Efficient Turing-Universal Computation with DNA Polymers

Lulu Qian[1], David Soloveichik[4], and Erik Winfree[1,2,3]

[1] Bioengineering, California Institute of Technology,
Pasadena, CA 91125, USA
luluqian@caltech.edu
[2] Computer Science
[3] Computation & Neural Systems, California Institute of Technology,
Pasadena, CA 91125, USA
winfree@caltech.edu
[4] Computer Science & Engineering, University of Washington,
Seattle, WA 98195, USA
dsolov@u.washington.edu

Abstract. Bennett's proposed chemical Turing machine is one of the most important thought experiments in the study of the thermodynamics of computation. Yet the sophistication of molecular engineering required to physically construct Bennett's hypothetical polymer substrate and enzymes has deterred experimental implementations. Here we propose a chemical implementation of stack machines — a Turing-universal model of computation similar to Turing machines — using DNA strand displacement cascades as the underlying chemical primitive. More specifically, the mechanism described herein is the addition and removal of monomers from the end of a DNA polymer, controlled by strand displacement logic. We capture the motivating feature of Bennett's scheme: that physical reversibility corresponds to logically reversible computation, and arbitrarily little energy per computation step is required. Further, as a method of embedding logic control into chemical and biological systems, polymer-based chemical computation is significantly more efficient than geometry-free chemical reaction networks.

1 Introduction

With the birth of molecular biology 70 years ago came the realization that the processes within biological cells are carried out by molecular machines, and that the most central processes involved the manipulation of information-bearing polymers. Roughly 30 years ago, Charles Bennett took that vision one step further by recognizing that arbitrarily complex information processing could be carried out, in principle, by molecular machines of no greater complexity than those already observed in nature [5,6]. Based on the intrinsic reversibility of chemical reactions, Bennett used this insight to give a thermodynamic argument that there is no fundamental energetic cost to computation — only

Y. Sakakibara and Y. Mi (Eds.): DNA 16, LNCS 6518, pp. 123–140, 2011.
© Springer-Verlag Berlin Heidelberg 2011

a cost to erase data. This conclusion derives from four principles: (1) as Landauer observed [15], making a logically irreversible decision entails an energetic expenditure of $kT \ln 2$, and thus there is an unavoidable cost to irreversible logical operations; (2) being logically reversible is not enough to ensure low-energy computation, since it is possible to implement reversible logic using irreversible mechanisms; (3) a physically reversible system with an essentially linear state space can be biased ever-so-slightly forward, in which case progress is made despite involving a Brownian random walk, with the mean speed being linear in the (arbitrarily near zero) energy expended per step; and (4) any logically irreversible computation can be recast with a minimal number of extra computational steps [5,7] as a logically reversible computation that requires irreversible operations only when preparing input and output during repeated use. It's intriguing to ask whether Landauer's and Bennett's principles have any bearing on the remarkable efficiency of living things, but cellular processes typically use several times more energy than needed for logical irreversibility. On the other hand, modern electrical computers expend many orders of magnitude more energy than required by logical irreversibility, presenting the challenge of building computers that have the efficiency Bennett argued is possible.

Direct implementation of Bennett's hypothetical chemical Turing machine has been hampered by our inability, as yet, to engineer molecular machinery to spec. Len Adleman's laboratory demonstration of a DNA computing paradigm for solving NP-complete problems [1] ignited renewed interest in the molecular implementation of Turing machines. Early theoretical proposals made use of existing enzymes but required a series of laboratory manipulations to step the molecular Turing machines through their operational cycle [20,2,23], while later theoretical proposals suggested how autonomous molecular Turing machines could be built but made use of hypothetical enzymes or DNA nanostructures [14,4,11,28,12]. Experimental demonstrations of autonomous biomolecular computers implemented weaker models of computation such as digital circuits or finite state machines [22,3]. Two-dimensional molecular self-assembly is Turing universal [26], implementable with DNA tiles [21], and can be physically and logically reversible [27], but it has the distinct disadvantage of storing the entire history of its computation within a supramolecular complex — it's bulky.

Recent work has pointed to an alternative to geometrical organization (in polymers or crystals) as the basis for Turing-universal molecular computation: abstract chemical reaction networks (CRNs) with a finite number of species in a well-mixed solution are structurally simple enough (essentially geometry-free) that in principle arbitrary networks can be implemented with DNA [25], yet they are (probabilistically) Turing universal [24]. This Turing universal computation using geometry-free chemical reaction networks is theoretically accurate and reasonably fast (only a polynomial slowdown), but requires molecular counts (and therefore volumes) that grow exponentially with the amount of memory used [16,24]. In contrast, reaction networks using heterogeneous polymers — the simplest kind of geometrical organization, as in Bennett's vision — can store all information as strings within a single polymer, therefore requiring volume that

grows only linearly with the memory usage [6,14]. Further, geometry-free models are not energy efficient, requiring much more than Landauer's energy limit because the computation must be driven irreversibly forward to avoid error. Here, we combine the advances in geometry-free CRN implementation [25] with a simple DNA polymer reaction primitive to obtain a plausible DNA implementation of time- and space- and energy-efficient Turing-universal computation. Our construction requires a small fixed number of polymers, thereby having the same efficient linear memory/volume tradeoff as Bennett's hypothetical scheme; it also has a time complexity nearly as good (only a quadratic slowdown). As in Bennett's scheme, the time complexity scales linearly with energy use (for small energies). Both constructions can perform irreversible computation using the minimum achievable amount of energy per step, $kT \ln m$, where m is the mean number of immediate predecessors to the logical states of the Turing machine simulation. (This energy bound is 0 for reversible Turing machines).

