Verifying polymer reaction networks using bisimulation

Robert Johnson[1] and Erik Winfree[2]

[1]*Biology, California Institute of Technology*
[1]*Computer Science & Bioengineering, California Institute of Technology*

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 no 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^0_2-complete, and thus undecidable in the general case. We hope that the ability to model and verify implementations of polymer reaction networks will aid in the rational design of molecular systems.

1 Introduction

A Chemical Reaction Network (CRN) models a finite set of chemical species with a finite number of specified reactions. CRNs have been shown to simulate digital circuits [14] and neural networks [10], to be able to compute semilinear functions deterministically [1], and to simulate register machines with arbitrarily small probability of error [20]. Since Soloveichik et. al. showed that any CRN can be approximated by a DNA strand displacement circuit [21], CRNs have become a candidate programming language for rational design of molecular devices. Recently, Chen et. al. [6] have successfully implemented a CRN-to-DNA translation scheme proposed by Cardelli [3] *in vitro*, and work is ongoing to implement such schemes *in vivo*.

The process of designing DNA implementations of CRNs has been somewhat ad-hoc and has not always produced the correct behavior. For example, in some irreversible reactions, when implemented by Cardelli's two-domain strand displacement system, the irreversible steps do not happen until after some outputs have been released. As Cardelli notes, this has a small probability of causing behavior that cannot happen in the CRN it attempts to implement. Cardelli also gives examples of rejected designs for gates which have

unintended behavior not obvious at first glance [3]. For this reason, we would like to have a formal definition of what it means for an implementation to be correct, and if possible a way to check that correctness algorithmically.

As a first step, reaction enumerators such as Visual DSD [13] [12] take a domain-level description of a set of DNA strands and complexes and produce a CRN that describes the strand displacement system. This reduces the problem to what it means for two CRNs to be equivalent, or for one CRN to be an implementation of the other. Shin [19] and Dong [7] proposed methods of verifying that a given CRN implementation of a formal CRN is a correct implementation. Shin's method, based on pathway decomposition, and Dong's method, based on bisimulation, both create a set of conditions under which we can say that the formal CRN is simulated, reaction by reaction, by the implementation CRN. In particular, both of them imply that the set of reachable formal states and the set of reachable implementation states are identical, after applying some idea of correspondence between implementation states and formal states. While both methods assert the correctness of the Soloveichik et. al. scheme, it is easy to find examples of implementation-formal CRN pairs which are correct according to pathway decomposition but not according to bisimulation, and vice versa.

Despite their potential use, computation with CRNs is limited in one important sense. An obvious way to do computation with CRNs is to construct a CRN that takes its input in the form of the counts of one or more of its species, and eventually produces its output in the form of counts of other species. For example, a CRN might take a number as input as the count of some molecule X, and simulate some Turing machine's computation on the binary representation of that number, producing one copy of Y if the Turing machine accepts and one copy of Z if it rejects (or neither if it does not halt). The problem of whether a given molecule can be produced from a given starting state in a CRN is decidable in exponential space [20] [18], meaning that if CRNs could simulate any Turing machine according to this concept of computation, the Halting Problem would be decidable. Thus, this concept of computation must be weaker than Turing machines. This result has been refined by Angluin et. al. [1], Chen et. al. [5], and Doty and Hajiaghayi [8], who show that CRNs which compute the correct answer on every path can compute exactly semilinear functions. On the positive side, Soloveichik et. al. have shown that CRNs can simulate arbitrary Turing machines with probability arbitrarily close to 1, by decreasing the probability of error at each step so that the limit is finite, which can be made arbitrarily small [20]. A DNA stack machine has been proposed by Qian et. al. that can simulate arbitrary Turing machines with no slowdown, at the cost of using unbounded polymers making it unable to be modeled as a CRN [17]. Both of those constructions require exactly one copy of the important molecules to be present, presenting a problem for practical implementation. Cardelli and Zavattaro proposed a model of reaction networks with association and dissociation reactions that is also Turing-universal, as it is able to simulate register machines as well as model the above stack machine [4]. As Cardelli's model is able to handle branching polymers, it is more capable of modelling biochemical systems than the model we will propose, however, it seems to have no way to model context-sensitive reactions. Cardelli's register machine, like Qian et. al.'s DNA stack machine, relies on the presence of exactly one copy of key molecules, while we will show that having context-sensitive reactions allows the simulation of Turing machines which do not have that restriction. Furthermore, the use of branching polymers makes it more difficult to verify the equivalence of two networks.

We propose a model of linear polymer reaction networks, which can model Qian et. al.'s DNA stack machine as well as copy-tolerant Turing machines. We extend Dong et. al.'s definition of bisimulation to this model, and use it to prove the correctness of the DNA stack machine. We then ask whether it is possible to algorithmically determine whether an implementation is a bisimulation. Unlike in the finite case, where bisimulation is decidable [7], we show that deciding bisimulation for polymer networks is undecidable, in fact Π^0_2-complete. That is, although bisimulation for Polymer Reaction Networks is undecidable, we might think that it would be possible to provide a proof that an implementation is or is not a bisimulation, such as a formula for producing the simulation of any formal reaction or an example of a state from which a

given reaction cannot be simulated, which can then be algorithmically verified. This result, plus properties of the arithmetic hierarchy [11], means that neither of these are possible. Finally, we define a subclass of polymer reaction networks called single-locus networks, which includes the DNA stack machine and our copy-tolerant Turing machines, and show that any member of this subclass can be bisimulated by a network using only reactions among monomers and four polymer primitives. Since bisimulation is transitive, this means that a DNA implementation of those four primitives that is proven to be a bisimulation will be able to simulate any single-locus polymer reaction network.

2 Definitions

2.1 Transition Systems and Weak Bisimulation

The theory of transition systems and bisimulation is described by Milner [16], and CRN and PRN bisimulation are both special cases of the general theory. We recall Milner's definitions, with some changes in notation.

Definition 2.1. A *Labelled Transition System* is a tuple (S, L, T), where S is a set of states, L a set of labels or actions, and $T \subset S \times S \times L$ is a set of transitions.

If $(s, t, \alpha) \in T$ is a transition, we say that state $s \in S$ can transition to state $t \in S$ by action $\alpha \in L$, and write $s \xrightarrow{\alpha} t$. Note that Milner does not require the system to be finite.

Definition 2.2. Given a labelled transition system (S, L, T), a relation $\leftrightarrow \subset S \times S$ *is a strong bisimulation* if

$$s_1 \leftrightarrow s_2 \text{ and } s_1 \xrightarrow{\alpha} t_1 \Rightarrow s_2 \xrightarrow{\alpha} t_2 \text{ for some } t_2 \in S \text{ such that } t_1 \leftrightarrow t_2$$
$$s_1 \leftrightarrow s_2 \text{ and } s_2 \xrightarrow{\alpha} t_2 \Rightarrow s_1 \xrightarrow{\alpha} t_1 \text{ for some } t_1 \in S \text{ such that } t_1 \leftrightarrow t_2$$

for all $\alpha \in L$. That is, a relation is a strong bisimulation if for any pair of related states, any action that can be taken by one can be taken by the other, and the resulting states are also related.

