is the blank symbol), with states Q, start state q_0 and halt state q_H, we construct a pair of PRNs and an interpretation which is a bisimulation if and only if M halts from every instantaneous description (with finitely many nonblank characters). The formal PRN is $(\{Q, H\}, Q \mid H, \{Q \to H\})$. The implementation PRN is the simulation of M generalized from Example 3.3.2, simulating M using state monomers q_i and tape squares 0^l, 1^l to the left of the state, 0^r, and 1^r to the right, and between one and six reaction schemata for each transition in M. $\mu(x) = \emptyset$ for all x, $\pi(0^l) = \pi(0^r) = \pi(1^l) = \pi(1^r) = \varepsilon$, $\pi(q_i) = Q$ for each non-halting state q_i, and $\pi(q_H) = H$.

Given $e' = (0^l \mid 1^l)^*(q_i)(0^r \mid 1^r)^*$ from the generalized Example 3.3.2, the valid implementation polymers are exactly the valid instantaneous descriptions of M, and the only reactions that can happen are simulations of steps of M. Any valid implementation species has only one state q_i, and thus interprets to either Q or H, both of which are valid formal species, which also satisfies the atomic condition. Any implementation reaction is a transition of M, so the corresponding formal step is either trivial, if the transition is not to q_H, or $Q \to H$ if it is, satisfying the delimiting condition. In any formal state with a Q, and any implementation state interpreted as that formal state, there is at least one non-halting instantaneous description of M, and the statement that all such states can eventually do $Q \to H$ (the permissive condition) is equivalent to the statement that all instantaneous descriptions eventually halt. □

3.6 Single-Locus Networks

Given a class of interesting Polymer Reaction Networks, we would naturally want to find a physical implementation of some or all of those networks. So far, steps taken towards implementing polymer reactions include the stack machine implementation by Qian et al. [57], and the surface CRN implementations proposed by Qian and Winfree [56]. To illustrate one challenge in implementation, recall the string copying and equality/reverse detection PRNs from Figure 3.4. For example, the one-step string copying PRN uses reaction schemata of the form

$$P + A{*_1} \to P + A *_1 + A *_1 .$$

While this schema describes the copying of an arbitrarily long string starting with A and catalyzed by P, physical systems (biological, engineered, or otherwise) tend not to copy arbitrarily long strings in one step. The local model string-copying PRN in Figure 3.4 transcribes a string of length n in $O(n)$ steps, each of which affects only a constant number of monomers (specifically, at most 3). In general, physical systems will—on the most realistic level—be modeled as such local and bounded reactions, by which we mean reactions that only “read” and “write” a finite number of monomers and/or connections between monomers.

If we try to model the local mechanism of DNA polymerase as an implementation of $P + A{*_1} \to P + A{*_1} + A{*_1}$, an immediate problem is that the structure is no longer linear, but branched. This problem is somewhat related to an issue with naively enumerating a PRN from a DNA strand displacement system: in the stack machine, for example, treating a single strand as a monomer will fail when some strands have enough domains to bind to three other strands at once. In that case, since the “third branch” never exceeded a fixed size, a clever choice of DNA complexes to be treated as implementation monomers allowed us to model the system as a linear PRN, but the same is not true for DNA polymerase. A DNA polymerase "implementation" network could be modeled in Cardelli's Biochemical Ground Form [9], or in the branching PRN extended model we discuss in Section 3.7, but not in the linear Polymer Reaction Network model. Even if we use a model with branching polymers, the implementation will not be correct according to bisimulation: in the formal network, the second copy of the arbitrarily long polymer $A{*_1}$ is produced in one step, which is impossible in an implementation network made up of only local and bounded reactions. (The network could be correct according to CRN bisimulation on the induced infinite CRNs, where for each polymer w the branched structure being built up from an initial $A{*_1} = w$ is interpreted as $w + P$ until the final dissociation step, at which point each copy of w is interpreted as w. However, PRN bisimulation would require each individual monomer to have an interpretation, preventing this workaround.)

The key obstacle here is the (so far informal) concept of "local and bounded", and the difficulty of implementing formal reaction schemata that are not "local and bounded" using only implementation schemata that are. (Or at least, the difficulty of doing so in a way bisimulation can recognize and verify.) For the moment, therefore, we will turn to implementation of reaction schemata that *are* local and bounded, with a suitable definition. We define a concept of a *single-locus reaction schema*, which we feel captures the informal concept of "local and bounded". We will show that these single-locus reaction schemata can be implemented up to bisimulation by a set of four polymer primitives, three of which have candidate DNA implementations from the stack machine [57] or surface CRNs [56]. We show that a class of infinite CRNs, which is intuitively the class of single-locus PRNs plus compatibility relation-based computational power, is closed under bisimulation and any member of that class can be implemented by the given primitives, suggesting that the concept of single-locus schemata is a natural class to discuss.

Definition 3.6.1. A reaction schema is *single-locus* if:

(i) Any wildcard that appears at all, appears exactly once in the reactants and exactly once in the products.

(ii) Wildcards appear only at the beginning or at the end of a polymer, and each wildcard that appears, appears at the same place (beginning or

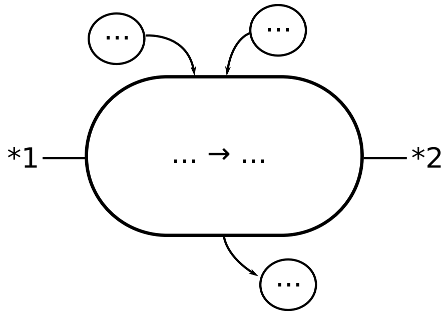


Figure 3.10: Single-locus reaction schemata are, intuitively, schemata whose reactions occur entirely within one region whose size is not affected by wildcards. This region may be in the middle of one polymer (if $*_1$ and $*_2$ are on different edges of the same polymer), at the joining of two polymers (if $*_1$ and $*_2$ are on separate polymers), or at either end of one polymer (if either $*_1$ and/or $*_2$ does not exist). The schema may consume, produce, and/or be catalyzed by any number of additional, finite polymers, since such a reaction can still be thought of as taking place within a finite region.

end) in the products as in the reactants.

(iii) No two distinct wildcards appear at the beginning of a polymer, and no two distinct wildcards appear at the end of a polymer.

(For the purpose of conditions (ii) and (iii), a wildcard that is the entire polymer can be counted as at the beginning or at the end, and the reaction schema is single-locus if it satisfies the conditions for at least one of those two choices. For example, $*_1 \rightleftharpoons *_1E$, $*_1 \rightleftharpoons E*_1$, and $*_1 + *_2 \rightleftharpoons *_1I*_2$ are all single-locus.)

A PRN (Σ, e, Ψ) is single-locus if e is local and each reaction schema in Ψ is single-locus. An augmented PRN (Σ, e, Ψ) is *augmented single-locus* if each reaction schema in Ψ, ignoring its regular expression restrictions, is single-locus; here e is not required to be local.

Intuitively, a reaction schema is single-locus if it only "reads from" (is conditional on) and "writes to" (changes) one region of finite size, i.e. containing no wildcards. $*_1AB*_2 \to *_1CDE*_2$ is the ideal example of this; it "reads" AB and "writes" CDE, a region of size at most 3, while leaving $*_1$ and $*_2$ unchanged. Similarly, in $*_1A *_2 + B \to *_1C*_2$, the "region" includes both the region A on the polymer $*_1A*_2$ and the free monomer B, which since the monomer B has no wildcards is still a finite size. A reaction schema $*_1 + A*_1 \rightarrow *_1 + B*_1$ would *not* be single-locus, since to check that both polymers have the same sequence substituted for the wildcard $*_1$ requires reading that sequence, and a wildcard's sequence is not bounded by a finite size. Similarly, $*_1 A *_2 B *_3 \rightarrow *_1 C *_2 D *_3$ requires reading both the A and B and writing both C and D, which thanks to the intervening $*_2$ cannot be included in one region of finite size. (These examples correspond to violations of conditions (i) and (ii), respectively.) A schema such as $*_1 A *_2 + *_3 B *_4 \rightarrow *_1 C *_2 + *_3 D *_4$ is not single-locus by the above definition because it violates condition (iii), even though the A and B could be viewed as a finite region to read from and write to. To do so, however, we would have to view the A-B region as a single region on a branched polymer, and for the same reason, implementing this reaction schema (up to PRN bisimulation) with "physically possible" (i.e., single-locus) reactions of *linear* polymers is impossible. We will, however, return to this topic in Section 3.7. An augmented single-locus PRN is not exactly local, and reaction schemata may read (but not change) unbounded regions; however, it turns out that augmented single-locus PRNs are a natural class of PRNs closed under PRN bisimulation.

Theorem 3.6.1. *For any formal single-locus PRN* (Σ, e, Ψ)*, there is an implementation PRN* $(\Sigma', (\Sigma')^*, \Psi')$ *and bisimulation up to reachability interpretation* (π, μ) *such that all reaction schemata in* Ψ' *are of one of the following four forms:*

$$*_1 AB *_2 \rightarrow *_1 CD *_2 \qquad \text{(Context-sensitive Replacement)}$$

$$*_1 A *_2 + B \rightleftharpoons *_1 C *_2 \qquad \text{(Monomer-dependent Replacement)}$$

$$*_1 \rightleftharpoons *_1 E, *_1 \rightleftharpoons F *_1 \qquad \text{(Reversible Addition)}$$

$$*_1 + *_2 \rightleftharpoons *_1 I *_2 \qquad \text{(Reversible End-joining)}$$

Proof. We will later show how to implement any reaction schema of the form

$$*_1 x_1 \ldots x_n *_2 + r_1 + \cdots + r_k \rightarrow *_1 y_1 \ldots y_m *_2 + p_1 + \cdots + p_l,$$

where x_i and y_i are monomers, $r_i = r_{i,1} \ldots r_{i,n_i}$ and similarly p_i are strings of monomers. We can use such an implementation to implement the remaining classes of single-locus schema, as follows. Reactants or products of the form $*_1 w_1 + w_2 *_2$ (here w_i are strings of monomers) can be replaced by $*_1 w_1 I w_2 *_2$ together with the reaction schema $*_1 + *_2 \rightleftharpoons *_1 I *_2$, where $m(I) = (+; \emptyset)$.

Reactants or products without $*_1$ (resp. $*_2$) can replace $w*_2$ with $*_1F_Lw*_2$ with the reaction schema $*_1 \rightleftharpoons F_L*_1$ (resp. replace $*_1w$ with $*_1wF_R*_2$ and add the reaction schema $*_1 \rightleftharpoons *_1F_R$), where $m(F_L) = m(F_R) = (\varepsilon; \emptyset)$. This argument implicitly makes use of the transitivity property of CRN bisimulation [40], which applies equally well to infinite CRNs and thus to PRNs. For example, it is simple to confirm that (for any reasonable Σ, ρ, Σ', ρ') $\{*_1AIB*_2 \rightarrow *_1C*_2, *_1 + *_2 \rightleftharpoons *_1I*_2\}$ is a correct (up to PRN bisimulation) implementation of $\{*_1A + B*_2 \rightarrow *_1C*_2\}$, so by transitivity, any correct implementation of the former PRN will be a correct implementation of the latter. We also assume that $\{x \mid (x, x_1) \in \rho\} = \{x \mid (x, y_1) \in \rho\}$, and similarly $\{x \mid (x_n, x) \in \rho\} = \{x \mid (y_m, x) \in \rho\}$; if this is not the case, we can replace this schema with multiple schemata of the form $*_1x_0x_1 \dots x_nx_{-1} *_2 + \dots \rightarrow *_1x_0y_1 \dots y_mx_{-1} *_2 + \dots$ for every possible x_0 and x_{-1} (and consider each separately), each of which trivially satisfies the condition. Such a replacement will again be a correct implementation up to bisimulation of the original single schema, and again transitivity applies. Given an implementation of each reaction schema in a formal PRN, combining the implementation reaction schemata will produce a correct implementation of the formal PRN; this relies on the modularity property of CRN bisimulation [40], and in fact the implementation we give will satisfy the condition for modularity to hold.

Given a formal reaction schema of the above form, we can implement it as follows: use $*_1AB*_2 \rightleftharpoons *_1CE*_2$ and $*_1A *_2 + B \rightleftharpoons *_1C*_2$ trivial reactions to combine all reactants into two monomers on one polymer; use a $*_1AB*_2 \rightarrow *_1CD*_2$ reaction to convert those two into two monomers representing the products; then use the reverse of the first process to separate those into the intended products. In the implementation CRN we have a monomer a for each formal monomer A (with $m(a) = (A; \emptyset)$), and a monomer for each prefix w of x, y, or any r_i or p_i. (If a string w is a prefix of multiple such strings, they will use the same w monomer.) We have an implementation monomer E with $m(E) = (\varepsilon, \emptyset)$, and reaction schemata $*_1 \rightleftharpoons *_1E$ and $*_1wE*_2 \rightleftharpoons *_1Ew*_2$ for every prefix monomer w (including formal monomers as prefixes of length 1). Where w is a prefix of any of the above and wA is the next prefix, we have a reaction schema $*_1wA*_2 \rightleftharpoons *_1E(wA)*_2$, where wA on the left means the two monomers w and A while (wA) on the right means the one monomer for the prefix wA. We have monomers r^i for $1 \leq i \leq k$ and p^i for $1 \leq i \leq l$; where r_i (resp. p_i) refers to the "prefix" monomer that is the entire string of the formal species r_i (resp. p_i) and $r^0 = p^0 = E$, we have reaction schemata $*_1 r^{i-1} *_2 + r_i \rightleftharpoons *_1 r^i *_2$ (resp. $*_1 p^{i-1} *_2 + p_i \rightleftharpoons *_1 p^i *_2$). Finally, where x (resp. y) is the prefix monomer for the entire string $x_1 \dots x_n$ (resp. $y_1 \dots y_m$), we have the reaction schema $*_1 r^k x *_2 \to *_1 p^l y *_2$. As an edge case, if $n = 0$ the reactants of that last reaction are $*_1 r^k *_2$, if $k = 0$ the reactants are $*_1 x *_2$, and if $n = k = 0$ the reactants are $*_1 E *_2$; the products are treated similarly if m and/or $l = 0$. We let $\rho' = (\Sigma' \cup \{\vdash\}) \times (\Sigma' \cup \{\dashv\})$; anything can bind to anything else, but we rely on the reaction schemata to keep the polymers formally valid and the consistency condition to ensure that they do. We show that this is a correct implementation, according to modular PRN bisimulation up to reachability, of the given formal reaction schema.

To show that this implementation is correct, we construct an interpretation; show that it satisfies the consistency condition; then show that it satisfies the atomic, delimiting, permissive, and modularity conditions. The interpretation is intuitive: where w is an implementation monomer that is a string of formal monomers, $m(w) = (w; \emptyset)$, $m(r^i) = (\varepsilon; \sum_{j=1}^{i} r_j)$, $m(p^i) = (\varepsilon; \sum_{j=1}^{i} p_j)$, and $m(E) = (\varepsilon, \emptyset)$. The consistency condition then follows from the assumption that $\{x_0 \mid (x_0, x_1) \in \rho\} = \{x_0 \mid (x_0, y_1) \in \rho\}$ and $\{x_{-1} \mid (x_n, x_{-1}) \in \rho\} = \{x_{-1} \mid (y_m, x_{-1}) \in \rho\}$: the only reaction schema that changes the π-interpretation of any polymer is the intended formal schema, $*_1 r^k x *_2 \to *_1 p^l y *_2$, which replaces an x_1 after $*_1$ and x_n before $*_2$ with y_1 after $*_1$ and y_m before $*_2$. (The $*_1 r^{i-1} *_2 + r_i \rightleftharpoons *_1 r^i *_2$ and similar p^i schemata create and destroy π-interpretations, but those r_i and p_i are by assumption valid formal species.) This allows m as a CRN interpretation to be defined.

The atomic condition follows from the polymer atomic condition, which is satisfied by the formal monomers as implementation monomers. The delimiting condition follows from the polymer delimiting condition: it is simple to confirm that all reaction schemata are syntactically interpreted as trivial except $*_1 r^k x *_2 \to *_1 p^l y *_2$, which is syntactically interpreted as the single formal reaction schema. To prove the permissive condition, it is simpler to prove the modularity condition first, with respect to all formal species as common formal species and all polymers made of only formal species as common implementation species. Given an arbitrary non-common implementation species, decompose it as follows: first, use $*_1 r^i *_2 \to *_1 r^{i-1} *_2 + r_i$ and $*_1 p^i *_2 \to *_1 p^{i-1} *_2$ schemata to produce a set of species with only prefix monomers and E monomers. Observe that $*_1 \rightleftharpoons *_1E$ and $*_1wE*_2 \rightleftharpoons *_1Ew*_2$ schemata can take any such polymer to any other such polymer with the same sequence of prefix monomers interspersed with any pattern of E's. In particular, for each polymer in the current decomposition, take that polymer to one where each prefix monomer w is to the right of exactly $|w| - 1$ E monomers. From such a state, $*_1E(wA)*_2 \to *_1wA*_2$ schemata will produce polymers with only formal monomers, finishing the decomposition to only common implementation species.