Our constructions will consist of two parts. First, a geometry-free chemical reaction network, and secondly, reactions involved in polymer modification. While the polymer modification reactions will perform the essential job of information storage and retrieval, the geometry-free reaction network will perform the logic operations. We describe the necessary elements for the implementation of relevant geometry-free chemical reaction networks in section 2. In the following section 3 we describe the polymer reactions. Based on these two DNA implementation schemes, section 4 shows how they can be used to efficiently simulate stack machines. Finally, in section 5 we show how Bennett's logically reversible Turing machines can be implemented with physically reversible DNA reactions. We conclude by evaluating our contributions and pointing out room for further improvement.

2 Irreversible and Reversible Chemical Reaction Networks

In this section we discuss the components necessary for the implementation of the geometry-free chemical reaction network part of our constructions.

Recent work has proposed a DNA implementation of arbitrary (geometry-free) chemical reaction networks [25], with which we assume the reader is familiar. (Even so, the construction given here is self-contained.) However, thermodynamic reversibility was not considered. Indeed, reversible reactions would simply correspond to two separate forward and reverse reactions, with both reactions having to be independently driven irreversibly by chemical potential energy provided by DNA fuels. This is wasteful for the consumption of both energy and fuel reagents. The construction we develop in this section is entirely physically reversible in the sense that firing a sequence of forward reactions and then the reverse sequence of the corresponding reverse reactions brings the chemical system into the same exact physical state as it was in the beginning, including recovery of fuel reagents and any energy used.

The major challenge in adapting the scheme of ref. [25] to implement reversible chemical reactions in a physically reversible manner, is that the exact DNA strand representing a particular signal species is a function of not just the signal, but also

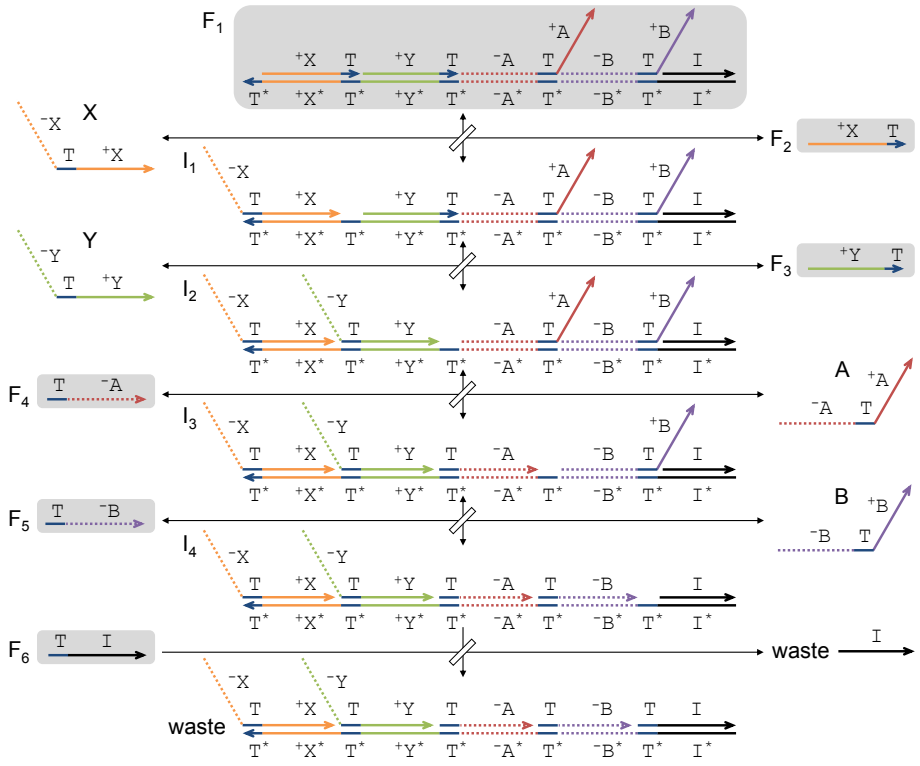

Fig. 1. The implementation of the formal bimolecular reaction $X + Y \rightarrow A + B$ using history-free signal species X, Y, A, B. Each strand displacement reaction is shown, with arrowed thin black lines connected by diagonal rectangles indicating reactants, products and reversibility. F_1 through F_6 are fuel species mediating the formal reaction, and are present in high concentration. F_1 is unique to this reaction, while F_2 through F_6 may be shared with other reactions. I_1 through I_4 are intermediates of the formal reaction and they are all unique to this reaction. Domains ${}^{+}X$, ${}^{-}X$, ${}^{+}Y$, ${}^{-}Y$, ${}^{+}A$, ${}^{-}A$, ${}^{+}B$, ${}^{-}B$ are specific to formal species X, Y, A, B respectively, while domain I is used for the irreversible step in all reactions, and the short toehold domain T is used universally wherever a toehold is needed. The asterisk indicates Watson-Crick complementary domains, e.g. ${}^{+}Y^{*}$ is complementary to ${}^{+}Y$.

of the formal reaction that produced it. Specifically, all signal strands contain a "history" domain that holds the strand in an inactive form before release from a DNA fuel complex. So even if we were to make every strand displacement step reversible, the reverse reaction of $X \rightarrow Y$ would only be able to uptake signal species Y that have the correct history domains for this reaction, rather than the entire population of Y which may have been generated by other reactions.