Especially when discussing DNA implementations of CRNs, we do not necessarily demand a one-to-one correspondence of actions. Milner's definition of weak bisimulation, which allows the system to contain a "silent" action τ which is ignored when comparing sequences of actions for bisimulation, is thus more appropriate.

Where $\tau \in L$ is a special symbol denoting a silent action, the relation $\xrightarrow{\tau}$ is as defined above, and $\xrightarrow{\tau}^*$ is the reflexive and transitive closure of $\xrightarrow{\tau}$. We then say $s \overset{\alpha}{\Rightarrow} t$ if (1) $s \xrightarrow{\tau}^* s' \xrightarrow{\alpha} t' \xrightarrow{\tau}^* t$ for some s', t', when $\alpha \neq \tau$, or (2) $s \xrightarrow{\tau}^* t$, when $\alpha = \tau$.

Definition 2.3. Given a labelled transition system (S, L, T) where $\tau \in L$ is the silent action, a relation $\leftrightarrow \subset S \times S$ *is a weak bisimulation* if

$$s_1 \leftrightarrow s_2 \text{ and } s_1 \xrightarrow{\alpha} t_1 \Rightarrow s_2 \overset{\alpha}{\Rightarrow} t_2 \text{ for some } t_2 \in S \text{ such that } t_1 \leftrightarrow t_2$$
$$s_1 \leftrightarrow s_2 \text{ and } s_2 \xrightarrow{\alpha} t_2 \Rightarrow s_1 \overset{\alpha}{\Rightarrow} t_1 \text{ for some } t_1 \in S \text{ such that } t_1 \leftrightarrow t_2$$

That is, for any pair of related states, any action that can be done in one state can be implemented from the other by a sequence of zero or more silent actions, followed by one corresponding action, followed by zero or more silent actions, with the final states being related.

Because weak bisimulation is more relevant to DNA implementations of CRNs, for the remainder of this paper we use "bisimulation" to refer specifically to weak bisimulation.

2.2 Chemical Reaction Networks

Definition 2.4. A *Chemical Reaction Network* (CRN) is a tuple (Sp, Rx), where Sp is a finite set of species with $k = |Sp|$ and $Rx \subset \mathbb{N}^k \times \mathbb{N}^k$ is a finite 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 $R \rightleftharpoons P$.

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. We can model this as a labelled transition system where $S = \mathbb{N}^k$, $L = Rx$, and $(s, t, R \to P) \in T$ if $R \leq s$ and $t = s - R + P$. In general this transition system will have infinitely many states and transitions. Thus, while we can extend the definition of bisimulation directly to CRNs, it would be difficult to use for practical purposes.

Dong [7] solves this problem by considering relations between two CRNs, one of which is designated the *formal* CRN (Sp, Rx) and the other the *implementation* CRN (Si, Ri), which are induced by an interpretation $m_1 : Si \to \mathbb{N}^{Sp}$ (here $\mathbb{N}^{Sp}$ refers to the set of functions from Sp to the natural numbers $\mathbb{N}$, or equivalently a count of each formal species). That is, each implementation species is interpreted as a set of formal species; for example, a signal strand may be interpreted as X, while a gate with three specific signal strands bound may be interpreted as $2X + Y$. This interpretation induces a function $m : \mathbb{N}^{Si} \to \mathbb{N}^{Sp}$, which is linear and agrees with m_1 on single-element sets, interpreting an implementation state as the formal state obtained by summing the interpretations of the species present. Implementation reactions can also be interpreted as formal reactions, where $m(R' \to P') = m(R') \to m(P')$. Some implementation reactions will have $m(R') = m(P')$, i.e. the reaction does not correspond to a change in the formal state but only a change in the representation of that state. We call these reactions "trivial" or "silent", corresponding to the silent action τ. In the language of Milner's definitions, we consider a transition system $S = \mathbb{N}^{Sp} \cup \mathbb{N}^{Si}$, $L = Rx \cup \{\tau\}$, and $(s, t, R \to P) \in T$ if s and t are formal states and the reaction $R \to P$ can occur in s and takes s to t, or s and t are implementation states and some reaction $R' \to P'$ can occur in s and takes s to t where $m(R' \to P') = R \to P$, while $(s, t, \tau) \in T$ if s and t are implementation states and some trivial reaction can occur in s and takes s to t. As a function, m is also a relation $m \subset \mathbb{N}^{Si} \times \mathbb{N}^{Sp} \subset S \times S$. We can then ask whether this relation is a bisimulation. Since we are discussing implementations of abstract CRNs, we would additionally like any formal state to have some implementation state that implements (is interpreted as) it, i.e. m is surjective.

Lemma 2.1. *If for some CRNs (Sp, Rx) and (Si, Ri) a relation $\leftrightarrow \subset \mathbb{N}^{Si} \times \mathbb{N}^{Sp}$ is a linear function from $\mathbb{N}^{Si} \to \mathbb{N}^{Sp}$ then $\leftrightarrow$ is the relation induced by some interpretation $m_1 : Si \to \mathbb{N}^{Sp}$.*

Proof. For each $x \in Si$, since $\leftrightarrow$ is a function there exists a unique state S_x such that $\{x\} \leftrightarrow S_x$; let $m(x) = S_x$. Since $\leftrightarrow$ and the induced m are both linear, this is sufficient to determine m, and since $m = \leftrightarrow$ on a basis, they are equal. □

For this reason we refer to this type of bisimulation as an *interpretation-based* or *surjective linear* bisimulation. An example interpretation is shown in Figure 1.

Dong [7] defines three conditions which are useful for examining the complexity of bisimulation and which, together, are equivalent to the fact that m is a surjective linear bisimulation.

Definition 2.5. Atomic condition: for every formal species X there is an implementation species x such that $m_1(x) = X$. Since m is linear, the atomic condition holds if and only if m is surjective [7].

Delimiting condition: for every implementation reaction $R' \to P'$, either $m(R') = m(P')$ in which case we say the interpretation is a trivial reaction, or $m(R') \to m(P')$ is a formal reaction.