Given that every non-common implementation species can be decomposed via trivial reactions to common implementation species, we need only prove the permissive condition from minimal states consisting of only common species. For each formal reaction, i.e. each choice of w_1 and w_2 to be substituted for $*_1$ and $*_2$, exactly one such minimal state exists: $w_1x_1 \dots x_nw_2 + r_1 + \cdots + r_n$. This minimal state implements the formal reaction by the intuitive path: $*_1wA*_2 \to *_1E(wA)*_2$ reactions to reach $w_1E^{n-1}xw_2$ (in the edge case where $k > 1$ and $n = 0$ or $n = 1$, use $*_1 \to *_1E$ and $*_1wE*_2 \to *_1Ew*_2$ to reach w_1Exw_2); $*_1r^{i-1}*_2 + r_i \to *_1r^i*_2$ reactions with the initial $r^0 = E$ on the E directly to the left of x, reaching $w_1E^{n-2}r^kxw_2$ (in the edge case $k = 0$ ignore this step; in the edge case $n < 2$ the result will be $w_1r^kxw_2$); then the reaction $w_1E^{n-2}r^kxw_2 \to w_1E^{n-2}p^lyw_2$ is enumerated from $*_1r^kx*_2 \to *_1p^ly*_2$ and is interpreted as $w_1x_1 \dots x_nw_2 + r_1 + \cdots + r_k \to w_1y_1 \dots y_mw_2 + p_1 + \cdots + p_l$, satisfying the permissive condition. Any minimal state within this module implements that formal reaction by first decomposing all non-common implementation species then following the above path; any minimal state from outside this module satisfies the permissive condition by the modularity theorem; so this completes the proof that this interpretation is a PRN bisimulation up to reachability. □

Intuitively we thought the class of single-locus PRNs would be closed under PRN bisimulation, but quickly found a counterexample: a formal PRN with reaction schemata $A *_1 X \to A *_1 Y$ and $B *_1 X \to B *_1 Z$ is not single-locus, but can be implemented by single-locus reaction schemata $*_1x_A \to *_1y$ and $*_1x_B \to *_1z$ if $\pi(x_A) = \pi(x_B) = X$, if the implementation compatibility relation guarantees that x_A can only appear in a polymer whose interpretation begins with A, and x_B only in a polymer whose interpretation begins with B. In fact, a single-locus implementation schema has "computational power" equal to the computational power of its formal syntactic interpretation (in the sense of the polymer delimiting condition, Theorem 3.3.3) plus that of the regular expression restriction. (Given Lemma 3.3.2, this extra power would still be present had we defined PRNs using compatibility relations instead of regular expressions.) It is also important to note that in Definition 3.3.12 we defined PRN bisimulation as roughly a (π, μ) polymer interpretation whose induced CRN interpretation m is well-defined and is a CRN bisimulation, which cares about the set of implementation and formal reactions but not about the set of reaction schemata from which they were enumerated. Thus our statement about closed classes takes the form, "given a (possibly infinite) formal CRN and a single-locus implementation PRN with polymer interpretation that is a CRN bisimulation, the set of formal reactions is equal to the set of reactions enumerated from some set of augmented single-locus formal reaction schemata".

Theorem 3.6.2. *Let (Σ', e', Ψ') be a single-locus implementation PRN and (Σ, e, Ψ) be an arbitrary formal PRN, with PRN bisimulation interpretation (π, μ). Then $\mathcal{R}(\Sigma, e, \Psi) = \mathcal{R}(\Sigma, e, \Psi_0)$ for some augmented single-locus PRN (Σ, e, Ψ_0). Conversely, given an augmented single-locus PRN (Σ, e, Ψ_0) there is an implementation PRN (Σ', e', Ψ') where all schemata in Ψ' are of the types described in Theorem 3.6.1 with PRN bisimulation interpretation (π, μ). If e is local, then (Σ', e', Ψ') is single-locus, and further $(\Sigma', (\Sigma')^*, \Psi')$ is also single-locus and the same (π, μ) defined on that PRN is a PRN bisimulation up to reachability.*

Proof. Given (Σ', e', Ψ'), (Σ, e, Ψ), and (π, μ), we produce a Ψ_0 set of augmented single-locus reaction schemata with $\mathcal{R}(\Sigma, e, \Psi) = \mathcal{R}(\Sigma, e, \Psi_0)$. Recall the concept of "syntactically interpreting" a reaction schema, as used in Theorem 3.3.3: replace each implementation monomer with its π-interpretation and add its μ interpretation to the appropriate side of the reaction schema, producing a reaction schema defined in terms of formal monomers. The desired Ψ_0 is the set of syntactic interpretations ψ_i of each reaction schema $\psi'_i \in \Psi'$ (which, given that syntactic interpretations preserve the placement of wildcards, will be single-locus). For each $*_1$ (or $*_2$) in ψ'_i, because the schema is single-locus, it appears as either $*_1 x' \ldots$, $\ldots x' *_1$, $*_1 *_2$, or $*_1$ alone. In either case, the set of possible implementation sequences preceding or following some x', or forming the first (or last) part of a polymer, or forming an entire polymer, can be described by a regular expression over Σ'. The regular expression restriction $e_{i,1}$ (or $e_{i,2}$) is obtained from this regular expression by replacing each implementation monomer with its π-interpretation.

It follows from the three conditions of CRN bisimulation that the set of formal reactions $\mathcal{R}(\Sigma, e, \Psi)$ equals the set of nontrivial interpretations of implementation reactions in $\mathcal{R}(\Sigma', e', \Psi')$. (If π satisfies the consistency condition but not the compatibility condition, then this is true for the set of nontrivial interpretations of implementation reactions whose reactants are interpreted as valid formal species.) Then given any reaction enumerated from some $\psi_i' \in \Psi'$ (with the above condition if π only satisfies the consistency condition), its interpretation will be enumerated from the corresponding ψ_i: whatever values $*_1$ and $*_2$ take in the implementation enumeration, their π-interpretations will be the values of $*_1$ and $*_2$ in the formal enumeration. Those values, by construction, will satisfy the regular expressions $e_{i,1}$ and $e_{i,2}$, and the full (i.e., combining π and μ) interpretation of the monomers in ψ_i' will be the monomers and extra polymers in ψ_i; the compatibility or consistency condition, as appropriate, ensures that the formal interpretations match e so that the reaction is in fact enumerated. (If the μ-interpretation of implementation monomers in $*_1$ or $*_2$ is nonempty, then those monomers will appear as both reactants and products, and as discussed in Definition 3.3.12 its interpretation is the reaction without those spurious catalysts.) Given any reaction enumerated from some $\psi_i \in \Psi$, similarly consider the corresponding $\psi_i' \in \Psi'$. By construction, the regular expression restrictions on ψ_i represent all strings that are π-interpretations of some string of implementation monomers that would be a valid substitution for the appropriate wildcard; those strings for the formal values of $*_1$ and $*_2$ will be the implementation values of $*_1$ and $*_2$. Recalling that the spurious catalysts definition of bisimulation removes nonempty μ-interpretations, the interpretation of the implementation reaction so produced will be the formal reaction in question. Thus the set of reactions enumerated from Ψ_0 with restrictions is the set of interpretations of nontrivial reactions enumerated from Ψ', which since (π, μ) is a PRN bisimulation is equal to $\mathcal{R}(\Sigma, e, \Psi)$.

Given a formal augmented single-locus PRN $(\Sigma, e, \{\psi\})$ with one reaction schema, we construct an unaugmented implementation PRN (Σ', e', Ψ') and PRN bisimulation interpretation (π, μ), where every reaction schema in Ψ' is single-locus, and if e is local then so is e'. Given Theorem 3.6.1 and the transitivity and modularity results, this is sufficient to prove the statement of this theorem. Say ψ takes the form $\ast_1 x_1 \dots x_n \ast_2 + r_1 + \cdots + r_k \to \ast_1 y_1 \dots y_m + p_1 + \cdots + p_l$, where the string $x_1 \dots x_n$ may include $+$ (if $\ast_1$ and $\ast_2$ are on different polymers), and $\ast_1$ is restricted to match the regular expression e_1 while $\ast_2$ must match e_2. Let M_1 be an NFA recognizing e_1 and M_2 an NFA recognizing the reverse of e_2. Let Σ' be Σ together with species x_q and x'_q for $x \in \Sigma$ and q a state in M_1 or M_2. Construct e' as the intersection of three regular expressions as follows. First, replacing each x_q and x'_q with x should match e. Second, starting from the leftmost monomer may trace a valid partial computation of M_1 as follows: $k-1$ monomers of the form $(x_i)'_{q_i}$, $0 < i < k$, followed by a monomer $(x_k)_{q_k}$, such that where q_0 is the start state of M_1, $q_{i-1} \xrightarrow{x_i} q_i$ for $1 \leq i \leq k$. Third, starting from the rightmost monomer reading right-to-left may trace a valid partial computation of M_2 in the same manner; between these partial computations, only monomers $x \in \Sigma$ will appear. Observe that because the partial computation regular expressions are local, if e is local then so is e'.

Ψ' will have reversible reaction schemata

$$x \ast_1 \rightleftharpoons x_q \ast_1 \text{ for } (\vdash, x) \in \rho,\ q_{0;M_1} \xrightarrow{x} q;$$

$$\ast_1 x_q y \ast_2 \rightleftharpoons \ast_1 x'_q y_r \ast_2 \text{ for } (x, y) \in \rho,\ q \xrightarrow{y} r \in M_1;$$

similarly

$$\ast_1 x \rightleftharpoons \ast_1 x_q \text{ for } (x, \dashv) \in \rho,\ q_{0;M_2} \xrightarrow{x} q$$

and

$$\ast_1 x y_q \ast_2 \rightleftharpoons \ast_1 x_r y'_q \ast_2 \text{ for } (x, y) \in \rho,\ q \xrightarrow{x} r \in M_2;$$

and

$$\ast_1 x_q x_1 \dots x_n y_r \ast_2 + r_1 + \cdots + r_k \to \ast_1 x_q y_1 \dots y_m y_r \ast_2 + p_1 + \cdots + p_l$$

whenever q and r are accepting states of M_1 and M_2 respectively, and (x, x_1), (x_n, y), (x, y_1), and (y_m, y) are all in ρ. The polymer interpretation will have

$$\pi(x) = \pi(x_q) = \pi(x'_q) = x$$

and

$$\mu(x) = \mu(x_q) = \mu(x'_q) = \emptyset$$

for all x, q. The construction of e' implies that π satisfies the compatibility condition, and $\Sigma \subset \Sigma'$ with strings matching e also matching e' implies the atomic condition. Any reaction enumerated from one of the reversible reaction schemata will be trivial; for the last schema, which matches the formal schema, any reaction enumerated from that schema, to be made of valid implementation species, must have $*_1 x_q$ be an accepting computation of M_1 and $y_r *_2$ an accepting computation of M_2, implying that the corresponding formal strings match e_1 and e_2, and that the interpreted formal reaction is a reaction enumerated from ψ with the restrictions; thus the delimiting condition is satisfied. For the permissive condition, observe that any implementation polymer containing partial computations can reverse itself to a formal polymer, thus proving modularity (with respect to both sets of common species being $\mathcal{S}$), and that starting from only formal polymers whose interpretation can implement a reaction enumerated from Ψ, the implementation polymers can use the reversible reaction schemata to simulate M_1 on the beginning and M_2 on the end, at which point the nontrivial schema applies and the formal reaction can be implemented.

If e is not local, then applying Theorem 3.6.1 with transitivity and modularity to any number of reaction schemata in (Σ, e, Ψ_0) produces a (Σ', e', Ψ') with single-locus schemata in Ψ' (since the classes of schemata in Theorem 3.6.1 are all single-locus), but the PRN itself is not single-locus since e' is not local. However, if e is local, then so is e', so the same (Σ', e', Ψ') is single-locus, and for the same reason as shown in Theorem 3.6.1, the same (π, μ) defined on $(\Sigma', (\Sigma')^*, \Psi')$ is a PRN bisimulation up to reachability. □

3.7 Alternate polymer models and extended models

In defining linear Polymer Reaction Networks and PRN bisimulation, we made various choices of model properties. Alternative choices would have led to different models, some of which would have the same theorems applicable, some of which would have had different results. Here we briefly discuss two of those alternative choices, and what effects they would have had on the above theory.

Altered wildcards

Recall again the one-step and local string copying and comparison models in Figure 3.4. As opposed to the previous section, here we pay attention to what behaviors we can define if we don't care about single-locus restrictions on wildcards. If we want to copy, move, or remove an arbitrary polymer, or compare two polymers, wildcards as defined can do that:

$$*_1 + P \to *_1 + *_1 + P$$

On the other hand, consider a simplified model of RNA polymerase, written (in not-yet-defined notation) as:

$$*_1\{A, T, C, G\} + P \to *_1 + P + *_1[A \to a, T \to u, C \to c, G \to g]$$

Here RNA polymerase acting on a polymer made up of the DNA bases A, T, C, and G produces a copy replacing each DNA base with the corresponding RNA base a, u, c, or g, similar to the result of the string copying local model. String transcribing is not signficantly stranger than string copying, and it seems reasonable to construct a model that, if it can describe one as a one-step process, can do the same for both. We might similarly want to model effects that have a wildcard and its reverse, such as polymerase reverse-copying a single strand of DNA, or a stack and its reverse meeting and annihilating each other (also shown as a multi-step mechanism in Figure 3.4), or possibly other transformations of wildcards.

One way to define such a model is as follows: In a reaction schema, each wildcard $*_i$ must, in exactly one spot in the reactants, be written $*_i\{A_1, \dots, A_n\}$, for some set of monomers $\{A_j\}$. (As a notational convenience, $*_i\{\Sigma\}$ can be written as just $*_i$.) At any other point in the reactants and/or products where $*_i$ appears, it can appear as $*_i[A_1 \to B_1, \dots, A_k \to B_k]$, and/or be tagged $*_i^{rev}$. Such a schema is enumerated as follows: $*_i\{A_1, \dots, A_n\}$ is replaced by a string w_i containing only the A_j's, and modified instances of $*_i$ are replaced by w_i reversed and/or with each A_j replaced by B_j, as appropriate. Other tags, with the same syntax as reverse and with corresponding modifications in the semantics, could be defined as necessary. Any schema that uses any of these features, except $*_i\{\Sigma\}$ in the reactants with $*_i$ unmodified in the products, is not single-locus, since these features involve reading and/or writing arbitrarily large strings in the wildcards.

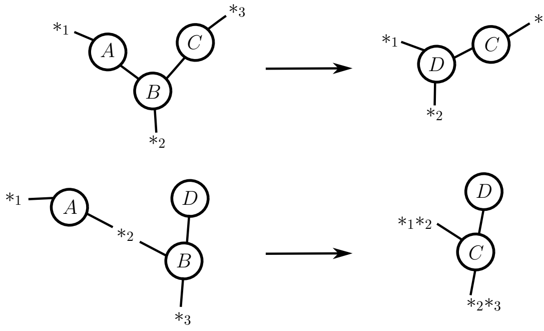
Mostly, the main content of this paper is orthogonal to this aspect of the model. A PRN with this extension is still enumerated into and treated as a (probably infinite) CRN; PRN bisimulation is still defined as previously discussed; the hardness results still apply. Single-locus PRNs are defined (as they should be) to exclude these features, so those results are similarly unaffected. Overall, we did not define PRNs with these features because we did not need these features to discuss the DNA stack machine, but the model should handle these features without too much difficulty.

It is also of interest when physically rotating a polymer in a way that reverses left and right causes the same molecule to be described by a different string of monomers. For example, say we want to represent fully double-stranded DNA as a linear polymer, and we let the monomer T represent a T base on the top strand paired with an A on the bottom strand, and similarly for A, C, and G. Then e.g. the strings $TGGC$ and $GCCA$ represent the same physical molecule, and any good model should recognize that. To handle this we could say that in a *rotatable PRN*, there is some function on monomers $x \to x^t$ with $(x^t)^t = x$, which extends to polymers such that $(uv)^t = u^t v^t$, and the species of the enumerated CRN are pairs of equivalent strings $\{w, w^t\}$. Reactions are enumerated from schemata such that for some substitution into the wildcards, whatever strings are produced, the pairs containing those strings are the CRN species involved in the enumerated reaction. This requires intuitive restrictions on the compatibility relation, $(x, y) \in \rho \iff (y^t, x^t) \in \rho$ where $\vdash^t = \dashv$, and on polymer interpretations, $\pi(x^t) = \pi(x)^t$ and $\mu(x^t) = \mu(x)$. The rest of the theory should be compatible without further changes. As an example, if the string-reverse detection local model from Figure 3.4 is taken as a rotatable PRN with $0_l^t = 0_r$, $1_l^t = 1_r$, $S^t = S$, and $Y^t = Y$, then it will identify a string over $\{0_l, 1_l\}$ with its reverse over $\{0_r, 1_r\}$, and two copies of the same such string will go through the mechanism that eventually produces Y.

Branched polymers

Examples of Turing-universal computation used in molecular programming, such as register machines [9], stack machines [57], and Turing machines themselves [56], tend to be linear. We therefore studied a system of linear PRNs, where each monomer can bind to at most two other monomers, and we can write a polymer as a string of monomers. We could instead have allowed each monomer to make either an arbitrary number of bonds, or up to some finite (characteristic of the monomer) number of bonds, either of which would allow us to model much more general systems. Such an approach would present some complications for defining reaction schemata, and present further complications for defining bisimulation, but we believe those complications are all solvable.