In order to solve this problem, we develop a "history-free" implementation of arbitrary chemical reaction networks. (Cardelli has proposed an elegant and even further reduced scheme [10] that is also history-free, but it appears unsuitable for our polymer reaction construction.) We describe an irreversible scheme,

that with slight modifications can become reversible. In addition to making the reversible scheme straightforward, the history-free signal strand motif simplifies the correspondence between the abstract CRN and the DNA implementation: each CRN species now corresponds to exactly one DNA species.

Like ref. [9], but unlike ref. [25], we use stochastic semantics in this paper where reactions manipulate integer molecular counts of the reacting species, rather than real-valued concentrations. The applicable kinetic and thermodynamic laws are widely known from the consideration of small-scale chemical systems. Unlike the quantitative kinetics requirements of ref. [25], in the context of this paper a successful implementation of an irreversible reaction such as $X + Y \rightarrow A + B$ must simply be qualitatively correct by satisfying two conditions: First, there must be some overall irreversible reaction pathway that first consumes a molecule of X, a molecule of Y, and then produces a molecule of A and a molecule of B. Second, the reaction pathway must become irreversible at some point only *after* X and Y have been consumed. If the pathway were to become irreversible before Y is consumed, then in the absence of Y, X would still be used up. An implementation of a reversible reaction $X + Y \rightleftharpoons A + B$ must be a reversible reaction pathway that first consumes a molecule of X, a molecule of Y, and then produces a molecule of A and a molecule of B. While we do not explicitly address the question of quantitatively correct reaction kinetics, our constructions respect the usual scaling laws for kinetics of unimolecular, bimolecular, and higher-order reactions; similar techniques as in ref. [25] could applied to these constructions.

Fig. 1 shows the history-free implementation of the irreversible reaction $X + Y \rightarrow A + B$, and Fig. 2 shows the corresponding implementation of the reversible reaction $X + Y \rightleftharpoons A + B$.

In Fig. 1, fuel DNA species F_1, F_2, F_3, F_4, F_5 and F_6 are initially present in high concentration, and we assume they remain present in high concentration throughout. Signal DNA species X, Y, A and B are present in low amounts relative to the fuel species and indicate meaningful signals. To make the reaction module composable, all signal DNA species are of the same form, and allow the coupling of such formal reactions together. (They are also of the same form as signal species in another DNA strand displacement network architecture [19], which allows even broader couplings.) All signal species have one short toehold domain in the middle, one long recognition domain "$-$" on the $5'$ end and another long recognition domain "$+$" on the $3'$ end. The bottom strand of fuel F_1 has five long recognition domains connected by five short toehold domains. Initially, the left-most toehold domain is single-stranded and is thus available for binding, and the other four toeholds are double-stranded and thus sequestered.

Signal X first binds to fuel F_1 by the exposed toehold and branch migration occurs through domain ${}^{+}X$. The top strand F_2 will fall off when it's only held to the bottom strand by the toehold and leave F_1 as intermediate product I_1. (This is the principle of toehold exchange [29,30].) Compared to F_1, the bottom strand of I_1 has its first toehold covered and the second toehold revealed. Signal Y then binds to I_1 at the second toehold, branch migrates to the $3'$ end of ${}^{+}Y$ and kicks