Permissive condition: for every implementation state S' and formal reaction $R \to P$ that can occur in $m(S')$, there exists a sequence of implementation reactions $r_1, \ldots, r_n$ that can occur starting from S' such

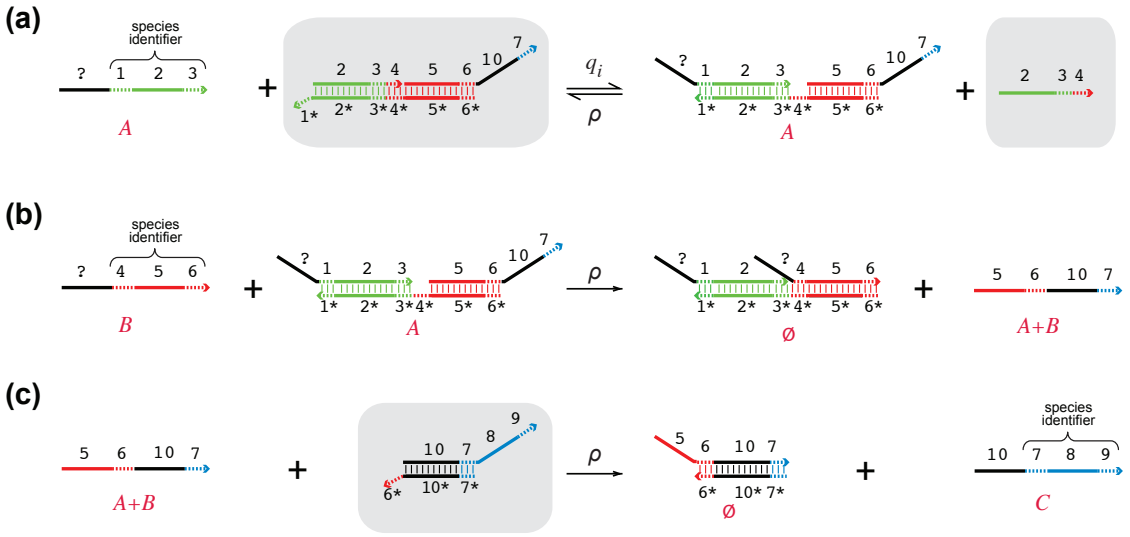

Figure 1: An interpretation (red labels) of the Soloveichik et. al. implementation scheme for the reaction $A + B \to C$. Grey boxes indicate *fuel species* whose concentrations are assumed to be held constant, and thus are eliminated from the implementation CRN, as described in [19]. Figure modified from [21].

that $m(r_i)$ is trivial for $1 \leq i \leq n-1$, $m(r_n) = (R \to P)$. Given the atomic condition, the delimiting and permissive conditions are equivalent to the two statements in Definition 2.3.

For computational complexity purposes, we assume CRNs are encoded as a string of n 1's, where n is the number of species, followed by a list of reactions. Each reaction is encoded as a pair of vectors in $\mathbb{N}^n$, the first being the reactants and the second the products, i.e. two lists of n numbers each, which are the counts of each species in order, with counts given in binary. In other words, a time or space complexity is polynomial in "the size of the CRN" if it is polynomial in (a) the number of species, (b) the number of reactions, and (c) the binary representation (base-2 logarithm) of the highest count of any species in the reactants or products of any reaction. The use of n 1's to encode the number of species allows us to ignore the case of a CRN with 0 reactions.

2.3 Polymer Reaction Networks

Like a Chemical Reaction Network is a finite specification of an infinite transition system, a Polymer Reaction Network can be thought of as a finite specification of an infinite Chemical Reaction Network. Unlike transition systems, CRNs are defined to be finite, which leads to results such as the decidability of the reachability problem and thus the inability to deterministically simulate arbitrary Turing machines. However, the semantics of CRNs are not significantly affected by removing the requirement that they be finite, and doing so allows Turing-universal computation as we will soon see. Again, the problem is how to specify such an infinite CRN with a finite description. We consider linear Polymer Reaction Networks whose species are the set of all linear, unbranching polymers made up of some finite set of monomers Σ such that any two adjacent monomers are compatible as specified by a "compatibility relation" ρ, and whose reactions are generated a set Rs of reaction schemes by replacing wildcards with arbitrary strings.

Definition 2.6. The species of a Polymer Reaction Network are specified by a *monomer set* or *alphabet* Σ, a finite set, and *compatibility relation* $\rho \subset (\Sigma \cup \{\vdash\}) \times (\Sigma \cup \{\dashv\})$. The set of species $Sp(\Sigma', \rho')$ is the set of all polymers $w_1 \ldots w_n$ with monomers $w_i \in \Sigma$ such that $(\vdash, w_1)$, $(w_n, \dashv)$, and (w_i, w_{i+1}) for $1 \leq i \leq n-1$ are all elements of ρ; in other words, all linear polymers where consecutive monomers are compatible, which includes the initial and final monomers being "compatible" with being on the end of a

polymer. In particular, individual monomers are a special case of this as strings of length 1, but not every monomer is capable of existing by itself.

In many cases, especially PRNs which are meant to model actual systems such as DNA strand displacement circuits, not every pair of monomers can bind to each other, but whether a polymer is possible depends only on whether two adjacent monomers can bind to each other in that orientation, hence the use of a compatibility relation. Since the compatibility relation is local, the set of valid species can be expressed as a regular expression over Σ. In fact, the two notions are almost equivalent.

Lemma 2.2. *For any regular language L over an alphabet Σ, there is a species specification (Σ', ρ') and interpretation $\pi : \Sigma' \to \Sigma$ such that a string $x = x_1 \dots x_n \in \Sigma^*$ is in L if and only if there exists a valid species $x' = x'_1 \dots x'_n \in Sp(\Sigma', \rho')$ such that $\pi(x'_i) = x_i$ for $1 \leq i \leq n$.*

Proof. Consider a nondeterministic finite automaton M with no ε-transitions that recognizes L. 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 $(\varepsilon, (x, q)) \in \rho'$ if and only if M can transition from its start state q_0 to q by reading x, and $((x, q), \varepsilon) \in \rho'$ if and only if q is an accept state of M. Valid polymers correspond exactly to accepting computation paths of M on their interpretations. □

Because compatibility relations and regular expressions are equivalent up to interpretation, we will sometimes specify ρ implicitly by giving a regular expression which the set of valid species matches. For example, polymers made of monomers $\Sigma = \{a, b, c\}$ where a is stable at the beginning, c at the end, a and b can bind in any order and c can follow b, can be described by $\rho = \{(\vdash, a), (a, b), (b, a), (b, b), (b, c), (c, \dashv)\}$ or by the regular expression $a(bb^*a)^*bc$.

Definition 2.7. The reactions of a Polymer Reaction Network are specified by a finite set Rs of *reaction schemes*, each of which is a pair (R, P) of multisets of strings over $\Sigma \cup \mathbb{N}$. A reaction is obtained from a reaction scheme by, for each number (which may be written $*_1, *_2, \dots$ and is used as a wildcard) that appears in the reaction scheme, choosing a string over Σ and substituting that string for all instances of that number in R and P, provided that the resulting strings are valid species. The set of reactions Rx is the set of all such substitutions into a reaction scheme.

Definition 2.8. A *Polymer Reaction Network* is a tuple (Σ, ρ, Rs) which are a monomer set, compatibility relation, and set of reaction schemes, respectively.