The obvious way to extend our definition of linear Polymer Reaction Networks to more general PRNs is effectively a graph-rewriting system with wildcards. In this sense, a (linear) polymer reaction schema such as $*_1 A *_2 + B \to *_1 C *_2$ is already a graph rewriting rule, where all graphs must be lines, and this is a straightforward generalization. Two examples of this would be:



When we defined polymer interpretations for PRN bisimulation, we defined a π-interpretation and a μ-interpretation. With linear polymers, it was easy to say that, given a string of monomers, we interpret them by concatenating their π-interpretations. The equivalent for nonlinear polymers, if a monomer's π-interpretation is anything other than a single monomer, is not obvious. One solution might be to say that each monomer must have a finite set of "faces", i.e. potential bonds, and for each face of an implementation monomer, its π-interpretation specifies a face of a formal monomer in the π-interpretation to correspond to that implementation face. As a special case, we could say that an implementation monomer whose π-interpretation is ε must have at most two faces, and if it has two faces both connected to something, the connections are connected to each other in the interpreted formal polymer. This concept of faces would also solve a similar problem with defining the compatibility relation, and would allow linear PRNs as defined above to be a subcase of branched PRNs, where every monomer has exactly two faces, "left" and "right". The rest of the theory of PRN bisimulation should extend naturally to branched PRNs. (The concept of faces still has a few details to be worked out, but we suspect it can be done.)

Single-locus PRNs can be defined for branched PRNs, and in fact can be defined in a somewhat more natural way than for linear PRNs. Recall Definition 3.6.1 of linear single-locus PRNs, in particular condition (iii), that no two distinct wildcards appear at the beginning of a polymer, and similarly for the end; this definition was motivated by, imagining a physical implementation of a single-locus reaction schema, such an implementation of a schema that violates condition (iii) would require an intermediate step that is not linear. When the underlying PRN model allows branched polymers, this is not as much of a problem. For branched PRNs, we would define single-locus reaction schemata as follows: (i) any wildcard that appears at all, appears exactly once in the reactants and exactly once in the products, and (ii) all wildcards have at most one bond. (Of the above branched reaction schemata, the first is single-locus and the second is in multiple ways not single-locus.) We suspect a theorem similar to Theorem 3.6.1 would be provable for branched single-locus PRNs.

The Biochemical Ground Form (BGF), discussed by Cardelli and Zavattaro [9], serves as an example of what a branched polymer model could look like. (The concept of faces, for example, corresponds roughly to association labels in the BGF.) The BGF, instead of graph rewriting reaction schemata, defines reactions in terms of the actions of different "agents" (monomers), some of which may require coordination with other agents. Implicitly, if a monomer A can take an action a, it can do so regardless of what that monomer is bound to, which in our way of writing means every reaction (schema) has all possible wildcards. In particular, we suspect every BGF system could be written a single-locus branched PRN. The BGF as described has no mechanism for a monomer to coordinate *specifically with another monomer bound to it*, as opposed to another monomer of the specific type that may be on a different polymer; with such a mechanism, we suspect but have not proven that the BGF could implement, up to bisimulation, any single-locus branched PRN.

3.8 Discussion

Our main claim is that polymer CRN-like systems are a strong candidate for powerful and practical molecular computation; that formal verification is useful for systematic construction of (eventually, large) polymer systems; and that bisimulation is a useful technique in formal verification of polymer systems. To show this, we defined a model of linear Polymer Reaction Networks, and defined PRN bisimulation based on that model. We proved some useful properties of PRN bisimulation; we showed how to use PRN bisimulation to verify an existing system; and we showed an example of how PRN bisimulation can identify good design strategies for implementing a large class of systems. Although we did all of this within the model of linear PRNs, we discussed how PRN bisimulation is likely to be applicable, and our results translatable, to other models of polymer CRN-like systems. Thus, even if this model of linear PRNs is not the optimal model for polymer systems in molecular programming, the concept of PRN bisimulation will likely remain useful.

Our definition of PRN bisimulation interprets each state of the implementation system as a state of the formal system, and checks whether, from any initial state, the possible trajectories of the two systems are equivalent under that interpretation. However, it ignores quantitative aspects of the system such as rate constants, meaning PRN bisimulation says nothing about the kinetics of the system (i.e. how long things take) or the probabilities of the various possible trajectories. It also assumes that the model of the implementation system as a PRN is accurate, and the model we used in this case ignores the "leak reactions" and other side reactions typical of DNA strand displacement systems; with no way to distinguish between likely and unlikely reactions, PRN bisimulation evaluated on a model including leak reactions would say that the implementation is incorrect. This means that *when an implementation is proven correct according to PRN bisimulation, we know that a specific class of its behavior is equivalent to the corresponding behavior of its specification (formal PRN), namely the rate-independent behavior up to (if applicable) whatever model we used to describe the implementation system as an implementation PRN.* In systems such as stack machines and Turing machines, the rate-independent behavior is the only relevant behavior of the abstract system, so PRN bisimulation proves that the implementation has the behavior we want. In systems such as oscillators or dynamic instability, while PRN bisimulation can prove some correspondence between the implementation and the abstract system, it may not be able to say anything about the kinetics that imply the relevant behavior. (Whether an extension of PRN bisimulation can take kinetics into account is, as it is for CRN bisimulation [40], an important open question.) Intuitively we expect that for "systematic implementations" such as the stack machine or the various CRN translation schemes, if the scheme has no qualitative (i.e., detectable by CRN/PRN bisimulation) errors then its kinetics are "close enough" to and/or can be tuned to match those of the abstract system. Experimental implementations such as the CRN oscillator by Srinivas et al. [67] suggest this is the case, and the experiments of Chen et al. [19] demonstrate an experimentally working CRN even when CRN bisimulation identifies a potential error (that, presumably, averages out). Polymer systems especially, compared to well-mixed CRNs, are more likely to depend on rate-independent computation and not care about kinetics; for example, well-mixed CRNs require kinetics to approximate the behavior of "A happens, then B happens", while polymer systems can use geometric separation to achieve the same thing with less probability of error. (This is more true for polymer systems that simulate classic models of computation than for those found in biology.) This is why we claim that, while PRN bisimulation cannot prove correct every relevant aspect of an implementation PRN in general, it is a useful tool to verify the important aspects of many useful polymer systems.

The simplest thing to do with PRN bisimulation is to, given one formal PRN and one putative implementation, verify by hand that the implementation matches the formal PRN. We demonstrated an example of this with the DNA stack machine from Qian et al. [57]. We suspect that bisimulation can be used in more powerful ways, such as automated verification of systems too large to verify by hand, or as a basis for formal proofs that certain classes of systems will or will not be correct implementations of other classes, or as an intuition to guide designers of molecular devices in their search for correct implementations.

As we would expect for a model equivalent in power to Turing machines, whether two systems are PRN bisimulation equivalent is undecidable in general, but this does not rule out any form of computer-aided verification. Exactly what form such verification could take, we don't know, but we have two possibilities to suggest. The main problem that produces the undecidability result stems from the permissive condition, that for every formal reaction in any implementation state whose interpretation can do that reaction, the implementation state can implement the formal reaction after some sequence of zero or more trivial reactions. The problem is that there is no upper bound on the number of trivial reactions; the undecidability result uses a formal reaction that can be implemented only if a Turing machine computation made of trivial reactions halts. Systems intended to be built in practice typically use a small, and in particular bounded, number of trivial reactions per formal reaction. Based on this, the first suggestion is that some bound on the number of trivial reactions may give a definition of PRN bisimulation that is decidable or even tractable. Exactly what type of bound is best, and whether this idea covers all the physical implementations we care about, is unknown. Similarly, systems intended to be built in practice typically have a designer who knows how the system is intended to work, and can provide a “proof” that the permissive condition is satisfied, as we did for the DNA stack machine above. The hardness result shows that not every correct implementation will have a finite proof at all, let alone one that can be checked in reasonable time, but it may be that a large enough class of “reasonable” implementations does. How exactly such a proof should be specified, and what class of systems can be proven correct this way, is unknown.

That formal verification methods such as PRN bisimulation can be used to guide design is a speculation of ours. We showed a concrete example of this idea with the proof that any “physically realistic” (single-locus) PRN can be implemented by five reaction schema “primitives”. This sort of result will likely be helpful for designing complex polymer systems, where whatever complex behavior the designer needs can be implemented in a known way with simple primitives, which themselves can be implemented in some known way yet to be discovered. We further hope that, with a formal definition in mind of what makes a correct implementation, someone designing physical implementations would have a better idea of what systems to design.

Chapter 4

SIMPLIFYING CHEMICAL REACTION NETWORK IMPLEMENTATIONS WITH TWO-STRANDED DNA BUILDING BLOCKS

4.1 Perspective

If there's a common bond between the chapters of this thesis, it's the attempt to improve the CRN-to-DSD compilation process, from a proof-of-concept that CRN compilation can be done in theory to a process efficient enough to be *the* practical way to implement its class of molecular programs. One necessary part of this is optimizing the result of the compilation. Since the original CRN-to-DSD implementation scheme [66] a few other ideas [6, 57] have been proposed for what, concretely, the DSD implementation of a CRN should look like. Each scheme has its own strengths and weaknesses. In this chapter I discuss another two such schemes, with properties often not shared by previous schemes: both use DNA complexes with no more than 2 strands each; both use 4-way branch migration instead of 3-way branch migration; and both are physically reversible.

What does it mean to optimize a DSD system? First, all DSD CRN implementations so far require "fuel species" (or "fuels"), DNA complexes that have to be synthesized by whatever method and added to the DSD system at the start. When testing DSD circuits in the lab, fuels are chemically synthesized, annealed, and manually added to the test tube; in the hypothetical future where DSD is used in autonomous molecular devices, those devices would need some as-yet-undecided mechanism to synthesize or input fuels. Any property of the fuel species, such as length of strands, number of strands, or number of fuels, that makes them more costly to synthesize, or more difficult to synthesize without undesired byproducts, is thus a target for optimization. Second, no physical DSD system ever does exactly what the formal DSD model says it should. Some of this is due to improbable, but not impossible, "leak reactions" not included in the formal model, while some is due to the aforementioned undesired byproducts or other imperfect synthesis of the fuels [67]. In particular, Lulu Qian and I focus on limiting fuels to 2-stranded complexes because the seesaw gates, which are always 2-stranded, have been shown to be particularly robust [55, 72]. Physical reversibility is also useful, since it reduces the quantity of fuel consumed by reversible reactions.

Now that we know what an optimized DSD system is, how do we as researchers and/or designers find one? I don't know the general answer to that, but in this chapter I suggest a direction in which to search. We discuss five "motifs", simple DSD reactions involving 1- or 2-stranded reactants, including one novel motif based on cooperative 4-way branch migration. Each motif can be discussed in terms of its inputs and outputs, what features each has, and which upstream or downstream motifs they can feed into, on a level more abstract than the low-level formal DSD model (which is itself more abstract than physical DNA strand behavior) but less so than abstract CRNs. When we discuss CRN implementations, including existing CRN implementations as well as the two new implementation schemes we propose, we describe the implementations mostly in terms of the motifs without needing the details of the low-level DSD reactions. Abstraction hierarchies are generally important for effectively designing systems, and this "motif way of thinking" is a candidate abstraction hierarchy for designing complex DSD systems.

In this chapter, we present one novel motif and two novel CRN implementation schemes, plus describe how we used motifs to design and analyze the schemes. We also prove both schemes correct according to CRN bisimulation, and discuss how they compare to existing implementation schemes. To summarize the result of that comparison, we reduced complexity in the size of our fuels at the expense of needing a significantly larger number of fuels. Whether this tradeoff is worth it—in general or in some specific circumstances—we don't know, but now the option exists. This then suggests the question of whether our design can be improved to use fewer complexes while still remaining 2-stranded and reversible, or whether we innately have to trade simpler mechanisms for more steps. It also suggests the question of, since we're comparing our 4-way-branch-migration-based schemes to existing 3-way-branch-migration-based schemes, if the two different types of DSD mechanism have different natures when used to make larger systems. I suspect that there are some interesting features both of 2-stranded complexes and of 4-way branch migration, and discuss those questions further in Chapter 5.

The remainder of this chapter has an introduction and discussion specific to this thesis, but is otherwise based on the following manuscript in progress:

> Robert F. Johnson and Lulu Qian. Simplifying chemical reaction network implementations with two-stranded DNA building blocks. *In preparation*, 2020.
> Contributions: Work done primarily by RFJ with advice and assistance from LQ.

4.2 Introduction

Past chapters have discussed the Chemical Reaction Network (CRN) model as a programming language, and the ability to "compile" CRNs to DNA Strand Displacement (DSD) systems. Implementation schemes such as those of Soloveichik et al. [66], Qian et al. [57], and Cardelli [6] are the current state of the art, and have been demonstrated experimentally on small CRNs [19, 67]. The next step in making CRN programs practical is to "scale up" the size of the CRNs that can be physically built, and generally reduce the leak and error rates.

In terms of robust DSD systems, we can take a lesson from experiments with seesaw gates [55, 72]. The key point is based on the fuel complexes, which were briefly mentioned in previous chapters on bisimulation: from that perspective, fuels were DNA complexes that were assumed to be always present and thus removed from the enumerated CRNs before verification analysis. From an experimental perspective, to be "always present" means one has to synthesize and add a large number of each fuel complex, and if the fuel is imperfectly annealed or otherwise misbehaves, the error is large relative to the system's signal. For a two-reactant two-product reaction, the Soloveichik et al. translation scheme uses 3-stranded fuels [66], the Cardelli scheme 4-stranded fuels [6], and the Qian et al. scheme (in the corrected version in Chapter 3) a 5-stranded or a 7-stranded fuel. The seesaw gates compute logic gates which are less complex than chemical reactions, but they do so with only single strands and 2-stranded complexes [55]. Possibly because of this, they have been used to build larger circuits and to be robust to experimental imperfections, such as unpurified strands [72].

For this purpose, Lulu Qian and I have been investigating implementing CRNs using only 2-stranded fuels. Simple DSD systems, such as detecting a desired sequence [18] or AND gates [35], are often 2-stranded, in addition to the seesaw gates mentioned above. There is even a class of hairpin-based systems that construct larger structures from single-stranded initial complexes [83], including the Hybridization Chain Reaction often used in imaging [23], and a design for hairpin-based logic circuits [28]. However, none of these are a full Chemical Reaction Network implementation, or even an equivalently powerful dynamical system—while logic gates are universal for computing functions, CRNs have a dynamical behavior that logic gates in general do not.

This chapter contains the positive side of our work on 2-stranded CRNs, with some limits discussed in Chapter 5. We discuss four known 2-stranded DSD motifs that can serve as building blocks for such implementations, and we present a new cooperative 4-way strand exchange motif that starts with 2-stranded complexes. We discuss two ways of implementing general CRNs with these motifs, and tradeoffs between the two schemes. Finally, we show how, using CRN bisimulation, these schemes can be proven correct assuming the assumptions of the formal DSD model reflect real DSD systems.

We believe that having abstract descriptions of simple motifs will help the design of complex DSD systems. Whatever complex behavior is desired, it may be easier to implement by combining the simple logical operations of known motifs. To demonstrate this, we first discuss the 5 motifs and their behavior on an abstract level, then show how various CRN implementations can be constructed and comprehended by combining those abstract behaviors.

4.3 Two-stranded motifs

We identify five “motifs”, or simple condensed reactions, out of which we build two-stranded CRN implementations. Four of these motifs have been previously studied, while one is new. We discuss the properties of each motif in itself, while in Section 4.4 we will discuss how those properties interact when building larger circuits. For building two-stranded CRNs, key questions about a given motif are what logical operation it represents, whether its outputs have the form of its inputs and/or the inputs of the other motifs, and whether its outputs and reverse gates are 2-stranded.

Toehold Exchange A reversible 3-way strand displacement exchanges which of two strands is bound to a gate (Figure 4.1 (a)). The input strand is an unbound toehold-long domain combination, while the input gate has that long domain bound with that toehold open. The reaction has two high-level effects. First, the output strand has the same long domain (B, in the figure) in a

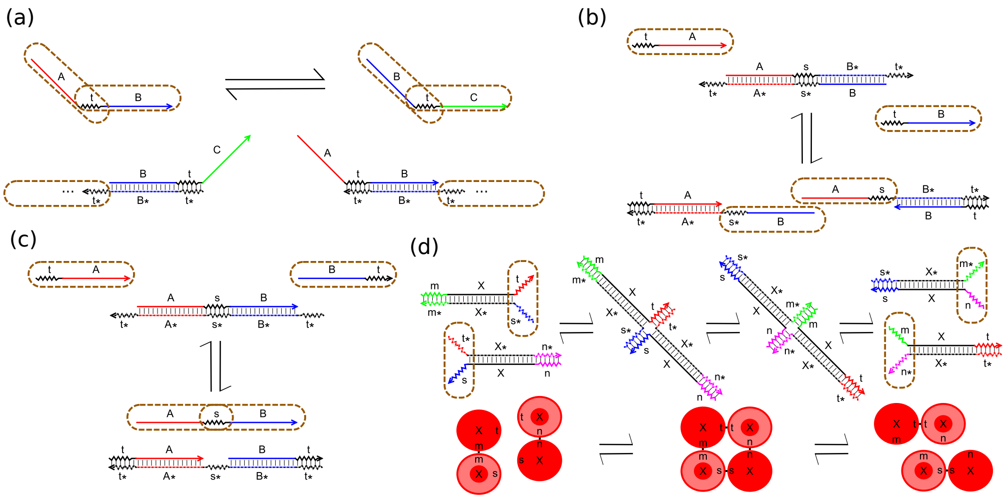


Figure 4.1: Four previously studied reversible 2-stranded DSD motifs, shown through common examples. (a) Toehold exchange; (b) Symmetric cooperative hybridization; (c) Asymmetric cooperative hybridization; (d) 4-way strand exchange, with a diagram used in the abstracted notation we will introduce.

different toehold context, and may have different long domains (A versus C) on the other side of its newly open toehold. Second, the gate now has a different toehold open, which may allow interaction with adjacent domains. See for example the first CRN implementations [66], seesaw gates [55], and various others [87].