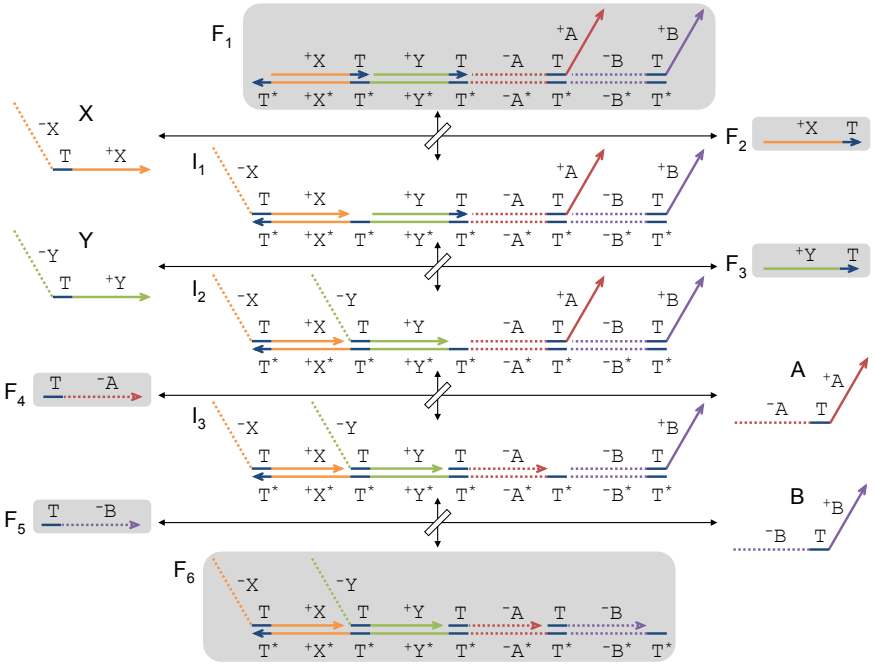

Fig. 2. The implementation of the formal bimolecular reaction $X + Y \rightleftharpoons A + B$ using history-free signal species X, Y, A, B. F_1 through F_6 are fuel species mediating the formal reaction, and are present in high concentration. F_1 and F_6 are unique to this reaction, while F_2 through F_5 may be shared with other reactions. I_1 through I_3 are intermediates of the formal reaction and they are all unique to this reaction. Reaction and domain notation is as in Fig. 1.

off the top strand F_3, producing intermediate I_2. The bottom strand of I_2 has its third toehold revealed and all the other toeholds covered. Now F_4 binds to I_2 at the third toehold, releases signal A and leaves intermediate I_3. F_5 binds to I_3 at the fourth toehold, releases signal B and leaves intermediate I_4. All the above reactions can be reversed by F_2, F_3, A and B reacting with I_1, I_2, I_3 and I_4 respectively. Finally, F_6 binds to I_4 at the last toehold and displaces the last top strand — which has no toehold domain by which to initiate the reverse reaction. Because of this last irreversible step, the overall reaction is irreversible.

If there is only signal Y but not X, nothing will happen because Y cannot directly react with any of the fuels. If there is only signal X but not Y, only the first step can happen and the backward reaction $F_2 + I_1 \rightarrow X + F_1$ ensures X is not permanently consumed. So, A and B will be produced, and X and Y consumed, only if both X and Y were initially present.

In Fig. 2, we simply remove domain I from F_1 in Fig. 1 to make the formal reaction reversible. Therefore, the F_6 of Fig. 1 becomes unnecessary and the final product of the forward reaction becomes the new F_6, which also serves as

the first fuel of the backward reaction. Now there are only three intermediates instead of four. Note also that increasing the concentration of F_1 speeds up the forward reaction, while increasing the concentration of F_6 speeds up the backwards reaction — so this reaction can be individually tuned to be unbiased or biased forward or backward to any desired extent.

In considering the energy use of the reversible stack machine implementation it is important to verify that we are not cheating when we ignore the entropic contribution of the concentration changes of the fuels in the process of computation. Luckily, in the limit of large molecular counts of the fuel species, the contribution of the changing partial pressures of the fuels to the energy consumed in a reaction occurrence is independent of the history of the previous reaction events, and can be considered fixed for each reaction. Therefore we can use the fuels to provide a fixed forward bias for each reaction, or no bias if we start with equilibrium concentrations of the fuels.

Proper functioning of both the irreversible and reversible reaction modules involves a Brownian exploration of the system's state space to test if the preconditions for the reaction are met. Necessarily, this exploration must be reversible, in case the preconditions aren't met. This basic structure — reversible exploration to determine whether a subsequent step is possible — is used again and again in our constructions below. Despite the lack of determinism for individual steps, no species is incorrectly consumed or incorrectly produced — and so long as the system is biased forward (or unbiased), any species that should be produced will be produced eventually. In this sense, there can be no errors.

Creating new reactions between a set of signal species only requires the construction of appropriate fuel species and is thus programmable by the choice of fuel species without any alteration of the signal species themselves. Therefore, to implement a system of reactions, the fuels for each reaction may be individually constructed, and the union of these fuels correctly implements the full system.

We will consider stochastic dynamics for a small integral number of each signal species (typically a single copy each) reacting in a volume V in which the concentrations of fuel species are maintained constant. In a perfect implementation of a bimolecular reaction, the reaction rate should slow down exactly in proportion to V. It is easy to see that our multistep DNA implementation should have the same asymptotic scaling for kinetics. Consider the irreversible case. Among all the forward strand displacement reactions, only the second step (Y displacing F_3) is a bimolecular reaction between two low concentration species (the signal species and their reaction intermediates). The forward rate of this reaction (the second step) will scale as $1/V$. All the other steps are bimolecular reactions between one high concentration (fuel) species and one low concentration species and thus will be fast. Further, the backward reactions are either slow or (in the case of $F_3 + I_2 \to Y + I_1$) exactly balanced by a fast reaction ($F_4 + I_2 \to A + I_3$) in the forward direction. Thus, the second step is the rate limiting step for the reaction pathway. (A similar argument holds for both forward and backward reactions in the reversible scheme.)