Example 2.1. Microtubule Dynamics

A simplified model of microtubule dynamics in biochemistry is: each tubulin monomer can be bound to either GTP or GDP; a GTP-bound monomer can add on the "right" end of a microtubule; the "leftmost" GTP-bound monomer can hydrolyze its GTP; a GDP-bound monomer can fall off either end. This is well-suited to the PRN model. In this model, $\Sigma = \{T, D\}$ (GTP and GDP, respectively); $\rho = \{(\vdash, T), (\vdash, D), (D, D), (D, T), (T, T), (D, \dashv), (T, \dashv)\}$; and reaction schemes are $*_1 \to *_1T$, $*_1DT*_2 \to *_1DD*_2$, $D*_1 \to *_1$, and $*_1D \to *_1$.

Example 2.2. Copy-tolerant Turing Machine

Molecular Turing machines that encode head state and tape in a single polymer, with hypothetical enzymes that perform the relevant modifications such that any number of Turing machines can be run in a single well-mixed reaction vessel, have been considered previously [2]. Here we illustrate how such systems can be described using our formalism. Given a Turing machine M over alphabet $\Gamma = \{0, 1, b\}$, where b is the blank symbol, we can construct a PRN that simulates it. Let $\Sigma = Q \cup \Gamma^l \cup \Gamma^r$, where Q is the set of states of M, and $\Gamma^i = \{0^i, 1^i, b^i\}$ for $i \in \{l, r\}$, i.e. there are separate monomers for each symbol on the "left" and on the "right". Let ρ be given in regular expression notation $(0^l|1^l|b^l)^*Q(0^r|1^r|b^r)^*$

where $Q = q_0|q_1|\ldots|q_H$. Where transitions of M are given as 5-tuples (q_i, x, q_j, y, D) meaning in state q_i reading x, move to state q_j, write y, and move in direction D on the tape, we have reaction schemes $*_1 q_i x^r *_2 \to *_1 y^l q_j *_2$ if $D = R$ and $*_1 c^l q_i x^r *_2 \to *_1 q_j c^r y^r *_2$ if $D = L$. Finally, we have reaction schemes $*_1 \rightleftharpoons *_1 b^r$ and $*_1 \rightleftharpoons b^l *_1$ allowing the tape to extend arbitrarily, simulating the ideal Turing machine tape which is infinite in both directions.

Since every reaction scheme is either playing with the length of the tape or a transition of M, this PRN simulates M "deterministically" in the following sense: for any input string x, from initial state $q_0 x^r$ (where x is a string, we use x^r to refer to the sequence of PRN monomers corresponding to characters of x), if M halts on x, after any (finite) sequence of reaction in this PRN there exists another finite sequence of reactions after which the state is $u^l q_H v^r$, where on input x, M halts with uv written on the tape while reading the first character of v, and if M does not halt on x, then any state with any polymer that includes a q_H monomer is unreachable from $q_0 x^r$. An even stronger statement of simulation is that any sequence of reactions from $q_0 x^r$ will pass through states including the polymer $u^l q_i v^r$ corresponding to the computation of M on input x. The stack machine in [17] and the register machine in [4] both have the same property. However, unlike the stack machine and the register machine, this PRN is *copy-tolerant*, that is, multiple polymers will simulate their own computations without interfering with each other. Formally, from an initial state $q_0 x^r + S$, where S is any set of polymers, after any initial sequence of reactions there exists another sequence of reactions after which the state is $u^l q_H v^r + T$, where M on input x halts with uv on the tape reading the first character of v and T is reachable from S. In particular, arbitrarily many different "tapes", whether on the same or different input, do not interfere with each others' computations. The previously existing stack machine and register machine models rely on having exactly one copy of the computation in the solution, posing an additional challenge to *in vitro* implementation.

In the same way as extending the definition of bisimulation from transition systems to CRNs, we can extend the definition of interpretation-based bisimulation from CRNs to PRNs, but doing so would require an infinite interpretation. In the same way as bisimulation for finite CRNs built up a relation between states from an interpretation of species, we solve this by building up an interpretation of species from an interpretation of monomers. The most obvious thing to do is to have the interpretation of a polymer be the concatenation of interpretations of its monomers, however, we also want a given implementation species to represent multiple, seperate formal polymers as is possible in the finite case. We therefore require that our interpretation be induced by two finite functions, μ and π, defined on the implementation monomers, where $\pi(x)$ is the contribution of the monomer x to the polymer it is contained in and $\mu(x)$ is any other, free-floating species represented by x. We sometimes say that x *polymerizes as* $\pi(x)$ and *carries* $\mu(x)$.

Definition 2.9. Given a formal PRN (Σ, ρ, Rs) and implementation PRN (Σ', ρ', Rs'), a *polymer interpretation* is a pair (μ, π) of functions $\mu : \Sigma' \to \mathbb{N}^{Sf}$ and $\pi : \Sigma' \to (\Sigma \cup \{|\})^*$. These functions induce an interpretation $m_1 : Si \to \mathbb{N}^{Sf}$ defined by

$$m_1(x_1 \ldots x_n) = \pi(x_1) \ldots \pi(x_n) + \sum_{i=1}^{n} \mu(x_i).$$

The symbol | is interpreted as breaking a polymer—that is, $AB|CD = AB + CD$, and if $\pi(x) = AB|CD$ and $\mu(x) = EF + 2GH$ then $m(xx) = AB + CDAB + CD + 2EF + 4GH$.

Provided that this interpretation preserves validity of species, adapting the definition of bisimulation is almost straightforward. Under the literal interpretation of CRN bisimulation, if $\pi(x) = X$ and $\mu(x) = Y$ while $\pi(z) = Z$ and $\mu(z) = \emptyset$ then the reaction scheme $*_1 x *_2 \to *_1 z *_2$ logically should be interpreted as $*_1 X *_2 + Y \to *_1 Z *_2$. 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. To avoid this, we label the transitions with formal reaction schemes, and

define the interpretation of a wildcard in a reaction scheme to be a wildcard, regardless of whether it is possible for monomers in that wildcard to have nonempty μ. Then (μ, π) is a bisimulation if and only if $x_1 \dots x_n \in Si$ implies $\pi(x_1) \dots \pi(x_n)$ is a |-separated list of elements of Sp, and the relation m induced by the interpretation m_1 induced by (μ, π) is a surjective linear bisimulation.

3 Verifying the DNA Stack Machine

Previously, Qian et. al. proposed a DNA strand displacement circuit to simulate arbitrary stack machines [17]. The strands in this circuit are capable of reacting to form polymers of unbounded length, and thus the circuit 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 circuit is a correct bisimulation of an abstract stack machine. We show that the obvious interpretation on DNA stack machine, with a correction for irreversible reactions, is a bisimulation between the DNA strand displacement circuit and the set of abstract reactions discussed in the original stack machine paper. We also show that there is a bisimulation in the original sense of Milner [16] between the abstract reactions and a stack machine, though not an interpretation-induced bisimulation.