Cooperative Hybridization (symmetric) Two 3-way strand displacement reactions occur simultaneously on either side of a gate complex, meeting in the middle and allowing the two halves to dissociate only if both inputs are present (Figure 4.1 (b)). The input strands are unbound toehold-long domain combinations, while the output signals have the same long domains adjacent to different open toeholds. See for example Cherry et al.'s winner-take-all circuits [20].

Cooperative Hybridization (asymmetric) Two 3-way strand displacement reactions occur simultaneously on either side of a gate complex, meeting in the middle and releasing an output strand only if both inputs are present (Figure 4.1 (c)). The input strands are unbound toehold-long domain combinations, while the output strand has those two long domains in combination with a different toehold; but with only one toehold, barring complex mechanisms either one but only one of them can react. However, even if both inputs are single strands the reverse gate is a 3-stranded complex, so this motif is not “reversible with 2-stranded fuels”. Introduced and tested by Zhang [86].

4-way strand exchange Two 2-stranded complexes bind by two toeholds and exchange strands via 4-way branch migration (Figure 4.1 (d)). The inputs are 2-stranded complexes sharing a common long domain, with complementary pairs of open toeholds and (if the reaction is reversible) a closed toehold on each. The outputs are 2-stranded complexes in the same form, with the formerly open toeholds now paired up and closed and the formerly closed toeholds now split and open. Experimentally tested by Dabby [22].

4-way Cooperative Hybridization Two 4-way branch migrations happen on either side of a gate, meeting in the middle and separating into two intermediate complexes (Figure 4.2). Each of the two products carries only half of the information of the original reactants, thus products of different instances of this reaction can interact in the reverse reaction. The effect of such a quadruplet of reactions is strand exchange between one pair of complexes coupled to strand exchange between the other, simultaneously changing the open toehold combinations on distinct long domains.

While the other four mechanisms discussed have been experimentally demonstrated to work, cooperative 4-way branch migration has not yet been tested. In particular, the final dissociation step requires 3 toeholds separated by two 4-way junctions to dissociate. We think this is plausible, based on Dabby’s observation that 2 toeholds separated by one 4-way junction can dissociate [22]; or, if this is not the case, that there is some $0 < Length(l) \leq 6$ for which that dissociation is possible and reversible.

An abstraction for 4-way strand exchange Common to both uncooperative and cooperative 4-way strand exchange is a basic signal complex: two strands, one long domain bound to its complement flanked by one bound pair of complementary toeholds and one open pair of non-complementary toeholds, as seen repeatedly in Figures 4.1 (d) and 4.2. As both types of 4-way strand exchange transform complexes of this form into complexes of the same form with different domain combinations, we find an abstract description of this type of molecule useful. For example, we write the molecule with long domain X, open 3’ (end of the DNA) toehold t, open 5’ toehold s^*, and bound toehold m as $X(t, s; m)$. When the long domain is unimportant or universal, such as a system composed entirely of uncooperative 4-way strand exchange, we omit it and write simply $(t, s; m)$. For experimental reasons we prefer to have strands made up of only non-$*$ or only $*$ domains, and design non-$*$ and $*$ domains to have distinct sequence properties (for example, using a three-letter code [55]). Then $X(t,s;m)$ unambiguously describes the top reactant of Figure 4.1 (d), with s understood to mean an open s^* toehold. With that assumption, the top product in Figure 4.1 (d) would be $X(m,n;s)^*$, with the first toehold listed still being on the 3' end of its strand, but now understood to mean an open m^* toehold. Without that assumption, we might use a more general notation where those molecules are $X(t,s^*;m)$ and $X^*(m^*,n;s^*)$ respectively. The circle abstraction shown in said figures is also useful to illustrate strand exchange reactions. Each circle represents a strand with one long domain and two toeholds, where half-faded circles represent strands made of $*$ domains. Thin connections (both figures) represent strands bonded directly, requiring matching domains; thick connections labelled with a toehold domain (horizontal in Figure 4.2) represent strands connected by gate strands from a cooperative 4-way strand exchange reaction, which can be between any domains so long as the appropriate gate exists.

4.4 Chemical Reaction Network implementations

The above motifs can be combined in various ways to construct implementations of arbitrary Chemical Reaction Networks. To implement arbitrary CRNs, the reaction $A+B \to C+D$ (or $A+B \to C$ and $A \to B+C$) is sufficient; for arbitrary reversible CRNs, the reaction $A+B \rightleftharpoons C$ (or *a fortiori*, $A+B \rightleftharpoons C+D$) is sufficient. From a logical perspective, “join” and “fork” operations are sufficient; the above reactions represent those logics.

CRN implementations typically have *signal* complexes that are the primary form of a given formal species, and *fuel* complexes that are assumed to be always present and drive the reactions. For a CRN to have “only 2-stranded inputs”, as desired in this work, means that all signal complexes and fuel complexes are single strands or 2-stranded. We implicitly assume that we are discussing *systematic* CRN implementations, where we give a template for a generic reaction and construct larger CRNs by combining independent copies of the template with different domain identities. In such a case we can ask how the number of toehold domains scales, i.e. whether different reactions can use the same toeholds or have to create new ones; as toeholds are limited in length by thermodynamics, a system with $O(n)$ toeholds may be able to implement small CRNs but a system with $O(1)$ toeholds is better if possible.

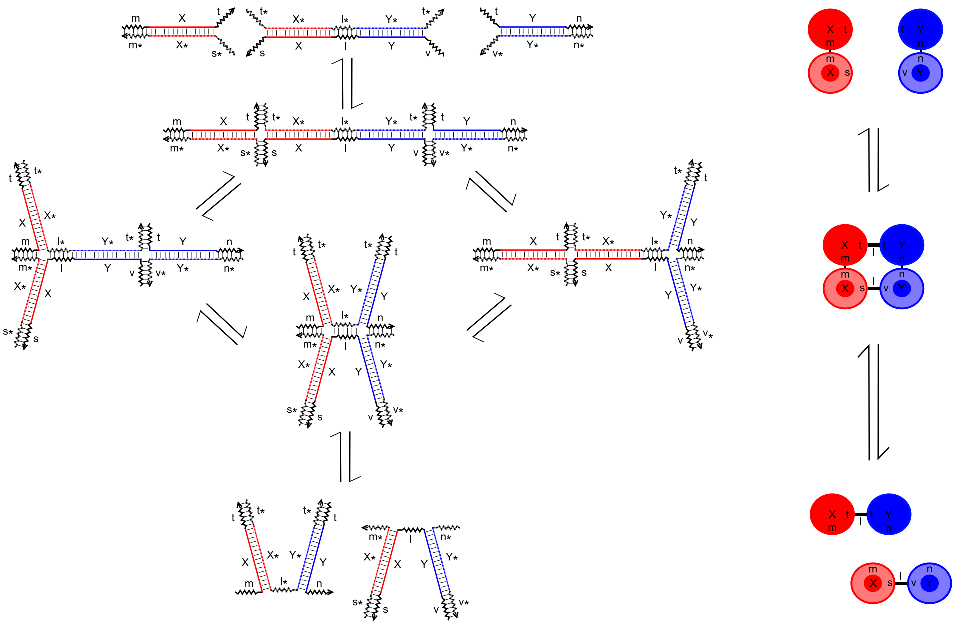


Figure 4.2: A cooperative 4-way branch migration mechanism. Initial X and Y complexes combine with a gate that matches their open toehold combinations, producing two 3-stranded complexes each with one of the strands of X and one of the strands of Y. These complexes can recombine with each other or with the corresponding products of a similar reaction, which in the latter case will produce X and Y complexes with different toehold combinations. On the right, this reaction is shown in abstracted form.

Whether a scheme requires cooperative mechanisms is worth noting. Finally, it is desirable if reversible reactions ($A+B \rightleftharpoons C+D$) can be implemented with physically reversible mechanisms, so that going forward and backward multiple times does not consume fuel; to be truly reversible, the 2-stranded fuel criterion should include the reverse fuels as well. These criteria are discussed more fully and more formally in Chapter 5.

Toehold Exchange-based CRNs Existing CRN implementations [6, 57, 66] are often based on toehold exchange mechanisms where e.g. $A + B \rightarrow C$ is implemented by a toehold exchange reaction with A opening a toehold on the gate for a reaction involving B. These schemes can be understood in light of the motifs previously discussed: the property of toehold exchange that a different toehold on the gate is opened allows join and fork logic. The property that the released strand has a different long domain/toehold combination is used to pass signals between gates.

Such a mechanism seems to require a 3-stranded complex for the gate molecule to achieve join logic, so it does not meet the goal of this paper, but is worth mentioning as the current state of the art. Another relevant mechanism using toehold exchange is the seesaw gate [55], where transduction logic combines with threshold logic to check whether the total amount of signal is more than either A or B can produce by itself. This achieves join logic for macroscopic signals but cannot satisfy criteria such as CRN bisimulation for individual molecules.

3-way Cooperative CRNs The symmetric cooperative hybridization is $A + B \rightleftharpoons C + D$ logic, if we consider the same long domain in a different toehold context to be a different signal. Since toehold exchange reactions depend on the combination of long domain and toehold, this is valid. Thachuk et al. use a combination of symmetric cooperative hybridization and toehold exchange to implement leakless $A + B \rightarrow C + D$ reactions in exactly this manner (personal communication with Chris Thachuk regarding unpublished results, but based on their work on leakless gates [70, 78]).

From our perspective, the only problem is that symmetric cooperative hybridization with 1-stranded inputs produces 2-stranded products, and toehold exchange with a 2-stranded input signal produces a 3-stranded reverse gate. For physically reversible reactions, this 3-stranded gate would be considered a reverse fuel, and the system would not have entirely 2-stranded fuels. Thus this mechanism meets all our criteria for irreversible CRNs, but not reversible CRNs.

4-way-based CRNs with $O(n)$ toeholds The two-toehold-mediated 4-way strand exchange mechanism effectively exchanges toeholds on a common long domain; note that while the inputs both have t and s toeholds, the outputs have one with only t and one with only s. When a signal complex goes through multiple copies of this reaction with different fuels, it can turn any combination of toeholds into any other combination. When two signals with complementary pairs of toeholds meet in this reaction, it produces two signals with different combinations in $A + B \rightleftharpoons C + D$ logic. So for example, we can turn $(a_1, a_2; a_3)$ into $(r_1, r_2; r_3)$ and $(b_1, b_2; b_3)$ into $(r_2, r_1; r_4)$, which will react and produce $(r_3, r_4; r_2)^*$ and $(r_4, r_3; r_1)^*$, which can be turned into $(c_1, c_2; c_3)$ and $(d_1, d_2; d_3)$ respectively. Thus two-toehold-mediated 4-way strand exchange alone can implement arbitrary reversible CRNs if we allow $O(n)$ toeholds.

A list of all species involved is given in Table 4.1. Note that fuels $(r_2, r_1; r_5)$

A	$\emptyset$	B	$\emptyset$
$(a_1, a_2; a_3)$	$(a_2, a_1; r_5)$	$(b_1, b_2; b_3)$	$(b_2, b_1; r_6)$
$(r_5, a_3; a_1)^*$	$(a_3, r_5; a_2)^*$ $(a_3, r_5; r_2)^*$	$(r_6, b_3; b_1)^*$	$(b_3, r_6; b_2)^*$ $(b_3, r_6; r_1)^*$
$(r_2, a_1; r_5)$	$(a_1, r_2; a_3)$ $(a_1, r_2; r_3)$	$(r_1, b_1; r_6)$	$(b_1, r_1; b_3)$ $(b_1, r_1; r_4)$
$(r_3, r_5; r_2)^*$	$(r_5, r_3; a_1)^*$ $(r_5, r_3; r_1)^*$	$(r_4, r_6; r_1)^*$	$(r_6, r_4; b_1)^*$ $(r_6, r_4; r_2)^*$
$(r_1, r_2; r_3)$	$(r_2, r_1; r_5)$	$(r_2, r_1; r_4)$	$(r_1, r_2; r_6)$

C	$\emptyset$	D	$\emptyset$
$(c_1, c_2; c_3)$	$(c_2, c_1; r_3)$	$(d_1, d_2; d_3)$	$(d_2, d_1; r_4)$
$(c_3, r_3; c_2)^*$	$(r_3, c_3; c_1)^*$ $(r_3, c_3; r_2)^*$	$(d_3, r_4; d_2)^*$	$(r_4, d_3; d_1)^*$ $(r_4, d_3; r_1)^*$
$(c_2, r_2; r_3)$	$(r_2, c_2; c_3)$ $(r_2, c_2; r_4)$	$(d_2, r_1; r_4)$	$(r_1, d_2; d_3)$ $(r_1, d_2; r_3)$
$(r_3, r_4; r_2)^*$	$(r_4, r_3; c_2)^*$	$(r_4, r_3; r_1)^*$	$(r_3, r_4; d_2)^*$

Table 4.1: List of species for the 4-way $O(n)$-toeholds reaction $A+B \rightleftharpoons C+D$, in the abstracted notation. Species in columns A, B, C, and D represent the given formal species. Species in columns labeled $\emptyset$ are fuels and assumed to be always present. a_i domains are toeholds specific to species A, and similarly for B, C, and D; r_i domains are specific to the reaction $A + B \rightleftharpoons C + D$; this ensures no crosstalk with other pathways.

and $(r_1, r_2; r_6)$ can interact, but the products can do nothing but reverse the reaction, and the same is true for $(r_4, r_3; c_1)^*$ with $(r_3, r_4; d_1)^*$.

4-way Cooperative CRNs The cooperative 4-way strand exchange motif, when its products recombine with products of a different instance of the reaction, *simultaneously* exchanges the toehold combinations on a complex with long domain X and a complex with long domain Y. If $A(t, s; m)$ is the signal molecule for A, then simultaneously breaking the (t, s) combination on A and putting together a (u, v) combination on some long domain R is effectively converting $A(t, s; m) \rightleftharpoons R(v, u; n)^*$ if all other molecules involved are considered fuels. Where R is unique to the reaction $A + B \rightleftharpoons C + D$, we can convert the four signal species from their own long domains to the R domain, then use a two-toehold-mediated 4-way strand exchange reaction to implement the reaction itself. In contrast to the previous implementation scheme, that each reaction has a different long domain allows the toeholds (u, v, etc.) to be universal, using $O(1)$ toeholds at the expense of requiring cooperative hybridization.

As this scheme is based on the $O(n)$-toehold scheme, we reuse the mechanism from Table 4.1. Assume all complexes in that list have long domain R, unique to the reaction $A + B \rightleftharpoons C + D$. To the toeholds listed, add toeholds t, s, m, n, l, and let $a_3 = b_3 = c_3 = d_3 = n^*$. Then use cooperative 4-way strand exchange to convert $A(t, s; m) \rightleftharpoons (R^*(a_1^*, a_2^*; n))^* = R(a_1, a_2; n^*)$ (the fuel will have R^* on the "top" strand with A), $B(t, s; m) \rightleftharpoons R(b_1, b_2; n^*)$, $C(t, s; m) \rightleftharpoons R(c_1, c_2; n^*)$, and $D(t, s; m) \rightleftharpoons R(d_1, d_2; n^*)$. This gives a mechanism with one long domain per species, one long domain per reaction, and a total of 19 toeholds. Because the long domains now indicate species/reaction identity, the toeholds can be shared between all species and reactions without crosstalk.

4.5 Correctness of the schemes

Table 4.1 is effectively a proof of the correctness of the $O(n)$-toehold 4-way-based scheme according to CRN bisimulation (chapter 2). For each $A + B \rightleftharpoons C + D$ reaction, construct a copy of this mechanism with unique r_i domains, but any a_i domains in common with other reactions using the same formal species; reactions with fewer reactants or products can have one of A, B, C, or D as a fuel; reactions with more reactants or products should be broken into steps with at most 2 of each. DNA complexes in columns labeled A, B, C, or D are interpreted as one copy of the corresponding species, while complexes in columns labeled $\emptyset$ are fuels. Formally, fuels are assumed always present and removed from the enumerated implementation CRN before bisimulation verification; so for example the physical pathway $(r_2, a_2; r_3) + (a_2, r_2; r_5) \rightleftharpoons (r_5, r_3; r_2)^* + (r_3, r_5; a_2)^*$ would be represented as $(r_2, a_2; r_3) \rightleftharpoons (r_5, r_3; r_2)^*$, and then interpreted as the trivial reaction $A \rightleftharpoons A$. Using the abstraction for 4-way strand exchange notation, the table is structured such that each non-fuel species can interact with the (usually two) fuel species in the same row, producing the corresponding fuel+non-fuel pair above or below it; that the final $A + B$ forms react to produce the final $C + D$ forms, while their fuels also have a spurious-but-harmless reaction with each other; and that, given the uniqueness of the domains, no other intra-module or inter-module reactions exist. Then all reactions are trivial except $(r_1, r_2; r_3) + (r_2, r_1; r_4) \rightleftharpoons (r_3, r_4; r_2)^* + (r_4, r_3; r_1)^*$ which is interpreted as $A + B \rightleftharpoons C + D$; with $(a_1, a_2; a_3)$ etc. as the "common species", the common species can implement the formal reaction; and any intermediate species can turn into the common species with the same interpretation by interacting with only fuels. As these are the conditions of modular CRN bisimulation, this completes the proof: any combination of these modules will be a correct implementation according to CRN bisimulation.