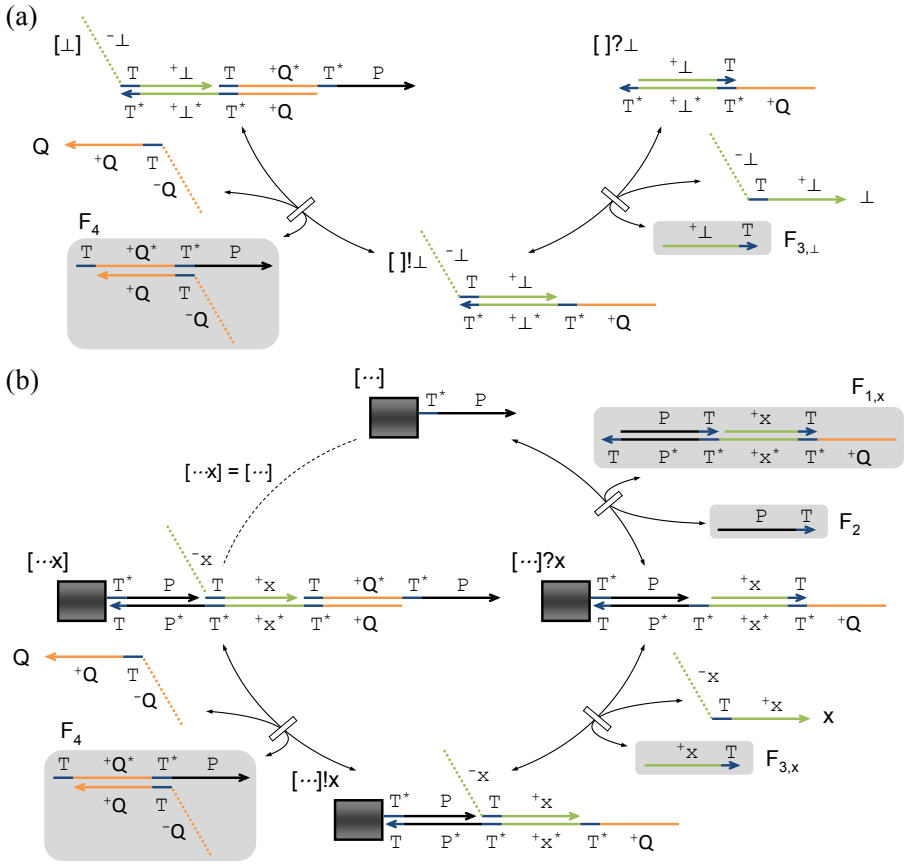

Fig. 3. The implementation of the formal polymer reaction $[\cdots] + x \rightleftharpoons [\cdots x] + Q$. Intuitively, in the forward direction this reaction adds a new monomer to the end of the polymer, releasing Q to signal completion. In the backward direction, Q detaches the last monomer from the polymer. **(a)** The beginning of the polymer and the strand displacement reactions when the stack is empty, i.e., implementation of reaction (1b). Note that the first monomer is $\perp$ to indicate the end of the stack for the stack machine simulation. **(b)** The strand displacement reactions for the monomer addition / removal cycle, i.e., implementation of reaction (1a). Clockwise: adding a new monomer; counterclockwise: removing the last monomer. The dark box indicates the 'left' side of the polymer, with an arbitrary number of subunits. The dotted line indicates conceptually encapsulating the x subunit repeat block within the dark box. Reaction and domain notation is as in Fig. 1.

Unimolecular reactions such as $X \rightarrow A$ can be implemented analogously with the appropriate shortening or extension of fuel F_1, as can higher order reactions such as $X + Y + Z \rightarrow A + B + C$ and asymmetric reactions such as $X \rightarrow A + B$. (Similar modifications can be used for reactions $X \rightleftharpoons A$, $X + Y + Z \rightleftharpoons A + B + C$, $X \rightleftharpoons A + B$, etc.) Extending our construction to reactions of order n yields $n - 1$ forward steps that involve bimolecular reactions between two low concentration

species. Therefore we would encounter a slowdown scaling as $1/V^{n-1}$, with the unimolecular implementation being independent of volume, as is required to agree with standard chemical kinetics.

3 A Reversible Polymer Addition Primitive

Finite CRNs by definition involve a finite number of possible molecular species, and this limits their behavioral complexity. Extending CRNs to include polymers allows one to give a finite specification of a molecular system involving potentially an infinite number of distinguishable species — polymers of different lengths and with different sequences — that interact according to a finite number of local rules. A variety of extensions of CRNs to polymers (and other combinatorial structures) have been considered, differing in the types of local rules (e.g. end-localized reactions, interior-localized reactions, polymer joining and scission) and the types of polymer structures (e.g. strictly linear, branched, networks with cycles) that are allowed [13,8,11]. All these natural CRN extensions can efficiently simulate Turing machines (c.f. [11]). However, whereas these extensions were designed to be general for modeling biochemical systems, our interest here is in a language for specifying polymer systems that can be implemented with DNA strand displacement reactions — and fully general modeling languages presently may be too difficult to compile into DNA.