To model the DNA stack machine as a PRN we choose various DNA complexes as monomers in Σ, after which ρ is determined by whether complexes have complementary long domains and Rs is determined by the set of strand displacement reactions. We use the reaction enumeration semantics described by Wolfe and Shin to enumerate Rs. Certain strands complexes are denoted as "fuel" or "waste" complexes and are removed as described by Shin. For a simple stack machine such as the one described in the original paper this gives

$$\begin{aligned}\Sigma' = \{\ & 0_1, 0_1^f, 1_1, 1_1^f, \lambda_1, \lambda_1^f, 0_2, 0_2^f, 1_2, 1_2^f, \lambda_2, \lambda_2^f, 0_3, 0_3^f, 1_3, 1_3^f, \lambda_3, \lambda_3^f, \\ & 0_1^+, 0_1^-, 1_1^+, 1_1^-, \lambda_1^+, \lambda_1^-, 0_2^+, 0_2^-, 1_2^+, 1_2^-, \lambda_2^+, \lambda_2^-, 0_3^+, 0_3^-, 1_3^+, 1_3^-, \lambda_3^+, \lambda_3^-, \\ & Q, Q_1, Q_2, Q_3, I_1^Q, I_2^Q, I_3^Q, S_1, S_2, S_3, S_4, S_5, S_6, \\ & I_1^{1012Q}, I_2^{1012Q}, I_3^{1012Q}, I_4^{1012Q}, I_1^{1114Q}, I_2^{1114Q}, I_3^{1114Q}, I_4^{1114Q}, \\ & I_1^{1\lambda 16}, I_2^{1\lambda 16}, I_3^{1\lambda 16}, I_4^{1\lambda 16}, \\ & I_1^{2Q302}, I_2^{2Q302}, I_3^{2Q302}, I_4^{2Q302}, I_1^{3Q103}, I_2^{3Q103}, I_3^{3Q103}, I_4^{3Q103}, \\ & I_1^{4Q512}, I_2^{4Q512}, I_3^{4Q512}, I_4^{4Q512}, I_1^{5Q113}, I_2^{5Q113}, I_3^{5Q113}, I_4^{5Q113}\}\end{aligned}$$

Using regular expression notation

$$\begin{aligned}\rho' = & \lambda_1(0_1 \mid 1_1)^*(0_1^+ \mid 0_1^- \mid 1_1^+ \mid 1_1^- \mid \varepsilon) \mid \lambda_1^+ \mid \lambda_1^- \\ & \mid \lambda_2(0_2 \mid 1_2)^*(0_2^+ \mid 0_2^- \mid 1_2^+ \mid 1_2^- \mid \varepsilon) \mid \lambda_2^+ \mid \lambda_2^- \\ & \mid \lambda_3(0_3 \mid 1_3)^*(0_3^+ \mid 0_3^- \mid 1_3^+ \mid 1_3^- \mid \varepsilon) \mid \lambda_3^+ \mid \lambda_3^- \\ & \mid Q \mid Q_1 \mid Q_2 \mid Q_3 \mid I_1^Q \mid I_2^Q \mid I_3^Q \mid S_1 \mid S_2 \mid S_3 \mid S_4 \mid S_5 \mid S_6 \\ & \mid I_1^{1012Q} \mid I_2^{1012Q} \mid I_3^{1012Q} \mid I_4^{1012Q} \mid I_1^{1114Q} \mid I_2^{1114Q} \mid I_3^{1114Q} \mid I_4^{1114Q} \\ & \mid I_1^{1\lambda 16} \mid I_2^{1\lambda 16} \mid I_3^{1\lambda 16} \mid I_4^{1\lambda 16} \\ & \mid I_1^{2Q302} \mid I_2^{2Q302} \mid I_3^{2Q302} \mid I_4^{2Q302} \mid I_1^{3Q103} \mid I_2^{3Q103} \mid I_3^{3Q103} \mid I_4^{3Q103} \\ & \mid I_1^{4Q512} \mid I_2^{4Q512} \mid I_3^{4Q512} \mid I_4^{4Q512} \mid I_1^{5Q113} \mid I_2^{5Q113} \mid I_3^{5Q113} \mid I_4^{5Q113}\end{aligned}$$

The reaction schemes which implement pushing and popping onto the stack are, for each stack $i \in \{1, 2, 3\}$, symbol $x \in \{0, 1\}$, and previous symbol $a \in \{0, 1, \lambda\}$

$$*_1 a_i \rightleftharpoons *_1 a_i x_i^- \tag{1}$$
$$*_1 x_i^- + x_i^f \rightleftharpoons *_1 x_i^+ \tag{2}$$
$$*_1 x_i^+ \rightleftharpoons *_1 x_i + Q_i \tag{3}$$

For each stack i, interchangeability of Q is implemented by

$$Q_i \rightleftharpoons I_i^Q \tag{4}$$
$$I_i^Q \rightleftharpoons Q \tag{5}$$

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

$$S_i \rightleftharpoons I_1^{iAjB} \tag{6}$$
$$I_1^{iAjB} + A \rightleftharpoons I_2^{iAjB} \tag{7}$$
$$I_2^{iAjB} \rightarrow I_3^{iAjB} + S_j \tag{8}$$
$$I_3^{iAjB} \rightleftharpoons I_4^{iAjB} + B \tag{9}$$

The formal PRN that describes the stack machine, copied from Qian et. al. [17], is

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

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

With reaction schemes

$$S_1 + 0_1^f \to S_2 + Q$$
$$S_1 + 1_1^f \to S_4 + Q$$
$$S_1 + \lambda_1^f \to S_6 + \lambda_1^f$$
$$S_2 + Q \to S_3 + 0_2^f$$
$$S_3 + Q \to S_1 + 0_3^f$$
$$S_4 + Q \to S_5 + 1_2^f$$
$$S_5 + Q \to S_1 + 1_3^f$$

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

$$*_1 a_i + x_i^f \rightleftharpoons *_1 a_i x_i + Q_i \mid a \in \{0, 1, \lambda\}, x \in \{0, 1\}, i \in \{1, 2, 3\}$$
$$\lambda_i^f \rightleftharpoons \lambda_i + Q_i \mid i \in \{1, 2, 3\}$$

We interpret any implementation species with the same name as a formal species v as $\pi(v) = \emptyset$ and $\mu(v) = \emptyset$. Intermediates I_i^Q are interpreted as $\pi(I_i^Q) = Q_i$ and $\mu(I_i^Q) = \emptyset$, while I^{iAjB} intermediates are interpreted as $\pi(I_l^{iAjB}) = \varepsilon$, $\mu(I_1^{iAjB}) = S_i$, $\mu(I_2^{iAjB}) = S_i + A$, $\mu(I_3^{iAjB}) = B$, and $\mu(I_4^{iAjB}) = \emptyset$. Finally, the pushing and popping intermediates are interpreted as $\pi(x_i^-) = \varepsilon$ and $\mu(x_i^-) = \emptyset$ while $\pi(x_i^+) = x_i$ and $\mu(x_i^+) = Q$.