For the cooperative 4-way scheme, the same bisimulation logic applies. In this case the lack of crosstalk between modules is assured by the distinct long domains; even if toehold combinations are identical, different long domains will make the reaction unproductive. The remaining caveat is with the cooperative 4-way mechanism itself. We designed the system so that the toeholds along the cooperative reaction are *always* m, l, n. Thus, we *assume* that intermediates of the cooperative pathway will all have the matching m, l, n toeholds, and all three toeholds will bind and dissociate as a unit. Whether this is actually true or not will be determined experimentally; if not, there may be problematic crosstalk between, for example, an (A, R_1) and (A, R_2) pair of long domains which leads to temporarily duplicated signals. If it is true, however, then the result of such a crosstalk will be a release of one side with the other suspended, one of which carries the signal, and the system will be correct according to bisimulation.

4.6 Discussion

We discussed the use of DNA Strand Displacement to implement Chemical Reaction Networks, and the desire to create larger, more robust DSD CRN implementations. We then presented 2-stranded DSD motifs which we used to build 2-stranded CRN implementations, in the hope that they would be more robust than those which rely on 3-or-more-stranded complexes. There is some indication that 2-stranded DSD systems in general are more robust, but whether these particular systems are more robust than the current state-of-the-art CRN implementations is an open question.

We can compare Soloveichik et al.'s original CRN scheme [66, 67] (which is reasonably representative of other toehold exchange schemes), our $O(n)$-toehold 4-way strand exchange scheme, and our ($O(1)$-toehold) cooperative 4-way strand exchange scheme. While 3- and 4-stranded complexes may be less robust, in other aspects the toehold exchange scheme is simpler than our two schemes: it uses one long domain per formal species, one long domain per reaction, and can be done with a single, universal toehold. To go from reactant signal species to product signal species in the toehold exchange scheme (as implemented experimentally [67]) takes 4 toehold exchange steps in an $A + B \to C + D$ reaction, and generalizes naturally to $n + m$ steps in an n-reactant m-product reaction. In contrast, while the cooperative 4-way scheme also uses one long domain per formal species and reaction, as described above it uses 19 universal toeholds and takes 30 reactions for $A + B \to C + D$. (By "reaction" I mean roughly one condensed reaction as described in Peppercorn [36] or Chapter 5 of this thesis, generalized to include trimolecular reactions. So one toehold exchange or one 2-toehold-mediated 4-way strand exchange is one reaction, as is the cooperative 4-way strand exchange shown in Figure 4.2; note that using that mechanism to exchange e.g. $A(t, s; m) \rightleftharpoons R(a_1, a_2; n^*)$ takes 4 such reactions.) The $O(n)$-toeholds scheme takes only 14 reactions for $A + B \to C + D$, but with one universal long domain it takes 3 toeholds per species and 6 per reaction, which may run out of design space for large CRNs. Also, 14 reactions is still much more than 4. These pathways are not provably optimal; we suspect they can be reduced to less than 14 and 30, but still more than 4.

The increase in number of reactions to implement $A + B \to C + D$ may just be a cost of using 2-stranded complexes. The fundamental question is, given a complex of a certain size, how much information can it store? How can complexes meant to represent A, C, and an E from another reaction all present different enough open and bound domains that none can undergo a reaction meant for a different one? With 3-stranded complexes and toehold exchange, the long domain identity and open toehold does this very efficiently. With 2-stranded complexes and 4-way strand exchange, we use pairs of toehold identity to represent signal identity, which means we need extra reactions to (a) change the toehold identity one strand at a time, and (b) ensure that intermediates of different pathways don't try to pass through the same toehold combination. The impossibility proofs of Chapter 5 of this thesis start to examine the theoretical question of how much distinguishing information one complex can hold.

Another aspect worth mentioning is the focus on motifs before building up CRN implementations. We argued that each of the 5 motifs has certain abstract behaviors, and that larger systems such as CRN implementations can be thought of in terms of those behaviors. When building large systems, it is much easier if one can build mid-sized building blocks out of the fundamental units, then build larger systems out of the mid-sized building blocks. Motifs take that role between fundamental DSD steps (bind, unbind, 3-way branch migration, 4-way branch migration) and systems on the scale of CRN implementations. To the extent that we were able to describe our CRN implementations in terms of the motifs rather than in terms of the underlying DSD steps, this approach should be considered for future DSD system design.

Chapter 5

IMPOSSIBILITY OF SUFFICIENTLY SIMPLE CHEMICAL REACTION NETWORK IMPLEMENTATIONS IN DNA STRAND DISPLACEMENT

5.1 Perspective

Whenever one wants to optimize something, it helps to know when to stop. Is there some DSD system so optimized for its task (for example, implementing a given CRN) that there's no possible better one? Or at least, are there multiple "best" systems with different tradeoffs relative to each other, each one best for its specific task? There must be, since typically "best" means some variant of "smallest", but how can we find them, or know when we have one? I don't have the final answer, but I have a preliminary result that suggests a direction to search. In this chapter I show that a general CRN-to-DSD implementation scheme that satisfies a certain set of desirable but restrictive criteria cannot exist.

Why are we interested in proving what can't be done? The two main reasons are, first, to develop the theory of DSD, and second, that knowing what can't be done helps us do what can be done. In computational complexity, we feel we have a good grasp on the difficulty of a problem when we can say it's *complete* for a certain complexity class. Intuitively, this means we have a way to solve the problem, we know how complex our way is, and we have proven that there's no less complex way to solve it. In DSD systems, we might prove that general CRN implementations cannot be done without at least one of mechanisms A, B, C, and D, then present an implementation scheme that uses mechanism A but not B, C, or D. In fact, in Chapter 4 I discussed two new CRN implementations, each one having a desirable feature ($O(1)$ toeholds or using only bimolecular reactions) that the other lacked, as well as a number of other desirable features. In this chapter, I prove that a general CRN implementation scheme with all of those desirable features, including both $O(1)$ toeholds and using only bimolecular reactions, cannot be done with DSD. Thus the lower bound proven in this chapter is a tight lower bound, and suggests that the schemes from Chapter 4 are in some sense optimal.

So how does one prove statements about DSD systems? The first step is to formalize what a DSD system is. From one point of view, “DNA strand displacement” is just a description of some things DNA strands do, which can be experimentally measured [22, 87] and used to build circuits that experimentally work [19, 67]. However, in order to more efficiently program with DSD, various researchers built models such as reaction enumerators [36, 45], and with those models come assumptions of how DSD works. The work in this chapter is based on a specific model as published by Petersen et al. [53], which assumes four basic DSD reactions: binding, toehold unbinding, 3-way branch migration, and 4-way branch migration. “All models are false, but some are useful,” as the saying goes; a formalization of DSD is not the underlying physical behavior of DNA strands, but it is a formal system about which some statements can be proven. To the extent that it approximates the behavior of the DNA strands, those statements may apply to the DNA strands as well.

If in general one can prove statements about the formalized DSD system, which specific things did I prove? I proved that one cannot design a systematic CRN-to-DSD implementation scheme with all of the following properties: correct according to modular CRN bisimulation; uses 4-way but not 3-way branch migration; uses a constant number of toehold domains; does not use “effectively trimolecular” DSD mechanisms; uses only reversible DSD reactions; and uses only 1- or 2-stranded fuels. (Each of these concepts, including systematic implementation scheme, has a formal definition in terms of the DSD model.) Most of these conditions are not necessary for a functioning CRN-to-DSD implementation scheme, but each has a reason to be desirable: “systematic and correct according to modular CRN bisimulation” is a formalization of the intuitive concept of “correct implementation scheme”; the number of distinct toehold domains is limited by physical properties of DNA; trimolecular mechanisms are less reliable at lower concentration; reversible mechanisms consume less fuel; and 2-stranded fuels are potentially more robust as described in Chapter 4. Regarding 3-way and 4-way branch migration, I intended this result to be an intermediate step in determining whether such an implementation scheme was possible using both DSD mechanisms, but as a first step I proved it impossible without 3-way branch migration. It is also worth noting that both of the implementation schemes we presented in Chapter 4 satisfy all but one of the above conditions: one uses a scaling number of toehold domains, while the other uses a trimolecular mechanism.

I also think this proof can teach us more about DSD systems than just its result. In general the method of proof I used in this chapter is to categorize all combinations of the four DSD reactions and show that none of the categories can transform DNA complexes in a certain way, thus making that transformation impossible. For example, as an intermediate step I prove a locality theorem: that for parts of a DNA complex on different sides of a 4-way junction to affect each other, the reactions must go through the 4-way junction, and cannot do so while leaving the junction unchanged. That is, no sequence of the four DSD reactions will have the effect of making a change on one side of the junction while leaving the junction itself as it was, unless that change can be made independently of any other side. Any of the intermediate steps, but in particular this locality theorem, may be interesting facts in their own right, and/or may be useful as parts of other proofs regarding DSD systems.

Finally, there's an implication of impossibility proofs such as this that is hard to define, but I argue is important. I often say that the best known technique for designing DSD systems is no technique at all—ad-hoc brainstorming ideas until you come up with one that does what you want or you give up. In other words, designing DSD circuits is more of an art than a science: there are no design techniques that can be explained, well-defined, and reproduced, but there are intuitive understandings that make people more or less skilled at designing a DSD system for a given task. Statements such as the locality theorem come with an intuitive understanding, both of the statement itself and of the way it was proven. The more such intuitive understandings can be communicated and spread, the more people will be able to design DSD systems with the higher levels of skill, and the more quickly someone new to the field can reach the higher levels of intuition. In the long term, we may even be able to find a more systematic way of designing DSD systems, if we can gain enough formal understanding.

The remainder of this chapter is a slightly modified version of the following previously published work:

> Robert F. Johnson. Impossibility of sufficiently simple chemical reaction network implementations in DNA strand displacement. In Ian McQuillan and Shinnosuke Seki, editors, *Unconventional Computation and Natural Computation*, pages 136–149. Springer International Publishing, 2019. ISBN 978-3-030-19311-9. doi: 10.1007/978-3-030-19311-9_12.

Contributions: RFJ formulated the question of impossibility, proved the theorems, and wrote the manuscript.

Abstract

DNA strand displacement (DSD) has recently become a common technology for constructing molecular devices, with a number of useful systems experimentally demonstrated. To help with DSD system design, various researchers are developing formal definitions to model DNA strand displacement systems. With these models a DSD system can be defined, described by a Chemical Reaction Network, simulated, and otherwise analyzed. Meanwhile, the research community is trying to use DSD to do increasingly complex tasks, while also trying to make DSD systems simpler and more robust. I suggest that formal modeling of DSD systems can be used not only to analyze DSD systems, but to guide their design. For instance, one might prove that a DSD system that implements a certain function must use a certain mechanism. As an example, I show that a physically reversible DSD system with no pseudoknots, no effectively trimolecular reactions, and using 4-way but not 3-way branch migration, cannot be a systematic implementation of reactions of the form $A \rightleftharpoons B$ that uses a constant number of toehold domains and does not crosstalk when multiple reactions of that type are combined. This result is a tight lower bound in the sense that, for most of those conditions, removing just that one condition makes the desired DSD system possible. I conjecture that a system with the same restrictions using both 3-way and 4-way branch migration still cannot systematically implement the reaction $A + B \rightleftharpoons C$.

5.2 Introduction

DNA strand displacement (DSD) has become a common method of designing programmable *in vitro* molecular systems. DSD systems have been designed and experimentally shown to simulate arbitrary Chemical Reaction Networks (CRNs) [6, 19, 66, 67] and some polymer systems [57], implement large logic circuits [55, 72], conditionally release a cargo [27], sort objects [71], perform computation on a surface [56], and a number of other tasks [87]. Most DSD systems are based on the 3-way strand displacement mechanism, but a number of interesting devices based on a 4-way strand exchange mechanism have been shown [18, 22, 35, 77]. More complex tasks may require combinations of these two mechanisms, such as a 3-way-initiated 4-way strand exchange mechanism used in Surface CRNs [56]. Meanwhile, simple DSD mechanisms such as seesaw circuits [55] have been found to function with more robustness to uncontrollable conditions, compared to more complex mechanisms [72]. The ideal case is, of course, to find as simple a DSD mechanism as possible to accomplish the desired task.

To help with design and analysis of DSD systems, a number of researchers are developing techniques to formally define and analyze the behavior of DSD systems. Reaction enumerators formally define a set of reaction types that DNA strands are assumed to do [36, 45, 53]. Given that, a reaction enumerator will take a set of strands and complexes and enumerate a CRN, which may have finite or countably infinite species and reactions, describing the behavior of a DSD system initialized with the given complexes. Formal verification methods of CRN equivalence define whether a given implementation CRN is a correct implementation of a given formal CRN [40, 46, 64]. Thus all the tools necessary are available to ask, given a formal CRN and a DSD implementation, is it correct? The Nuskell compiler, for example, combines all of the above tools to answer exactly that question [4].

I suspect that this level of formal analysis can prove that certain tasks in DSD cannot be done without certain mechanisms, or certain tasks require a certain minimum level of complexity. As an example of this, I chose general CRN implementation schemes as the task, using CRN bisimulation [40] as the measure of correctness. Current CRN implementation schemes require high initial concentrations of “fuel” complexes using 3 or more strands [6, 57, 66], while the seesaw gates that showed such robustness to uncertainty use only single strands and 2-stranded complexes [55, 72]. Thus, I investigated whether arbitrary CRN implementations could be made using only 2-stranded signal and fuel complexes. To further probe the limits of DNA strand displacement, I investigated DSD systems with additional restrictions: the system should implement multiple CRN reactions in a systematic way (as existing CRN implementation schemes do [6, 57, 66]); the number of “short” or “toehold” domains (which is bounded by thermodynamics) should not increase as the number of reactions increases; the system should work under Peppercorn’s condensed semantics [36] (excluding trimolecular mechanisms such as cooperative hybridization); and reversible formal reactions should be implemented by physically reversible DSD mechanisms. Under those restrictions I prove that, using only 4-way strand exchange (excluding 3-way strand displacement), the reactions $A \rightleftharpoons B$ and $C \rightleftharpoons D$ cannot be implemented without crosstalk. While 3-way strand displacement can easily implement those reactions (such as the seesaw gates), it has difficulty implementing bimolecular reactions such as $A + B \rightleftharpoons C$ with only 2-stranded fuels. I conjecture that, allowing both 3-way and 4-way branch migration with all the restrictions above, the reaction $A + B \rightleftharpoons C$ cannot be implemented systematically without crosstalk; this result is intended to be the first part of that proof.

In the following sections, I first formalize the concept of DSD system and the reaction rules I use. With that formalization, I prove a locality theorem for reversible, non-pseudoknotted DSD systems without 3-way branch migration, which will be essential to the remaining proofs. I then introduce the concept of condensed reaction as formalized by Peppercorn [36], and show that in the above type of system with 2-stranded reactants there is in fact only one type of condensed reaction possible, the *two-toehold-mediated reversible 4-way strand exchange* or *2-r4 reaction*. Finally, I formalize the concept of *systematic CRN implementation*, which existing DSD implementations of CRNs satisfy [6, 57, 66], and the *information-bearing set* of domains which, in a systematic implementation, identify which species a DNA complex represents. I then show that the 2-r4 reaction cannot add an information-bearing domain to a complex that did not already have a copy of that domain. This implies that there is no way to build a pathway from a species A to another species B with distinct information-bearing domains, which completes the proof.

5.3 Formalizing DNA Strand Displacement

The syntax of a DSD system is defined by a set of *strands*, which are sequences of *domains*, and grouping of strands into *complexes* by making bonds between pairs of complementary domains. The semantics of the system will be defined by *reaction rules*, each of which says that complexes matching a certain pattern (and possibly some predicate) will react in a certain way, and defines the products of that reaction. Starting from a given set of complexes, enumerating species and reactions by iteratively applying reaction rules produces a possibly infinite Chemical Reaction Network, which models the behavior of the DSD system.

Definition 5.3.1 (Petersen et al. [53]). The syntax of a DSD system is defined by the following grammar, in terms of *domain names* x, y, z and *bonds* i, j, k.

$$d ::= x \quad \text{or} \quad x^* \qquad \text{(Domain)}$$

$$o ::= d \quad \text{or} \quad d!i \qquad \text{(Domain instance)}$$

$$S ::= o \quad \text{or} \quad o\ S \qquad \text{(Sequence of domains)}$$

$$P ::= S \quad \text{or} \quad S \mid P \qquad \text{(Multiset of strands)}$$

A *complex* is a multiset of strands P that is connected by bonds. Complexes are considered equal up to reordering of strands and/or renaming of bonds.

In this paper I use a specific set of reaction rules: binding b, toehold unbinding u, 3-way strand displacement m_3, and 4-way strand exchange m_4. To define these rules, I use the following assumptions and notation. Each domain x has a complementary domain x^*, such that $(x^*)^* = x$. Each domain x is either *short* ("toehold") or *long*, and x^* is short iff x is short. Bonds are between exactly one domain instance and one instance of its complement: if $d!i$ appears in some P, then P has exactly one $d^*!i$ and no other instances of i.

A *pseudoknot* is a pair of bonds that cross over each other. Formally, a complex is *non-pseudoknotted* if, for some ordering of its strands, for every pair of bonds i, j, the two instances of i appear either both between or both outside the two instances of j. This *non-pseudoknotted ordering*, if it exists, is unique up to cyclic permutation [24]. Pseudoknots are poorly understood compared to non-pseudoknotted DSD systems; for this reason, DSD enumerators such as Peppercorn and Visual DSD often disallow them [36, 45]. For the same reason, I define the reaction rules in such a way that no pseudoknot can form from initially non-pseudoknotted complexes.