Therefore, we focus on a very limited subset of polymer reactions that simultaneously allows implementation with DNA and is capable of efficient simulation of Turing machines. Specifically we make use of a single reversible reaction mechanism that (in the forward direction) appends a desired subunit onto the polymer while releasing a confirmation signal, and that (in the reverse direction) upon receipt of a query detaches a subunit from the polymer. In our construction, each polymer has a fixed end and a growing end. Appending and detaching can only occur on the growing end. All polymers begin at their fixed ends with a special subunit, $\perp$, followed by an arbitrary sequence of subunits from the finite set Σ. Formally, a polymer with sequence $\perp w$, where $w \in \Sigma^*$, is written as $[\perp w]$. The subunits themselves may also exist as free species, $x \in \Sigma$, as may the special subunit $\perp$. The query/confirmation species is called Q. Then the implemented reversible polymer addition reaction may be written as

$$[\cdots] + x \rightleftharpoons [\cdots x] + Q \tag{1}$$

where informally $[\cdots]$ represents a polymer with some sequence $\perp w$ and $[\cdots x]$ represents that same polymer extended to sequence $\perp wx$. Formally, this single polymer reaction schema represents an infinite family of specific reactions

$$[\perp w] + x \rightleftharpoons [\perp wx] + Q \tag{1a}$$

for all $w \in \Sigma^*$ and $x \in \Sigma$, as well as the base case

$$[\,]?\perp + \perp \rightleftharpoons [\perp] + Q \tag{1b}$$

that enables detecting that the polymer is a monomer.

Fig. 3 shows the DNA implementation of this polymerization primitive. The formal species Q and $\perp$ and each $x \in \Sigma$ are implemented as finite CRN species using the same history-free motif described in section 2. The polymer itself is a chain of information-bearing subunits (green) spliced together by strands using the P and ${}^{+}Q$ domains. To mediate the desired reactions, a number of high-concentration fuels are used: F_2 and F_4 are independent of the monomer type, while a separate $F_{1,x}$ and $F_{3,x}$ is needed for each $x \in \Sigma$, as well as $F_{3,\perp}$.

The reversible and exploratory nature of the DNA implementation's reactions are essential to its function. For example, for $[\cdots]$ to react with a specific x that may be present, it may first react with several different $F_{1,y}$ to produce the intermediate $[\cdots]?y$ that attempts to add y to the polymer — but so long as y is not present in solution, this attempt fails, and the reaction reverses to recreate $[\cdots]$ with the help of F_2. Eventually, $[\cdots]$ will react with $F_{1,x}$ to produce $[\cdots]?x$, which can react with x to proceed to the next step, the $[\cdots]!x$ intermediate. Finally a reaction with F_4 brings the polymer back to its canonical state, $[\cdots x]$, but with the new subunit appended and the confirmation Q produced. Of course, since all the reactions are reversible, Q can also serve as a query and reverse $[\cdots x]$ to $[\cdots]!x$, after which the appropriate $F_{3,x}$ may succeed in detaching x, so that F_2 can bring $[\cdots]?x$ back to $[\cdots]$. A set of different $F_{3,y}$ make sure the detaching will happen no matter what subunit is on the growing end of the polymer. For the base case, the reverse can only go as far as $[\cdots]?\perp$, which has a special form and thus won't be able to react with F_2. This ensures that $\perp$ is always the first subunit of the polymer.

While the polymer reactions by themselves do nothing more than push and pop subunits back and forth onto and off of the end of the polymer, these reactions can be controlled and driven by a simultaneously active finite CRN that interacts with the formal species Q, $\perp$, and $x \in \Sigma$. To append a subunit to the polymer, the CRN must simply produce the desired species x and then wait for the confirmation Q. To read a subunit off the polymer, the CRN must simply produce the query Q and then wait for the arrival of some subunit $x \in \Sigma$ or else $\perp$.

Our polymer reaction primitive is also essentially bimolecular, and hence the kinetics also scales as $1/V$.

4 Irreversible Stack Machine Implementation

4.1 Definition of Stack Machines

In this paper we provide a direct molecular implementation of stack machines rather than the more familiar Turing machines because they are particularly matched to the kind of polymer operation we have available, which accesses the polymer at one end only. The stack machine model of computation intuitively consists of a finite state control together with memory in the form of a finite number of stacks. Each stack can hold an arbitrary sequence of symbols but can only be accessed at one end: stack operations include pushing a new symbol onto a stack, or popping a symbol off a stack, as well as detecting an empty stack.