With the close correspondence between the implementation monomers and formal monomers, it is easy to see that valid implementation polymers are interpreted as valid formal polymers and that the interpretation is surjective. Reaction schemes of types (2), (5), and (8) in the implementation correspond to the formal reaction schemes for pushing and popping, Q interconversion, and stack machine transitions, respectively, while all other implementation reaction schemes correspond to τ steps. Those same τ steps can convert any implementation state into polymers made up of monomers with the same name as formal monomers, so any step in the formal system from any implementation state is matched by that sequence of τ steps to do that conversion, followed by either reactions (1) then (2); (3) then (2); (4) then (5); just (5); or (6) through (8) if the formal step is pushing; popping; converting Q_i to Q; converting Q to Q_i; or a stack machine transition, respectively.

Interestingly, the proof that this abstract DNA stack machine simulates a stack machine in the conventional sense can also be done via bisimulation, in the sense used by Milner regarding transition systems. Here a state of the abstract DNA stack machine is related to a state of the stack machine if and only if either (a) the state of the abstract DNA stack machine has exactly one molecule of Q or some Q_i and no molecules of any x_i^f, exactly one polymer beginning with λ_i for each i, and those polymers from left to right read the same as the corresponding stacks in the stack machine from bottom to top; or (b) the state of the abstract DNA stack machine has no Q or any Q_i and exactly one molecule of some x_i^f and no others, plus exactly one polymer beginning with λ_j for each j such that the x_i^f is not λ_j^f, and those polymers read the same as the corresponding stacks with the symbol represented by the one x_i^f appended to the top of stack i.

4 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

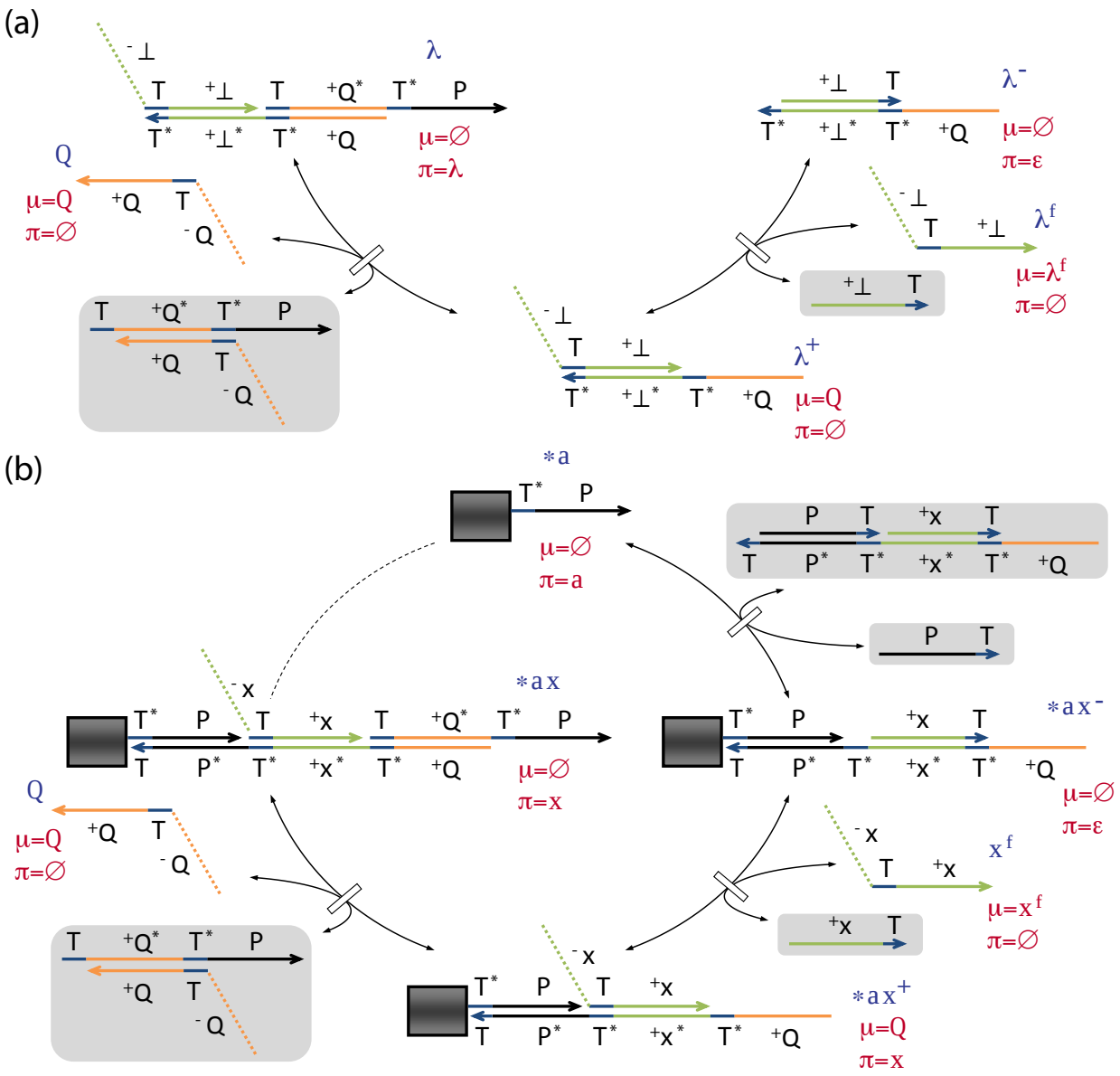

Figure 2: Interpretation of the monomers in Qian et. al.'s DNA stack machine [17]. Complexes in grey are fuels which are assumed to be always present and thus not considered according to the definition in [19]. The blue label above each species is the name of the species in the implementation PRN. The red labels below each species provide the μ and π interpretation functions for the rightmost monomer in the species. Subscripts, to indicate which stack we are considering, have been omitted for clarity. Figure modified from [17].

is a bisimulation. However, knowing that polymer networks are capable of Turing-universal computation, we might suspect that to be impossible. A next best thing would be to know that whenever an interpretation is a bisimulation, we can provide a witness to that fact which can be algorithmically verified, such as some method that gives the implementation of any given formal reaction from any given state, or a maximal number of trivial reactions necessary to implement any formal reaction from any state. Alternately, it might be that whenever an interpretation is not a bisimulation, we can provide some algorithmically verifiable witness to that fact, such as an implementation state and formal reaction which can occur in the interpretation of that state but cannot occur in the implementation network. These two cases correspond to bisimulation or non-bisimulation, respectively, being recursively enumerable. 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 [9]. The class Π_2^0, the complement of the second level of the arithmetic hierarchy, 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 [11]. Since the uniform halting problem is Π_2^0-complete [9], so is bisimulation. It is also interesting to note that Dong's atomic condition, which is trivial to check for finite CRNs, becomes PSPACE-complete for Polymer Reaction Networks, being equivalent to the problem of checking whether a regular expression describes the language of all strings, which is PSPACE-complete [22] [15].