Definition 5.3.2 (Reaction rules)**.** The reactions of these DSD systems come from the following rules:

1. Binding (b): $x, x^* \to x!i, x^*!i$ if the product is non-pseudoknotted.
2. Toehold unbinding (u): $x!i, x^*!i \to x, x^*$ if x is short and i is not anchored. A bond i is *anchored* if it matches

$$x!i\ y!j, y^*!j\ x^*!i \text{ or } y!j\ x!i, x^*!i\ y^*!j.$$

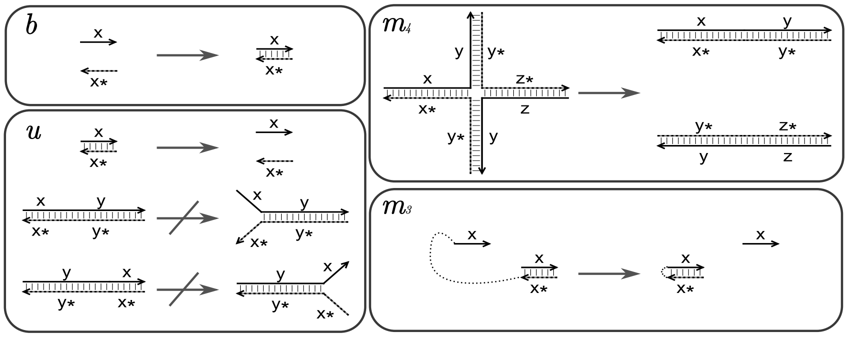


Figure 5.1: The reaction rules of Definition 5.3.2. Dotted line in m_3 indicates that the reactant must be one complex; the products of m_3 and m_4 can be either one complex or two.

3. 4-way branch migration (m_4): $x!i\ y!j, y^*!j\ z^*!k, z!k\ y!l, y^*!l\ x^*!i$
$\rightarrow x!i\ y!j, y^*!l\ z^*!k, z!k\ y!l, y^*!j\ x^*!i.$

4. Remote-toehold 3-way strand displacement (m_3):
$x, x!i, x^*!i \rightarrow x!i, x, x^*!i$ if the reactant is one complex and the product is non-pseudoknotted.

A *Chemical Reaction Network* is a pair $(\mathcal{S}, \mathcal{R})$ of a set of abstract species $\mathcal{S}$ and a set of reactions between those species $\mathcal{R}$. The *DSD system enumerated from* an initial multiset of strands P and a set of rules is the smallest DSD system such that any complex in P is a species in $\mathcal{S}$, any application of one of the rules to reactants in $\mathcal{S}$ is a reaction in $\mathcal{R}$, and its products are species in $\mathcal{S}$.

I use comma-separated sequences instead of |-separated sequences in the above definition to indicate that those sequences may each be part of a larger strand, and may or may not be on the same strand as each other. I use comma-separated sequences for the same purpose throughout the paper, which given the nature of the proofs is much more common than knowing the full sequence of a strand. Although in general checking whether a complex is non-pseudoknotted is hard due to having to find the non-pseudoknotted order, given a complex or complexes with known non-pseudoknotted order(s) checking whether an additional bond makes a pseudoknot (as in the b and m_3 conditions) is easy.

These reaction rules are similar, but not identical, to those from Petersen et al. [53]. The main difference is that Petersen's u rule counts 4-way (and n-way) junctions as "anchored" for the purpose of prohibiting u reactions. Based on Dabby's experiments and energy models of toehold-mediated 4-way branch migration [22] and the probe designed by Chen et al. using reversible 4-way branch migration [18], there is evidence that 2 toeholds separated by a 4-way junction *can* dissociate, and I would like to model and design DSD systems using this mechanism. As the binding of those toeholds is already modeled by two separate b reactions, and I am interested in physical reversibility of these systems, I modeled the unbinding as two separate u reactions, allowing u at anything but an unbroken helix. The other difference is that in the "sufficiently simple system" that I would like to prove cannot accomplish certain tasks, the m_4 reaction can only happen at a 4-way junction, while Petersen's equivalent rule could happen at larger junctions. All of these reactions are possible in Peppercorn [36].

Observe that a b reaction is reversible (by u) if the domain x is short and the bond formed is not anchored; a u is always reversible (by b); an m_3 is reversible (by m_3) if it does not separate complexes; and a m_4 is reversible (by m_4) if bonds j and l are part of a new 4-way junction. The impossibility proof I wish to present here deals with *reversible, 4-way-only DSD systems*; a DSD system is *reversible* if it contains only reversible reactions, and a DSD system is *4-way-only* if it contains no m_3 reactions. In such a system, largely because of the no-pseudoknots condition, I can prove an important locality theorem: that if a complex with a 4-way junction can, via unimolecular reactions, break the 4-way junction and eventually reach a different state with the same 4-way junction reformed, then that initial complex can reach the same final state (via unimolecular reactions) without ever breaking the 4-way junction. This, in a certain sense, one-way flow of information limits the type of reactions that can happen in such a system.

Theorem 5.3.1. *In a reversible, 4-way-only DSD system, consider a complex P containing a 4-way junction, where P is non-pseudoknotted. Assume $P \rightleftharpoons P'$ via a trajectory of unimolecular reactions, where P' contains the same 4-way junction (but may differ elsewhere). Then $P \rightleftharpoons P'$ via a trajectory of unimolecular reactions, not longer than the original trajectory, with no reaction changing the bonds within the 4-way junction.*

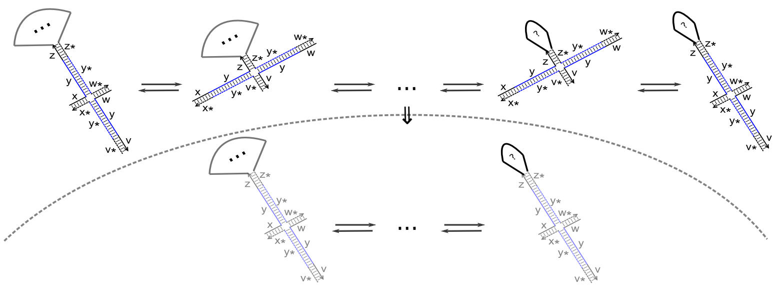


Figure 5.2: Theorem 5.3.1 states roughly that if some change outside of a four-way junction can occur after which the four-way junction is reformed, then the same change can occur without breaking the four-way junction.

Proof. This theorem assumes an initial complex P, with no details specified except the existence of a 4-way junction:

$$x!i\ y!j, \qquad y^*!j\ z^*!k, \qquad z!k\ w!l, \qquad w^*!l\ x^*!i.$$

This theorem also assumes some sequence of reactions by which $P \rightleftharpoons P'$, again with no details specified except that the system is reversible and 4-way-only, the path contains only unimolecular reactions, and the result P' has the same 4-way junction made out of the same strand(s). I focus on three steps in the given trajectory: the first reaction that breaks the 4-way junction; the first reaction afterwards that requires that reaction to have happened; and the reaction that eventually reforms the original 4-way junction. (If the junction breaks and reforms multiple times, apply the theorem to each such sub-trajectory.) If those reactions do not exist, the result is trivial: if the junction never breaks, then the given trajectory is the desired trajectory, and if the first reaction that requires the break is the reaction that reforms the junction, then removing those two gives either the desired trajectory or (if what used to be the second reaction that breaks the 4-way junction is now the first) a shorter trajectory to which this theorem can be applied. Each of those three reactions has a limited number of possibilities, and I show that each combination of possibilities produces either a contradiction or the desired pathway.

In a reversible, 4-way-only system, a reaction that breaks a bond must be u or m_4. If the junction is first broken by u, then except for its reverse b, the only reaction that can depend on a u is a b with one of the newly opened domains:

$$x, x!i, x^*!i \xrightarrow{u} x, x, x^* \xrightarrow{b} x!i, x, x^*!i.$$

However, observe that this pair of reactions is equivalent to (and implies the possibility of) an m_3 reaction, thus the DSD system is not 4-way-only.

If the junction is first broken by m_4, to be reversible it must form a new 4-way junction with two smaller stems and two larger stems:

$$x!i\ y!j\ u!g, \qquad u^*!g\ y^*!l\ z^*!k, \qquad z!k\ y!l\ v!h, \qquad v^*!h\ y^*!j\ x^*!i.$$

To reform the junction, bonds j and l must be broken, specifically by u or m_4. If one is broken by u, then since both have domain identity y this allows an m_3 reaction with the other, contradicting 4-way-only. An m_4 reaction that is not the reverse of the original m_4, while it would break one of the bonds, would produce another bond that needs to be broken to reform the original junction, and to which this argument could be applied (formally, this is induction on the size of the remaining DNA complex). The only remaining possibility is that the original junction is reformed by the reverse m_4 of the one that broke it, implying that at that time the new (j, g, l, h) 4-way junction must be present.

I treat the above as a proof by induction on the length of the given trajectory by which $P \rightleftharpoons P'$. The base case is a trajectory of length 0, $P = P'$, which does not break the 4-way junction and thus is the desired trajectory. In a given trajectory of length $n > 0$, let P_1 be the state after the m_4 breaking the 4-way junction and P_1' the state before the reverse m_4; both have the same (j, g, l, h) 4-way junction and $P_1 \rightleftharpoons P_1'$ by a trajectory of length at most $n - 2$ satisfying the assumptions of this theorem. Thus $P_1 \rightleftharpoons P_1'$ by a trajectory not longer than the original in which no reaction breaks the (j, g, l, h) 4-way junction. But the only reactions that can require the original m_4 are a u or m_4 involving bonds j, g, l, and/or h, thus none of the new trajectory requires that m_4; and $P \rightleftharpoons P'$ by removing both m_4's and replacing the trajectory between them by the new $P_1 \rightleftharpoons P_1'$ trajectory. Because this new trajectory is valid both before and after the m_4 reaction exchanging bonds j and l ("j, l-m_4"), no reaction in it can break bonds i or k: a u reaction would be impossible post-j, l-m_4 as the pattern $x!i\ y!j, y^*!j\ x!i$ anchors i and similarly bond l anchors k, while an m_4 reaction on i or k would form a 6-way junction and thus be irreversible before the j, l-m_4. □

5.4 The 2-r4 Condensed Reaction

Grun et al. in Peppercorn used the concept of a *condensed reaction*, which models a sequence of multiple reaction rules as one reaction [36]. They divide

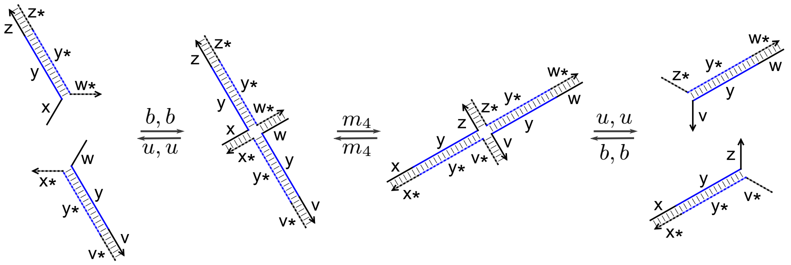


Figure 5.3: The two-toehold-mediated reversible 4-way branch migration (2-r4) condensed reaction mechanism. For conciseness, the first and last two b, b and u, u detailed steps are drawn together.

detailed reactions into “fast” and “slow” reactions, assume that no slow reaction can happen while any (productive sequence of) fast reactions can happen, and treat a single slow reaction followed by multiple fast reactions as a single, “condensed” reaction. The usual division, which I use, is that all unimolecular reactions are fast and all bimolecular reactions are slow.

Definition 5.4.1 (Grun et al. [36])**.** Take as given a DSD system $(\mathcal{S}, \mathcal{R})$ and a division of reactions in that DSD system into “fast” unimolecular and “slow” bimolecular reactions. A *resting set* (or *resting state*) is a set of complexes connected by fast reactions, and none of which have outgoing fast reactions with products outside the resting set.

A *condensed reaction* is a reaction of resting sets, such that for some multiset of representatives of the reactant resting sets, there is a sequence of one slow reaction followed by zero or more fast reactions that produces a multiset of representatives of the product resting sets. The *condensed DSD system* corresponding to $(\mathcal{S}, \mathcal{R})$ is $(\hat{\mathcal{S}}, \hat{\mathcal{R}})$ where $\hat{\mathcal{S}}$ is the set of resting states of $\mathcal{S}$ and $\hat{\mathcal{R}}$ is the set of condensed reactions. A *reversible, condensed DSD system* is a condensed DSD system where every reaction in $\mathcal{R}$ and condensed reaction in $\hat{\mathcal{R}}$ is reversible.

A given condensed reaction can correspond to multiple equivalent pathways of detailed reactions. To distinguish between detailed and condensed reactions, I will often use the word “step” to refer to a detailed reaction and “reaction” to refer to a condensed reaction. If all unimolecular steps are fast and all bimolecular steps slow, then a condensed reaction must be bimolecular and begin with a bimolecular b step. In a reversible system, all condensed reactions must have exactly two products and end with a u that separates complexes: one product can always reverse back to its reactants; with 3 or more products the reverse would not be a condensed reaction; and any steps after that u that do not separate complexes are reversible and thus are steps within a resting set.

An important example condensed reaction is the *two-toehold-mediated reversible 4-way strand exchange reaction*, or *2-r4* reaction, shown in Fig. 5.3.

Definition 5.4.2. A *2-r4 reaction* is a reversible condensed reaction of which one representative pathway is the following sequence of detailed reactions:

$$\begin{aligned}
& x\ y!j\ z!g, z^*!g\ y^*!j\ w^*, w\ y!l\ v!h, v^*!h\ y^*!l\ x^* \\
\underset{u}{\overset{b}{\rightleftharpoons}}\ & x!i\ y!j\ z!g, z^*!g\ y^*!j\ w^*, w\ y!l\ v!h, v^*!h\ y^*!l\ x^*!i \\
\underset{u}{\overset{b}{\rightleftharpoons}}\ & x!i\ y!j\ z!g, z^*!g\ y^*!j\ w^*!k, w!k\ y!l\ v!h, v^*!h\ y^*!l\ x^*!i \\
\underset{m_4}{\overset{m_4}{\rightleftharpoons}}\ & x!i\ y!j\ z!g, z^*!g\ y^*!l\ w^*!k, w!k\ y!l\ v!h, v^*!h\ y^*!j\ x^*!i \\
\underset{b}{\overset{u}{\rightleftharpoons}}\ & x!i\ y!j\ z!g, z^*!g\ y^*!l\ w^*!k, w!k\ y!l\ v, v^*\ y^*!j\ x^*!i \\
\underset{b}{\overset{u}{\rightleftharpoons}}\ & x!i\ y!j\ z, z^*\ y^*!l\ w^*!k, w!k\ y!l\ v, v^*\ y^*!j\ x^*!i
\end{aligned}$$

Any reversible condensed reaction that can be written as a sequence similar to the above, with more than one m_4 step across an unbroken sequence of migration domains, is also a 2-r4 reaction.

Recall that the comma-separated sequence notation means that the sequences above may be only part of their strands, and the complex may contain additional strands not mentioned. Note that the reverse pathway is also a 2-r4 reaction, hence the phrase "reversible condensed reaction". An important feature of the 2-r4 reaction is that if its reactants are at most 2-stranded, so are its products.

Lemma 5.4.1. *A 2-r4 reaction where one of the reactants is a single strand is impossible. In any 2-r4 reaction where the reactants are both 2-stranded complexes, the products are both 2-stranded complexes.*

Proof. The initial and final state must each be 2 separate complexes; in particular, the patterns $x\ y\ z$, $v^*\ y^*\ x$, $w\ y\ v$, and $z^*\ y^*\ w^*$ must be on 4 separate strands. If both of the initial complexes are 2-stranded, then those 4 are the only strands involved, and each product has 2 of them. □

In addition to the restrictions of a reversible, condensed, 4-way-only DSD system, I would like to consider systems where all initial complexes have at most 2 strands each. In this case Lemma 5.4.1 suggests, and Theorem 5.4.1 proves, that all complexes involved will have at most 2 strands. Using Theorem 5.3.1, I show via a trace rewriting argument that any condensed reaction between 2-stranded complexes in a reversible, condensed, 4-way-only system is a 2-r4 reaction. As a first step in that proof, I observe that only b and u steps cannot make a nontrivial condensed reaction.

Lemma 5.4.2. *In a reversible, condensed, 4-way-only system, if a pathway of some condensed reaction with non-pseudoknotted reactants consists of only b and u steps, then the condensed reaction is trivial.*

Proof. Observe that any two consecutive u steps can happen in any order, and any u step that can happen after a b step can happen before it, except the reverse of that b (which can thus be removed). Then any pathway matching the assumptions of this lemma is equivalent to a pathway where all u steps happen before all b steps. Such a path is either trivial or involves one of the reactants separating unimolecularly, in which case that reactant was not in a resting state. □

Theorem 5.4.1. *In a reversible, condensed, 4-way-only system, any condensed reaction between non-pseudoknotted 2-stranded complexes is a 2-r4 reaction.*

Proof. (Sketch) Given a reversible pathway representing a condensed reaction from two reactant resting states to two product resting states, I show that that pathway can be "rewritten" into a pathway that represents the same condensed reaction and matches Definition 5.4.2.

Lemma 5.4.2 implies that an m_4 step, which does not eventually reverse itself, happens. That is, the first m_4 step in the reaction goes from

$$x!i\ y!j\ w!g, \qquad w^*!g\ y^*!j\ z^*!k, \qquad z!k\ y!l\ v!h, \qquad v^*!h\ y^*!l\ x^*!i$$

to

$$x!i\ y!j\ w!g, \qquad w^*!g\ y^*!l\ z^*!k, \qquad z!k\ y!l\ v!h, \qquad v^*!h\ y^*!j\ x^*!i.$$

If this m_4 was not possible in a resting state (and neither was an m_3 between what are now bonds j and l), it must be that bonds i and k were formed by inter-reactant b steps. Since these three steps cannot depend on any other b or u steps, there is an equivalent pathway where those bbm_4 are the first three steps. (The case where y is a sequence of more than one domain and this m_4 is more than one m_4 is still covered by this pattern.)