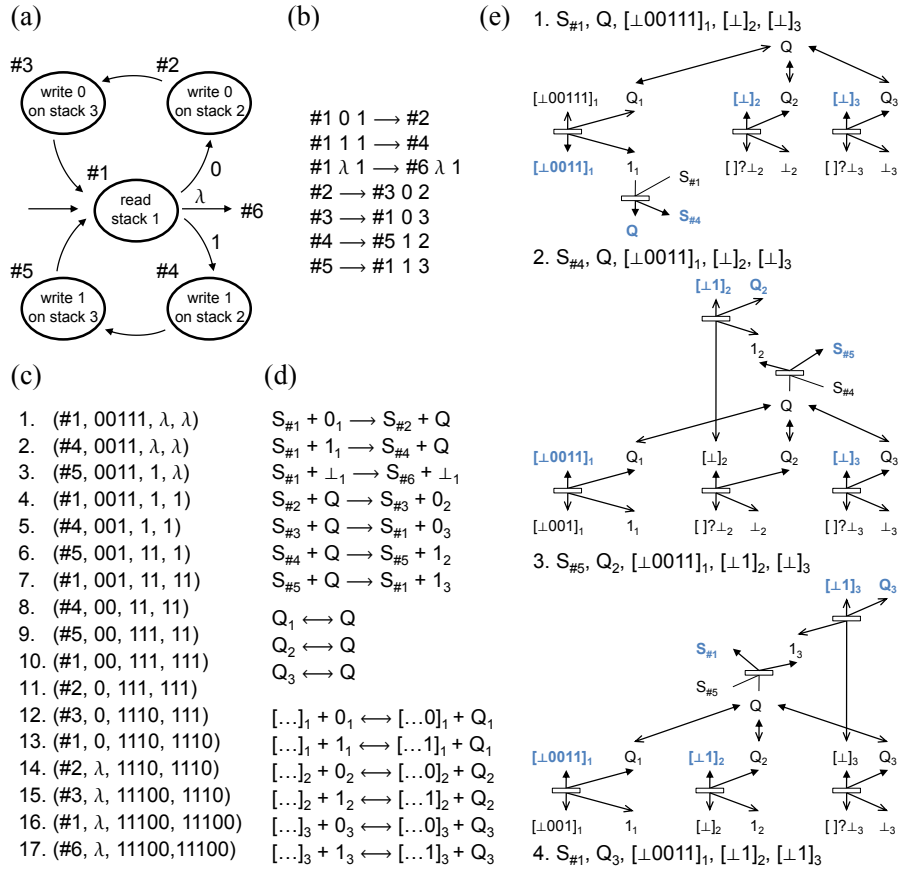

Fig. 4. Example execution of a stack machine program. **(a)** Diagrammatic representation of a stack machine that reads a string on stack 1 and writes a reversed copy onto both stack 2 and stack 3. **(b)** Transition rules for the same stack machine. **(c)** Execution history of stack machine configurations for computation with input string 00111. **(d)** Polymer CRN reactions for the same stack machine. Recall that polymer reaction schema of form (1) expand to reactions of forms (1a) and (1b). **(e)** Reaction pathways within the polymer CRN implementation, illustrated for the first three steps going from configuration 1 to configuration 4. Solid arrowheads indicate the direction of computation that is ratcheted forward by this step's irreversible reaction, and blue species represent the canonical endpoint species for each step of the computation. Each reaction is shown with longer polymer species on top, which is why in some steps reactions go "up" and in other steps reactions go "down".

Input is provided as the initial sequence of symbols in the first stack. While stack machines with only 1 stack are known to be less than Turing universal, 2 stacks are enough for universality. Similarly, while stack machines with just 1 symbol (also known as counter machines or register machines) are universal [17], they are exponentially slower than Turing machines, and efficient simulation of

Turing machines becomes possible only with 2 symbols or more. In fact, multi-stack multi-symbol stack machines can simulate multi-tape multi-symbol Turing machines with no slow-down (and vice versa). Consequently, many stacks and many symbols are preferred for elegance and efficiency. This is what we achieve with our DNA polymer implementation.

We allow any finite alphabet of symbols Σ, with an additional symbol $\lambda \notin \Sigma$ to indicate that the stack is empty. We specify stack machine transition rules in a somewhat non-standard manner — one that is better suited to discussing reversibility (see next section). There are 4 types of transition rules:

$$\begin{array}{ll} 1. & \alpha\ x\ i \longrightarrow \beta\ y\ j \\ 2. & \alpha\ x\ i \longrightarrow \beta \\ 3. & \alpha \longrightarrow \beta\ y\ j \\ 4. & \alpha\ \lambda\ i \longrightarrow \beta\ \lambda\ i \end{array}$$

where α, β are states, $x, y \in \Sigma$ are symbols, and $i, j \in \{1, \ldots, n\}$ are stacks. A transition rule of type (1) means: when in state α and the top symbol on stack i is x, pop it off and push symbol y onto stack j, transitioning to state β. A transition rule of type (2) means: when in state α and the top symbol on stack i is x, pop it off and transition to state β. A transition rule of type (3) means: when in state α, push symbol y onto stack j, transitioning to state β. A transition rule of type (4) means: when in state α and stack i is empty, move to state β. Note that the stack on the left and right must be the same for rule type (4).

A configuration of a stack machine consists of a state α and the contents of stacks $1, \ldots, n$. Computation begins in the designated start state (typically #1) with the input to the computation on the stacks (typically stack 1 has an input string, and the remaining stacks are empty) and proceeds by execution of applicable rules until no rule is applicable (typically, the machine will be in a 'halting state' that does not appear in the LHS of any rule). The contents of the stacks after halting may be considered the output of the machine.

We say that the stack machine is (syntactically) deterministic if for every configuration, there is at most one applicable transition rule. This can easily be verified by checking that for each state α, either all rules with α on the LHS read from the same stack with at most one transition per read symbol, or else there is at most one rule of type (3).

An example stack machine and computation is shown in Fig. 4abc.

4.2 Reactions Corresponding to the Transition Rules

The polymer CRN implementation of an n-stack machine with symbol alphabet Σ will comprise a finite collection of CRN reactions, one for each transition rule and one for each stack, combined with a polymer reaction for each stack. We require n distinct types of polymer reactions to implement the n stacks. We obtain them by generating n independent copies of the polymer reaction primitive of Fig. 3 wherein every species and domain is subscripted by i to

indicate that the domains are unique to that polymer type (with the exception of the universal toehold T). Thus the fuels are also unique to the polymer type. Because of the unique domains and fuels, the reactions steps of Fig. 3 will never result in crosstalk between polymers of different type. Therefore, generating x_i or Q_i will result in pushing x onto or popping a symbol off of stack i specifically. Later it will be convenient to have a single 'query' species Q that interconverts with the Q_i to (reversibly) read any stack; the symbol x_i that is read indicates which stack it came from, so no information is lost.