Lemma 4.1. *The problem of, given a formal species specification (Σ, ρ), implementation species specification (Σ', ρ'), and interpretation (μ, π), deciding whether for every formal species there exists an implementation species which interprets to exactly that formal species is PSPACE-complete.*

Proof. Since we only care about implementation species that interpret to exactly one formal species, we can separately consider implementation species made up of monomers which carry $\emptyset$, and those made up of monomers which polymerize as ε and exactly one of which carries exactly one formal species, the others carrying $\emptyset$. If for some monomer $x \in \Sigma'$ such that $|\mu(x)| = 1$ and $\pi(x) = \varepsilon$ there is a valid implementation polymer containing exactly one copy of x and otherwise only monomers that polymerize as ε and carry $\emptyset$, then there is a polymer with those properties such that no monomer appears multiple times before or multiple times after the one copy of x, since if $uyvyw$ is a valid polymer then so is uyw. For each such monomer x, we can then check all such strings to see if they are valid polymers in polynomial space since the size is bounded by the number of implementation monomers. Since there are finitely many such x, there are finitely many formal species that can be implemented this way. The language of all formal polymers that can be implemented by a string of implementation monomers all of which carry $\emptyset$ can be recognized by an non-deterministic finite state automata (NFA) with states corresponding to each implementation monomer, and transitions from q_x to q_y reading $\pi(y)$ if and only if $(x, y) \in \rho'$ (with intermediate states if $|\pi(y)| > 1$), starting in state $q_{\vdash}$ and accepting in state $q_{\dashv}$. The set of valid formal strings can be recognized by a deterministic finite state automata (DFA) with the same principle. Then we can construct an NFA which accepts if the implementation-recognizing NFA accepts or the formal-recognizing DFA rejects or the input is any of the finitely many strings implemented in the μ of some species found above, and the atomic condition is satisfied if and only if that NFA accepts all strings, i.e. every formal string is either the interpretation of some implementation polymer or an invalid formal polymer.

To prove completeness, given any regular language L over alphabet Σ, by Lemma 2.2 we can construct an implementation species specification (Σ', ρ') and interpretation π, leaving $\mu(x) = \emptyset$ for all x, such that the set of interpretations of valid implementation polymers is exactly L. Furthermore, this construction is polynomial-time given an NFA that recognizes L. Where the formal species specification is $(\Sigma, (\Sigma \cup \{\vdash\}) \times (\Sigma \cup \{\dashv\}))$, so that all strings are valid formal polymers, then the atomic condition is satisfied if and

only if L is the set of all strings. Deciding this given an NFA that recognizes L is PSPACE-complete, so so is the atomic condition. □

Theorem 4.1. *The problem of, given a formal PRN (Σ, ρ, Rs), implementation PRN (Σ', ρ', Ri), and interpretation (μ, π), deciding whether that interpretation is a bisimulation is Π_2^0-complete.*

Proof. 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_2^0 statement. To prove completeness, we reduce from the uniform halting problem. Since PRNs can simulate Turing machines, 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 it can halt. Since the uniform halting problem is Π_2^0-complete, 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 $\Sigma = \{Q, H\}$ with $\rho = \{(\vdash, Q), (Q, \dashv), (\vdash, H), (H, \dashv)\}$ and one reaction $Q \to H$. The implementation PRN is the simulation of M from Example 2.2. $\mu(x) = \emptyset$ 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 $\pi(q_H) = H$.

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. There are species, namely q_0 and q_H, interpreting to each of them, and from arbitrary counts of these we see that the interpretation is surjective. 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. In any formal state with a Q, and any implementation state that interprets to it, there is one non-halting instantaneous description of M, and the statement that all such states can eventually do $Q \to H$ is equivalent to the statement that all instantaneous descriptions eventually halt. □

5 Single-Locus Networks

Both the DNA stack machine and the copy-tolerant Turing machine have the property that any reaction scheme involves at most one unbounded polymer and only affects a bounded region on that one polymer. Reaction schemes that violate this condition seem to pose problems for implementations, as the obvious methods of implementation would require interactions between two polymers, or distant points on the same polymer, that during the time they are connected cannot be modeled as linear polymers. It turns out that any polymer network using only reactions of this type plus splitting one polymer into two or joining two polymers at the ends can be implemented—up to bisimulation—by four hypothetical primitives, one of which is the reversible addition primitive used in the DNA stack machine. Though we do not have a DNA implementation for the other three, they seem reasonable to implement.

Definition 5.1. A reaction scheme is *single-locus* if the reactants contain at most one wildcard at the beginning of one polymer and one wildcard at the end of one polymer (which may or may not be the same polymer), which are not the same number, and no others, and each wildcard that appears in the reactants appears exactly once in the products in the same position. A polymer reaction network is single-locus if all reaction schemes are single-locus.

Theorem 5.1. *For any single-locus PRN (Σ, ρ, Rs), there is a PRN (Σ', ρ', Ri) and interpretation (μ, π) which is a bisimulation such that all reaction schemes in Ri are of one of the following four forms:*

$$*_1AB*_2 \to *_1CD*_2 \qquad \text{(Context-sensitive Replacement)}$$
$$*_1A\,*_2 + B \rightleftharpoons *_1C*_2 \qquad \text{(Monomer-dependent Replacement)}$$
$$*_1 \rightleftharpoons *_1E \qquad \text{(Reversible Addition)}$$
$$*_1 + *_2 \rightleftharpoons *_1I*_2 \qquad \text{(Reversible End-joining)}$$