Looking ahead to the u step that separates complexes, separating bonds g and h by two u steps (if w and v are short) completes the 2-r4 pattern. Any other possible separation can be eliminated. If the entire (j, g, l, h) 4-way junction is on one product, with the other product coming from the g or h stems, then the appropriate reactant must have been either pseudoknotted or 3-stranded. If the other product comes from the j or l stems *without* breaking bonds j or l, then it could have separated without the j, l-m_4 step. If either of bonds j or l is broken, then this will either allow an m_3 reaction, allow an irreversible reaction involving 6-way junctions, or be equivalent to the j, l-m_4 step never happening. □

5.5 Chemical Reaction Network Implementations

Previous work on formal verification of CRNs allows one to define whether one CRN correctly implements another [40, 64], and combine that work with the above definitions of modeling a DSD system as a CRN to verify that a DSD system correctly implements a CRN [4]. The definition of "systematically correct" in Definition 5.5.2 is based on (modular) CRN bisimulation [40].

Most existing DSD-based CRN implementations are not intended to be an implementation of one specific CRN, but rather a general translation scheme to implement arbitrary (or at least a wide class of) CRNs [6, 57, 66]. A systematic implementation is one where (a) each species has a "signal complex", and the signal complexes corresponding to two different species are identical except for the identities of certain domains; and (b) similarly, the pathways by which formal reactions of the same "type" are implemented, are identical to each other except for the identities of some domains unique to the species involved and some domains unique to the reaction. Fig. 5.4 shows an example of this definition.

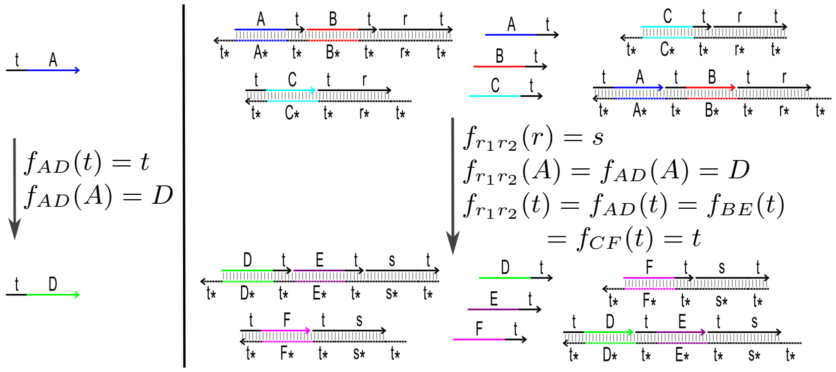


Figure 5.4: An example $O(1)$ toeholds systematic DSD implementation (Cardelli et al. [6]). Left: signal species s_A and s_D, with domain isomorphism f_{AD}. Right: fuels F_{r_1} and F_{r_2} for $r_1 = A + B \rightleftharpoons C$ and $r_2 = D + E \rightleftharpoons F$, and domain isomorphism $f_{r_1r_2}$.

Definition 5.5.1 (Systematic implementation)**.** A *domain isomorphism* is an injective partial function on domains f with $f(x^*) = f(x)^*$. Where P is a multiset of strands and f a domain isomorphism, let $P\{f\}$ be the multiset of strands obtained from P by replacing each d with $f(d)$ whenever $f(d)$ is defined.

A *DSD implementation* of a formal CRN $(\mathcal{S}, \mathcal{R})$ is, for each species $A \in \mathcal{S}$, a "signal" DSD complex s_A, and for each reversible or irreversible reaction $r \in \mathcal{R}$, a set of "fuel" complexes F_r. A DSD implementation is *systematic* if:

1. Given species A, B there is a domain isomorphism f_{AB} where $s_B = s_A\{f_{AB}\}$. If $A \neq B$ there is at least one domain d appearing in s_A with $f_{AB}(d) \neq d$.

2. If a domain d is in both s_A and s_B for $A \neq B$, then $f_{CD}(d) = d$ for all C, D.

3. Given reactions r_1, r_2, if there is a bijection ϕ on species such that $r_2 = r_1\{\phi\}$, then there is a domain isomorphism $f_{r_1r_2}$ where $F_{r_2} = F_{r_1}\{f_{r_1r_2}\}$. For one such ϕ, for each A in r_1, $f_{r_1r_2} = f_{A\phi(A)}$ wherever $f_{A\phi(A)}$ is defined.

A systematic DSD implementation is *O(1) toeholds* if, whenever d is a short domain, all $f_{AB}(d) = d$ and all $f_{r_1r_2}(d) = d$.

Where R is a multiset of species, the notation s_R means s_A for each $A \in R$.

Definition 5.5.2. Given a CRN $(\mathcal{S}, \mathcal{R})$ and a systematic DSD implementation, let $(\mathcal{S}', \mathcal{R}')$ be the (detailed or condensed) DSD system enumerated, plus reactions $\emptyset \to F_r$ for each $r \in \mathcal{R}$. The implementation is *systematically correct* if:

1. There is an *interpretation* m mapping species in $\mathcal{S}'$ to multisets of species in $\mathcal{S}$, with each $m(s_A) = A$ and $x \in F_r \Rightarrow m(x) = \emptyset$.
2. For each $r' = R' \to P' \in \mathcal{R}'$, either $m(R') = m(P')$ (r' is "trivial") or $m(R') \to m(P')$ is some reaction $r \in \mathcal{R}$.
3. For each $r = R \to P \in \mathcal{R}$, there is a pathway containing exactly one nontrivial reaction, from $F_r + s_R$ to some $W_r + s_P$, with $x \in W_r \Rightarrow m(x) = \emptyset$.
4. For each $x \in \mathcal{S}'$, there is a pathway of trivial reactions from x to $s_{m(x)}$.

I now consider whether the 2-r4 reaction can be used to construct a systematically correct implementation of $A \rightleftharpoons B$ and $C \rightleftharpoons D$ with $O(1)$ toeholds. I define the *information-bearing set* or *infoset* of an intermediate complex in $A \rightleftharpoons B$ as the long domains that, if changed appropriately, would make it "act like" $C \rightleftharpoons D$, as shown in Fig. 5.5. The infosets of s_A and s_B must be disjoint and nonempty. I show that the 2-r4 reaction can't add long domains to the infoset.

Definition 5.5.3. Consider a systematically correct implementation of the two reversible reactions $r_1 = A \rightleftharpoons B$ and $r_2 = C \rightleftharpoons D$. Let f_{AC}, f_{BD}, and $f_{r_1 r_2}$ be the appropriate domain isomorphisms as in Definition 5.5.1. Where x is a complex with $m(x) = A$ or $m(x) = B$, there is a pathway by which x produces s_B. Define the *information-bearing set* or *infoset* $I(x)$ to be the smallest set of domains for which, where x' is x with each domain $d \in I(x)$ replaced by $f_{r_1 r_2}(d)$, x' can do the following: follow the pathway by which x produces s_B but using fuels from F_{r_2} to produce some complex s'_D; s'_D then mimics $D \to C$ to produce some s'_C; s'_C then mimics $C \to D$ to produce s_D.

(The use of s'_C and s'_D is a technical detail to say that domains whose identity exists but does not matter are not in the infoset.)

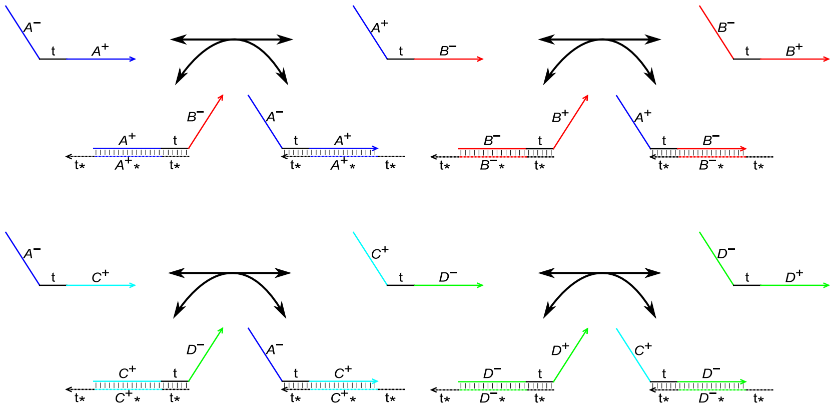


Figure 5.5: In this systematic implementation of $A \rightleftharpoons B$ and $C \rightleftharpoons D$ using 3-way strand displacement, replacing A^+ with C^+ in s_A (left) lets it follow the $C \rightleftharpoons D$ pathway (bottom) instead of the $A \rightleftharpoons B$ pathway (top). Thus, the infoset $I(s_A) = \{A^+\}$.

Lemma 5.5.1. *In a reversible, condensed, 4-way-only systematically correct $O(1)$-toehold DSD implementation of $A \rightleftharpoons B$ and $C \rightleftharpoons D$, let $x + f \rightleftharpoons x' + f'$ be a 2-r4 reaction on the pathway from s_A to s_B. Without loss of generality say $m(x)$ and $m(x')$ are each either A or B, and $m(f) = m(f') = \emptyset$. If x and f are non-pseudoknotted complexes with at most 2 strands each, then $I(x') \subset I(x)$.*

Proof. (Sketch) Observe that if x, f, x', or f' contains inter-strand long domain bonds not in the 2-r4 reaction pattern, then the 2-r4 reaction cannot happen.

Any domain $d \in I(x')$ must eventually be involved in a reaction; since $O(1)$ toeholds implies d is a long domain, the next such reaction must be an m_4 reaction. To participate in an m_4 reaction, it must be bound; for this to be a necessary reaction (i.e. not eliminated by Theorem 5.3.1; detailed proof omitted), its bound complement must be on a distinct strand. Since x' cannot have inter-strand bonds not mentioned in the 2-r4 reaction pattern, d can only be the y domain in Definition 5.4.2; thus $d \in I(x)$. □

Given Theorem 5.4.1 and Lemma 5.5.1, the desired result is trivial.

Theorem 5.5.1. *No reversible, condensed, 4-way only systematically correct $O(1)$-toehold DSD implementation of $A \rightleftharpoons B$ and $C \rightleftharpoons D$ where each signal and fuel complex has at most 2 strands can exist.*

5.6 Discussion

The above proofs show that a reversible, condensed, 4-way-only DSD system with at most 2-stranded inputs cannot be a systematically correct with $O(1)$ toeholds implementation of multiple, independent reactions of the form $A \rightleftharpoons B$, and therefore cannot implement more complex CRNs. I proposed that a proof of "Task X cannot be done without mechanism M" suggests "In order to do task X, use mechanism M". For most of the restrictions I investigated, removing just that one restriction makes $A \rightleftharpoons B$, or even $A + B \rightleftharpoons C$ (which can implement arbitrary reversible reactions), possible. For example, removing the 4-way-only restriction allows seesaw gates, which implement $A \rightleftharpoons B$ [55]. Existing CRN implementations are made with 3-stranded fuels [6, 66] and I suspect a similar mechanism can work using 2-r4 reactions instead of 3-way strand displacement reactions. The 2-r4 reaction *is* $A + B \rightleftharpoons C + D$ if A, B, C, and D are identified by combinations of toeholds, violating the $O(1)$ toeholds condition (Chapter 4 of this thesis). Finally, a cooperative 4-way branch migration mechanism (Chapter 4) implements $A + B \rightleftharpoons C$ but is not a condensed reaction. In that sense, this result is a tight lower bound.

I conjectured that a system with the same restrictions but allowing 3-way branch migration cannot implement $A + B \rightleftharpoons C$. Informally, 3-way strand displacement can easily implement $A \rightleftharpoons B$ [55], but has difficulty implementing "join" ($A + B \rightleftharpoons C$) logic without either 3-stranded fuels [6, 57, 66] or cooperative hybridization. Meanwhile, 4-way strand exchange can do join logic on $O(n)$ toeholds with the 2-r4 reaction (Chapter 4), but is provably unable to transduce long domains with $O(1)$ toeholds. In trying to combine 3-way and 4-way mechanisms I have found numerous pitfalls. I suspect that with 2-stranded fuels, combining 3-way and 4-way mechanisms will not gain the advantages of both, and $A + B \rightleftharpoons C$ under the above restrictions will remain impossible.

Acknowledgements. I thank Chris Thachuk, Stefan Badelt, Erik Winfree, and Lulu Qian for helpful discussions on formal verification and on two-stranded DSD systems. I also thank the anonymous reviewers of a rejected previous version of this paper for their suggestions, many of which appear in this version. I thank the NSF Graduate Research Fellowship Program for financial support.

5.7 Omitted Proofs

Two proofs were left in sketch form in order to fit the above into a conference paper format. They are presented below.

Proof of Theorem 5.4.1

Proof. Given an arbitrary sequence of steps representing a nontrivial condensed reaction between 2-stranded reactants, I show how that condensed reaction can be rewritten as the 2-r4 reaction from Definition 5.4.2, with any extra steps being unimolecular rearrangements within a resting set of either a reactant or a product.

Consider two DNA complexes, with nothing known of their structure except that each is in a resting state and has at most 2 strands. Consider a condensed reaction between them, that is, a pathway in a reversible, 4-way-only system that is a bimolecular b step, followed by any number of unimolecular b, u, and m_4, followed by a u step separating complexes. Since by Lemma 5.4.2 only b and u in a 4-way-only system can only produce a trivial condensed reaction, there must eventually be an m_4 step. It must be an m_4 step that never reverses itself (or otherwise re-forms its prior 4-way junction), because otherwise Theorem 5.3.1 would apply and remove it; since Theorem 5.3.1 produces a strictly shorter pathway when removing an m_4 and its reverse, this process must eventually terminate in a trivial condensed reaction or an m_4 that cannot be removed. Further, the m_4 that cannot be removed must be an m_4 step between the two reactant complexes, since all the initial b's and u's commute sufficiently that an m_4 within one reactant would be possible in the resting state. That is, the reaction goes from

$$x!i\ y!j\ w!g, \qquad w^*!g\ y^*!j\ z^*!k, \qquad z!k\ y!l\ v!h, \qquad v^*!h\ y^*!l\ x^*!i$$

to

$$x!i\ y!j\ w!g, \qquad w^*!g\ y^*!l\ z^*!k, \qquad z!k\ y!l\ v!h, \qquad v^*!h\ y^*!j\ x^*!i,$$

where i and k are bonds between the two reactants. (One bond between two originally separate complexes cannot make a 4-way junction, bonds g and h are necessary to be reversible but not necessary for the m_4 to happen, and if either bond j or bond l was not present in one of the reactants then that reactant could do an m_3 reaction.) This being the first necessary m_4 implies that bonds i and k were formed by b steps, and since b's and u's sufficiently commute I can assume that those two b steps followed by this m_4 were the first three steps of the (or an equivalent) pathway. If $v = w$, another m_4 may be possible; imagining that y is a sequence of 1 or more domains, e.g. $y = y_1\ y_2\ y_3$ (in which case $y!j = y_1!j_1\ y_2!j_2\ y_3!j_3$ and $y^*!j = y_1^*!j_1\ y_2^*!j_2\ y_3^*!j_3$, etc.), further m_4 steps can be described by the same pattern as above.

At this point consider the eventual u step separating 2 complexes, and which bonds are broken to allow their domains to be on separate product complexes. This "cut" can be divided into the following exhaustive cases: a cut "inside" the reaction, where the entire pattern above is in one product and the other product is from the stem of bond g or h; "outside" the reaction, where the other product is from the i or k stems; "between" the reaction where it separates bonds g and h; or "against" the reaction where it separates i and/or k. If w and v are both short, then two u steps on w and v is a cut between the reaction and also the completion of the 2-r4 pattern, since any other dissociations necessary for a cut between the reaction can be assumed to have happened in the resting state (again appealing to commutativity of u's, and that an m_4 cannot depend on a u); if this is not possible, then no cut between the reaction is possible. I will show that the other types of cuts all lead to a contradiction.

A cut "inside" the reaction is entirely within one reactant, and by the no-pseudoknots assumption implies that that reactant must have at least 3 strands: e.g. if it is in the g stem, then in the appropriate non-pseudoknotted order (that is, the order used when describing the 4-way junction above) the pattern $x!i\ y!j\ w!g$ must appear before the at least 1 strand that is separated, which must appear before the $w^*!g\ y^*!j\ z^*!k$ pattern, so those three must be separate strands. A cut inside the h stem is treated the same.

A cut "outside" the reaction, if it is possible after this m_4 reaction, is possible before it. Without loss of generality assume the cut occurs in the i stem; the same logic applies to the k stem. If bond i is never broken, then any interaction between the i stem and the (j, g, l, h) junction or any of the other stems would be a pseudoknot, so the sequence that leads to the cut is independent of whatever is going on on the other side of j. If bond i is broken by m_4 after the j, l-m_4 step, then that same m_4 would be possible but irreversible (making a 6-way junction) before the j, l-m_4 step. Bond i cannot be broken by u after the j, l-m_4 step without breaking bond j, and similarly bond j cannot be broken by u after the j, l-m_4 step without breaking bond i. This leaves the possibility that j is broken by an m_4 reaction, but *not* the j, l-m_4 reaction since that would invoke Theorem 5.3.1, meaning to match the m_4 pattern bonds g and h must be broken and replaced. If one is broken by u and replaced by b then that is equivalent to m_3, while if one is replaced by m_4 that would be irreversible as long as j has not yet changed. This eliminates the possibility of a cut "outside" the reaction, or at least eliminates the possibility that such a cut depends on the j, l-m_4 step. Since this contains the argument that bonds i and k cannot be broken while depending on the j, l-m_4 step, it also eliminates the possibility of a cut "against" the reaction. Thus I have shown that a 2-r4 condensed reaction may be possible, and eliminated all other possibilities. □

Proof of Lemma 5.5.1

The proof given in the main text omits only the justification that, for $d \in I(x')$, an instance of d must be bound to a complement on a distinct strand in x'. Since d is a long domain, the next reaction it is involved in must be an m_4 step; by Theorem 5.4.1, since the only condensed reactions are 2-r4 reactions, this must be either a 2-r4 reaction or a unimolecular rearrangement within a resting set. If it is a 2-r4 reaction, then d is bound to a complement on a distinct strand, as necessary for the proof in the main text.

If the next necessary step involving d is an m_4 step within a resting state, then either the entire m_4 reaction was within one strand, or the domain d is now bound to a complement on a distinct strand. In the first case, it cannot participate in any 2-r4 reactions (it doesn't match the pattern), contradicting the assumption that it is necessary for a future condensed reaction. In the second case, those strands must separate for a 2-r4 reaction to complete. As in the case for cuts "against" the reaction in the previous proof, if *one* side of the m_4 separates by u it enables an m_3; if *one* side separates by m_4 then it forms a 6-way junction; and if *both* of them separate simultaneously by m_4 then that is the previous m_4 reversing itself. In the last case, Theorem 5.3.1 applies (since all of this is preparation for a 2-r4 reaction, it must be all unimolecular), contradicting the assumption that the m_4 was necessary. So the d must be bound to a complement on another strand, and the proof from the main text follows.

BIBLIOGRAPHY

[1] 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.

[2] Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21:87–102, 2008.

[3] Marco Antoniotti, Carla Piazza, Alberto Policriti, Marta Simeoni, and Bud Mishra. Taming the complexity of biochemical models through bisimulation and collapsing: theory and practice. *Theoretical Computer Science*, 325:45–67, 2004.

[4] Stefan Badelt, Seung Woo Shin, Robert F. Johnson, Qing Dong, Chris Thachuk, and Erik Winfree. A general-purpose CRN-to-DSD compiler with formal verification, optimization, and simulation capabilities. In Damien Woods and Yannick Rondelez, editors, *DNA Computing and Molecular Programming*, volume 9818 of Lecture Notes in Computer Science, pages 232–248. Springer, 2017.

[5] Joseph Berleant, Christopher Berlind, Stefan Badelt, Frits Dannenberg, Joseph Schaeffer, and Erik Winfree. Automated sequence-level analysis of kinetics and thermodynamics for domain-level DNA strand-displacement systems. *Journal of the Royal Society Interface*, 15(149):20180107, 2018.

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

[7] Luca Cardelli. Morphisms of reaction networks that couple structure to function. *BMC Systems Biology*, 8:1–18, 2014.

[8] Luca Cardelli and Attila Csikász-Nagy. The cell cycle switch computes approximate majority. *Scientific Reports*, 2, 2012. doi: 10.1038/srep00656.

[9] Luca Cardelli and Gianluigi Zavattaro. On the computational power of biochemistry. In *Algebraic Biology*, pages 65–80. Springer, 2008.

[10] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Forward and backward bisimulations for chemical reaction networks. In Luca Aceto and David de Frutos Escrig, editors, *26th International Conference on Concurrency Theory (CONCUR 2015)*, volume 42 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 226–239, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-91-0. doi: 10.4230/LIPIcs.CONCUR.2015.226.

[11] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Efficient syntax-driven lumping of differential equations. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 93–111. Springer, 2016.

[12] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Comparing chemical reaction networks: a categorical and algorithmic perspective. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 485–494. ACM, 2016.

[13] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. ERODE: A tool for the evaluation and reduction of ordinary differential equations. In *TACAS (2)*, pages 310–328, 2017.

[14] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Syntactic markovian bisimulation for chemical reaction networks. In Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare, editors, *Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday*, pages 466–483. Springer International Publishing, Cham, 2017. ISBN 978-3-319-63121-9. doi: 10.1007/978-3-319-63121-9_23.

[15] Gourab Chatterjee, Neil Dalchau, Richard A Muscat, Andrew Phillips, and Georg Seelig. A spatially localized architecture for fast and modular DNA computing. *Nature Nanotechnology*, 12(9):920, 2017.

[16] Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. In *DNA 2012: Proceedings of The 18th International Meeting on DNA Computing and Molecular Programming*, volume 7433 of *Lecture Notes in Computer Science*, pages 25–42. Springer, 2012.

[17] Ho-Lin Chen, David Doty, and David Soloveichik. Rate-independent computation in continuous chemical reaction networks. In Moni Naor, editor, *Proceedings of the 5th conference on Innovations in Theoretical Computer Science*, pages 313–326. ACM, 2014.

[18] Sherry Xi Chen, David Yu Zhang, and Georg Seelig. Conditionally fluorescent molecular probes for detecting single base changes in double-stranded DNA. *Nature Chemistry*, 5(9):782, 2013.

[19] Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8(10):755–762, 2013.

[20] Kevin M Cherry and Lulu Qian. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, 559(7714):370, 2018.

[21] Evelyne Contejean and Hervé Devie. An efficient incremental algorithm for solving systems of linear Diophantine equations. *Information and Computation*, 113:143–172, 1994.

[22] Nadine L Dabby. *Synthetic molecular machines for active self-assembly: prototype algorithms, designs, and experimental study*. PhD thesis, California Institute of Technology, February 2013.

[23] Robert M Dirks and Niles A Pierce. Triggered amplification by hybridization chain reaction. *Proceedings of the National Academy of Sciences*, 101 (43):15275–15278, 2004.

[24] Robert M Dirks, Justin S Bois, Joseph M Schaeffer, Erik Winfree, and Niles A Pierce. Thermodynamic analysis of interacting nucleic acid strands. *SIAM review*, 49(1):65–88, 2007.

[25] Alexander Dobrinevski and Erwin Frey. Extinction in neutrally stable stochastic Lotka-Volterra models. *Physical Review E*, 85:051903, 2012.

[26] David Doty and Monir Hajiaghayi. Leaderless deterministic chemical reaction networks. In David Soloveichik and Bernard Yurke, editors, *DNA 2013: Proceedings of The 19th International Meeting on DNA Computing and Molecular Programming*, volume 8141 of *Lecture Notes in Computer Science*, pages 46–60. Springer International Publishing, 2013. doi: 10.1007/978-3-319-01928-4_4. URL `http://dx.doi.org/10.1007/978-3-319-01928-4_4`.

[27] Shawn M Douglas, Ido Bachelet, and George M Church. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335 (6070):831–834, 2012.

[28] Abeer Eshra, Shalin Shah, Tianqi Song, and John Reif. Renewable DNA hairpin-based logic circuits. *IEEE Transactions on Nanotechnology*, 18: 252–259, 2019.

[29] Constantine G Evans and Erik Winfree. DNA sticky end design and assignment for robust algorithmic self-assembly. In David Soloveichik and Bernard Yurke, editors, *DNA Computing and Molecular Programming*, volume 8141 of Lecture Notes in Computer Science, pages 61–75. Springer, 2013.

[30] Constantine Glen Evans. *Crystals that count! Physical principles and experimental investigations of DNA tile self-assembly*. PhD thesis, California Institute of Technology, 2014.

[31] Jean-Claude Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13:219–236, 1990.

[32] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

[33] Steven Gay, Sylvain Soliman, and François Fages. A graphical method for reducing and relating models in systems biology. *Bioinformatics*, 26: i575, 2010. doi: 10.1093/bioinformatics/btq388.

[34] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977. doi: 10.1021/j100540a008.

[35] Benjamin Groves, Yuan-Jyue Chen, Chiara Zurla, Sergii Pochekailov, Jonathan L Kirschman, Philip J Santangelo, and Georg Seelig. Computing in mammalian cells with nucleic acid strand exchange. *Nature Nanotechnology*, 11(3):287, 2016.

[36] Casey Grun, Karthik Sarma, Brian Wolfe, Seung Woo Shin, and Erik Winfree. A domain-level DNA strand displacement reaction enumerator allowing arbitrary non-pseudoknotted secondary structures. *CoRR*, 2015. URL `http://arxiv.org/abs/1505.03738`.

[37] Gabor T. Herman. Strong computability and variants of the uniform halting problem. *Mathematical Logic Quarterly*, 17(1):115–131, 1971. ISSN 1521-3870. doi: 10.1002/malq.19710170117.

[38] Allen Hjelmfelt, Edward D Weinberger, and John Ross. Chemical implementation of neural networks and Turing machines. *Proceedings of the National Academy of Sciences*, 88(24):10983–10987, 1991.

[39] Petr Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.

[40] Robert F Johnson, Qing Dong, and Erik Winfree. Verifying chemical reaction network implementations: A bisimulation approach. *Theoretical Computer Science*, 2018. doi: 10.1016/j.tcs.2018.01.002.

[41] Neil D Jones, Lawrence H Landweber, and Y Edmund Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

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

[43] Dexter Kozen. *Automata and computability.* Springer, 1997.

[44] Michael Lachmann and Guy Sella. The computationally complete ant colony: Global coordination in a system with no hierarchy. In Federico Morán, Alvaro Moreno, Juan Julián Merelo, and Pablo Chacón, editors,

Advances in Artificial Life: Third European Conference on Artificial Life Granada, Spain, June 4–6, 1995 Proceedings, pages 784–800. Springer, 1995. doi: 10.1007/3-540-59496-5_343.

[45] Matthew R. Lakin, Simon Youssef, Filippo Polo, Stephen Emmott, and Andrew Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011. doi: 10.1093/bioinformatics/btr543. URL `http://bioinformatics.oxfordjournals.org/content/27/22/3211.abstract`.

[46] Matthew R Lakin, Darko Stefanovic, and Andrew Phillips. Modular verification of chemical reaction network encodings via serializability analysis. *Theoretical Computer Science*, 632:21–42, 2016.

[47] Jérôme Leroux. Vector addition systems reachability problem (a simpler solution). In Andrei Voronkov, editor, *The Alan Turing Centenary Conference*, volume 10 of *EPiC Series*, pages 214–228, Manchester, United Kingdom, June 2012. Andrei Voronkov. URL `https://hal.archives-ouvertes.fr/hal-00674970`.

[48] Shen Lin and Tibor Rado. Computer studies of Turing machine problems. *J. ACM*, 12(2):196–212, April 1965. ISSN 0004-5411. doi: 10.1145/321264.321270.

[49] Richard Lipton. The reachability problem requires exponential space. *Research Report 63, Department of Computer Science, Yale University, New Haven, Connecticut*, pages 1–15, 1976.

[50] Marcelo O Magnasco. Chemical kinetics is Turing universal. *Physical Review Letters*, 78(6):1190–1193, 1997.

[51] Albert R Meyer and Larry J Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Switching and Automata Theory, 1972., IEEE Conference Record of 13th Annual Symposium on*, pages 125–129. IEEE, 1972.

[52] Robin Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[53] Rasmus L Petersen, Matthew R Lakin, and Andrew Phillips. A strand graph semantics for DNA-based computation. *Theoretical Computer Science*, 632:43–73, 2016.

[54] Loic Pottier. Minimal solutions of linear Diophantine systems: bounds and algorithms. In *International Conference on Rewriting Techniques and Applications*, pages 162–173. Springer, 1991.

[55] Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[56] Lulu Qian and Erik Winfree. Parallel and scalable computation and spatial dynamics with DNA-based chemical reaction networks on a surface. In Satoshi Murata and Satoshi Kobayashi, editors, *DNA Computing and Molecular Programming*, volume 8727 of Lecture Notes in Computer Science, pages 114–131. Springer, 2014.

[57] Lulu Qian, David Soloveichik, and Erik Winfree. Efficient Turing-universal computation with DNA polymers. In Yasubumi Sakakibara and Yongli Mi, editors, *DNA Computing and Molecular Programming*, volume 6518 of Lecture Notes in Computer Science, pages 123–140. Springer, 2011.

[58] Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978.

[59] Paul WK Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297, 2006.

[60] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177 – 192, 1970.

[61] Joseph Malcolm Schaeffer, Chris Thachuk, and Erik Winfree. Stochastic simulation of the kinetics of multiple interacting nucleic acid strands. In Andrew Phillips and Peng Yin, editors, *DNA Computing and Molecular Programming*, volume 9211 of Lecture Notes in Computer Science, pages 194–211. Springer, 2015.

[62] Jong-Shik Shin and Niles A Pierce. A synthetic DNA walker for molecular transport. *Journal of the American Chemical Society*, 126(35):10834–10835, 2004.

[63] Seung Woo Shin. *Compiling and verifying DNA-based chemical reaction network implementations*. PhD thesis, California Institute of Technology, 2011.

[64] Seung Woo Shin, Chris Thachuk, and Erik Winfree. Verifying chemical reaction network implementations: A pathway decomposition approach. *Theoretical Computer Science*, page doi:10.1016/j.tcs.2017.10.011, 2017.

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

[66] 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.

[67] Niranjan Srinivas, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik. Enzyme-free nucleic acid dynamical systems. *Science*, 358: doi:10.1126/science.aal2052, 2017. ISSN 0036-8075.

[68] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (Preliminary Report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, STOC '73, pages 1–9, New York, NY, USA, 1973. ACM. doi: 10.1145/800125.804029.

[69] Chris Thachuk and Anne Condon. Space and energy efficient computation with DNA strand displacement systems. In Darko Stefanovic and Andrew Turberfield, editors, *DNA Computing and Molecular Programming*, volume 7433 of Lecture Notes in Computer Science, pages 135–149. Springer, 2012.

[70] Chris Thachuk, Erik Winfree, and David Soloveichik. Leakless DNA strand displacement systems. In Andrew Phillips and Peng Yin, editors, *DNA Computing and Molecular Programming*, volume 9211 of Lecture Notes in Computer Science, pages 133–153. Springer, 2015.

[71] Anupama J. Thubagere, Wei Li, Robert F. Johnson, Zibo Chen, Shayan Doroudi, Yae Lim Lee, Gregory Izatt, Sarah Wittman, Niranjan Srinivas, Damien Woods, Erik Winfree, and Lulu Qian. A cargo-sorting DNA robot. *Science*, 357(6356), 2017. ISSN 0036-8075. doi: 10.1126/science.aan6558. URL `https://science.sciencemag.org/content/357/6356/eaan6558`.

[72] Anupama J Thubagere, Chris Thachuk, Joseph Berleant, Robert F Johnson, Diana A Ardelean, Kevin M Cherry, and Lulu Qian. Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. *Nature Communications*, 8:14373, 2017.

[73] Grigory Tikhomirov, Philip Petersen, and Lulu Qian. Fractal assembly of micrometre-scale DNA origami arrays with arbitrary patterns. *Nature*, 552(7683):67, 2017.

[74] Stefano Tognazzi, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. EGAC: A genetic algorithm to compare chemical reaction networks. In *Proceedings of 1st Genetic and Evolutionary Computation Conference-17 (GECCO'17)*, pages 833–840, 2017.

[75] Toma E Tomov, Roman Tsukanov, Yair Glick, Yaron Berger, Miran Liber, Dorit Avrahami, Doron Gerber, and Eyal Nir. DNA bipedal motor achieves a large number of steps due to operation using microfluidics-based interface. *Acs Nano*, 11(4):4002–4008, 2017.

[76] Marko Vasić, David Soloveichik, and Sarfraz Khurshid. CRN++: Molecular programming language. *Natural Computing*, pages 1–17, 2020.

[77] Suvir Venkataraman, Robert M Dirks, Paul WK Rothemund, Erik Winfree, and Niles A Pierce. An autonomous polymerization motor powered by DNA hybridization. *Nature Nanotechnology*, 2(8):490, 2007.

[78] Boya Wang, Chris Thachuk, Andrew D Ellington, Erik Winfree, and David Soloveichik. Effective design principles for leakless strand displacement systems. *Proceedings of the National Academy of Sciences*, 115(52): E12182–E12191, 2018.

[79] Bryan Wei, Mingjie Dai, and Peng Yin. Complex shapes self-assembled from single-stranded DNA tiles. *Nature*, 485(7400):623, 2012.

[80] Erik Winfree. On the computational power of DNA annealing and ligation. In Richard J. Lipton, editor, *DNA Based Computers*, volume 27 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 199–221. American Mathematical Society, 1996.

[81] Brian R Wolfe, Nicholas J Porubsky, Joseph N Zadeh, Robert M Dirks, and Niles A Pierce. Constrained multistate sequence design for nucleic acid reaction pathway engineering. *Journal of the American Chemical Society*, 139(8):3134–3144, 2017.

[82] Damien Woods, David Doty, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, and Erik Winfree. Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. *Nature*, 567(7748):366, 2019.

[83] Peng Yin, Harry MT Choi, Colby R Calvert, and Niles A Pierce. Programming biomolecular self-assembly pathways. *Nature*, 451(7176):318–322, 2008.

[84] Joseph N Zadeh, Conrad D Steenberg, Justin S Bois, Brian R Wolfe, Marshall B Pierce, Asif R Khan, Robert M Dirks, and Niles A Pierce. Nupack: analysis and design of nucleic acid systems. *Journal of computational chemistry*, 32(1):170–173, 2011.

[85] Yechezkel Zalcstein. Locally testable languages. *Journal of Computer and System Sciences*, 6(2):151–167, 1972.

[86] David Yu Zhang. Cooperative hybridization of oligonucleotides. *Journal of the American Chemical Society*, 133(4):1077–1086, 2010.

[87] David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry*, 3(2):103–113, 2011.