Proof. An equivalent way of defining single-locus reaction schemes is that they are any reaction scheme where both reactants and products are of the form $*_1x\,*_2 + y^1 + \cdots + y^n$, where $x = x_1 \ldots x_m$ is a string over Σ with up to one $|$ and $y^i = y^i_1 \ldots y^i_{l_i}$ are strings over Σ, possibly with $*_1$ and/or $*_2$ absent in both reactants and products. Such a reaction is implemented with a monomer for each prefix w of x or any y^i, a monomer r^i_R for $1 \leq i \leq n$ for each reaction scheme r that has reactants $y^1 + \cdots + y^n$ and similarly r^i_P for the products, and monomers E and I that represent the empty string and a break, respectively. $\pi(w) = w$ and $\mu(w) = \emptyset$; $\pi(r^i_R) = \varepsilon$, $\mu(r^i_R) = y^1 + \cdots + y^i$, and similarly for r^i_P; $\pi(E) = \varepsilon$ and $\mu(E) = \emptyset$; and $\pi(I) = |$ and $\mu(I) = \emptyset$. Reaction schemes $*_1AE*_2 \rightleftharpoons *_1EA*_2$ and $*_1 \rightleftharpoons *_1E$ for every monomer A allow any arrangement of E's interspersed in a set of polymers to be reached from any other arrangement of E's in the same set of polymers. A reaction scheme $*_1 + *_2 \rightleftharpoons *_1I*_2$ allows two polymers to join at the ends to test whether a reaction involving both can occur. Reaction schemes $*_1wA*_2 \rightleftharpoons *_1(wA)E*_2$ where (wA) is a single monomer combine strings into one monomer, and $*_1E\,*_2 + Y^1 \rightleftharpoons *_1r^1_R*_2$ and $*_1r^i_R\,*_2 + Y^{i+1} \rightleftharpoons *_1r^{i+1}_R$ and similarly for r^i_P's combine other reactants onto the polymer which contained the string x. Then any single-locus reaction scheme can be implemented by $*_1AB*_2 \to *_1CD*_2$, where A and C are single monomers interpreted via π as the reactant and product strings in the one unbounded polymer or the end-joining of the two polymers which are unbounded on opposite sides, while B and D are single monomers interpreted via π as ε while interpreted via μ as the sets of additional finite strings in the reactants and products, respectively. To deal with the case where $*_1$ and/or $*_2$ are absent, we introduce implementation monomers F_l and F_r with reaction schemes $*_1 \rightleftharpoons F_l*_1$ and $*_1 \rightleftharpoons *_1F_r$, where $\mu(F_i) = \emptyset$ and $\pi(F_i) = |$ for $i \in \{l, r\}$, and require them to be added to the end of the implementation polymer and absorbed into the A monomer to confirm that the reaction is taking place at the appropriate end. □

6 Conclusions

The definition of a Polymer Reaction Network and bisimulation of PRNs allows us to formally verify molecular systems that cannot be described by the finite CRN model. In particular, the DNA stack machine computes by producing one species if and only if the computation should accept and another if and only if it should reject. Since bisimulation implies reachability equivalence, this verifies that the DNA stack machine is correct.

The proof that bisimulation is Π^0_2-complete means that both an algorithm to check whether an interpretation is a bisimulation and an algorithm to verify human-provided proofs that an interpretation is (or is not) a bisimulation are impossible for all cases. However, the cases which are impossible to check or verify are the cases where the implementation is much more complex relative to the formal PRN, which are almost certain never to appear in a practical application. One potential direction of future investigation is to find a subclass of implementation PRNs relative to formal PRNs which include all cases of practical interest, while also being easy (or at least possible) to check whether an interpretation is a bisimulation. For example, one might try limiting the number of trivial reactions that must be done to reach a state where a given formal reaction can be directly implemented, or restricting the presence of arbitrarily long sequences of monomers that polymerize as ε and carry $\emptyset$.

The result that any single-locus PRN can be implemented using reaction schemes of four types means that implementations of those four primitives could be combined into a correct (according to bisimulation)

implementation of any single-locus PRN. The stack machine implementation by Qian et. al. already implements something similar to reversible addition.

References

[1] ANGLUIN, D., ASPNES, J., AND EISENSTAT, D. Stably computable predicates are semilinear. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing* (2006), ACM, pp. 292–299.

[2] BENNETT, C. H. The thermodynamics of computation–a review. *International Journal of Theoretical Physics 21*, 12 (1982), 905–940.

[3] CARDELLI, L. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science 23*, 02 (2013), 247–271.

[4] CARDELLI, L., AND ZAVATTARO, G. On the computational power of biochemistry. In *Algebraic Biology*. Springer, 2008, pp. 65–80.

[5] CHEN, H.-L., DOTY, D., AND SOLOVEICHIK, D. Deterministic function computation with chemical reaction networks. In *DNA 2012: Proceedings of The 18th International Meeting on DNA Computing and Molecular Programming* (2012), vol. 7433 of *Lecture Notes in Computer Science*, Springer, pp. 25–42.

[6] CHEN, Y.-J., DALCHAU, N., SRINIVAS, N., PHILLIPS, A., CARDELLI, L., SOLOVEICHIK, D., AND SEELIG, G. Programmable chemical controllers made from DNA. *Nature nanotechnology 8*, 10 (2013), 755–762.

[7] DONG, Q. A bisimulation approach to verification of molecular implementations of formal chemical reaction networks. Master's thesis, SUNY Stony Brook, 2012.

[8] DOTY, D., AND HAJIAGHAYI, M. Leaderless deterministic chemical reaction networks. In *DNA 2013: Proceedings of The 19th International Meeting on DNA Computing and Molecular Programming* (2013), D. Soloveichik and B. Yurke, Eds., vol. 8141 of *Lecture Notes in Computer Science*, Springer International Publishing, pp. 46–60.

[9] HERMAN, G. T. Strong computability and variants of the uniform halting problem. *Mathematical Logic Quarterly 17*, 1 (1971), 115–131.

[10] HJELMFELT, A., WEINBERGER, E. D., AND ROSS, J. Chemical implementation of neural networks and Turing machines. *Proceedings of the National Academy of Sciences 88*, 24 (1991), 10983–10987.

[11] KOZEN, D. *Automata and computability*. Springer, 1997.

[12] LAKIN, M. R., YOUSSEF, S., CARDELLI, L., AND PHILLIPS, A. Abstractions for DNA circuit design. *Journal of The Royal Society Interface 9*, 68 (2012), 470–486.

[13] LAKIN, M. R., YOUSSEF, S., POLO, F., EMMOTT, S., AND PHILLIPS, A. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics 27*, 22 (2011), 3211–3213.

[14] MAGNASCO, M. O. Chemical kinetics is Turing universal. *Physical Review Letters 78*, 6 (1997), 1190–1193.

[15] MEYER, A. R., AND STOCKMEYER, L. J. 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* (1972), IEEE, pp. 125–129.

[16] MILNER, R. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[17] QIAN, L., SOLOVEICHIK, D., AND WINFREE, E. Efficient Turing-universal computation with DNA polymers. In *DNA computing and molecular programming*. Springer, 2011, pp. 123–140.

[18] RACKOFF, C. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science 6*, 2 (1978), 223–231.

[19] SHIN, S. W. *Compiling and verifying DNA-based chemical reaction network implementations*. PhD thesis, California Institute of Technology, 2011.

[20] SOLOVEICHIK, D., COOK, M., WINFREE, E., AND BRUCK, J. Computation with finite stochastic chemical reaction networks. *Natural Computing 7*, 4 (2008), 615–633.

[21] SOLOVEICHIK, D., SEELIG, G., AND WINFREE, E. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences 107*, 12 (2010), 5393–5398.

[22] STOCKMEYER, L. J., AND MEYER, A. R. Word problems requiring exponential time (Preliminary Report). In *Proceedings of the fifth annual ACM symposium on Theory of computing* (New York, NY, USA, 1973), STOC '73, ACM, pp. 1–9.