In summary, for every stack $i \in \{1, \ldots, n\}$ and symbol $x \in \Sigma$, we have a distinct molecular species x_i. Further, for every stack i we have species $\perp_i$, the CRN reaction

$$Q \rightleftharpoons Q_i$$

and the polymer reaction

$$[\cdots]_i + x_i \rightleftharpoons [\cdots x]_i + Q_i.$$

If a polymer contains only the symbol $\perp_i$ then the polymer represents stack i being empty, i.e. having content λ.

To run the system, we start with (1) exactly one molecule of each stack polymer type, each containing the input strings for its corresponding stack; (2) exactly one molecule of the state species $S_{\#1}$; and (3) exactly one molecule of the 'query' species Q, which rapidly interconverts into the Q_i required for each stack. Our system will respect the conserved property that there is always either exactly one Q or one Q_i molecule for some stack, or else there is exactly one molecule x_i representing some symbol on some stack.

Now, each stack machine transition rule 1-4 corresponds to a single polymer CRN reaction as follows:

$$\begin{array}{lllll}
1. & \alpha\ x\ i \longrightarrow \beta\ y\ j & \Rightarrow & S_\alpha + x_i \rightarrow S_\beta + y_j \\
2. & \alpha\ x\ i \longrightarrow \beta & \Rightarrow & S_\alpha + x_i \rightarrow S_\beta + Q \\
3. & \alpha \longrightarrow \beta\ y\ j & \Rightarrow & S_\alpha + Q \rightarrow S_\beta + y_j \\
4. & \alpha\ \lambda\ i \longrightarrow \beta\ \lambda\ i & \Rightarrow & S_\alpha + \perp_i \rightarrow S_\beta + \perp_i
\end{array}$$

Illustrative steps for the implementation of the example stack machine are shown in Fig. 4de. Note that despite all the reversible reactions, each time an irreversible CRN reaction (corresponding to a transition rule) occurs, the overall computation ratchets forward.

Run in a reaction volume V, each irreversible step will take average time $O(V)$ since its rate scales as $O(1/V)$. So to simulate a stack machine (or TM) whose computation runs in time t using space s, our DNA implementation will take time $O(tV)$. However, the reaction volume must be large enough to contain the polymers, which will be $O(s)$ subunits long, and hence $V = O(s)$. Taking the worst-case bound $s = O(t)$, the overall time required by the DNA stack machine implementation is $O(t^2)$. In contrast, because Bennett's hypothetical polymer-chemistry Turing machine has no bimolecular reactions between low-concentrations species (all reactions are between a single polymer tape and high-concentration enzymes), its time requirement is just $O(t)$ — better than ours.

5 Reversible Stack Machine Implementation

Given a stack machine defined as in the preceding section, the set of reverse rules is formed by switching the left-hand side and the right-hand side of all rules. We say the stack machine is reversible if the set of reverse rules is deterministic.

To implement a stack machine that can proceed either forward or backward in chemistry we can use reversible reactions:

$$\begin{array}{lllll} 1. & \alpha\ x\ i \longrightarrow \beta\ y\ j & \Rightarrow & S_\alpha + x_i & \rightleftharpoons S_\beta + y_j \\ 2. & \alpha\ x\ i \longrightarrow \beta & \Rightarrow & S_\alpha + x_i & \rightleftharpoons S_\beta + Q \\ 3. & \alpha \longrightarrow \beta\ y\ j & \Rightarrow & S_\alpha + Q & \rightleftharpoons S_\beta + y_j \\ 4. & \alpha\ \lambda\ i \longrightarrow \beta\ \lambda\ i & \Rightarrow & S_\alpha + \perp_i & \rightleftharpoons S_\beta + \perp_i \end{array}$$

5.1 Simulating a Reversible Turing Machine

Most theoretical work on reversible computing uses Turing machines rather than stack machines. Are there non-trivial reversible stack machines according to the above definition? Are there universal reversible stack machines according to the above definition?

We can take the path of showing that reversible stack machines can simulate known reversible Turing machines. For simplicity let us consider a binary, reversible Turing machine with one tape that is bounded on the left and is infinite on the right; futher, we require that the Turing machine never tries to read past the left end of the tape. (For example ref. [18] describes such a Turing machine that is universal, although it is slow; multi-tape reversible Turing machines are faster, and can be similarly implemented with polymer CRNs.) We can consider three types of Turing machine transition rules, using Bennett's notation [5]:

$$\begin{array}{ll} 1. & \alpha\ x \longrightarrow \beta\ y \\ 2. & \alpha\ / \longrightarrow \beta- \\ 3. & \alpha\ / \longrightarrow \beta+ \end{array}$$

where α, β are states, and $x, y \in \{0, 1\}$ are symbols. The first rule means that when in state α with the head reading symbol x, transition to state β overwriting x with y. The second and third rules indicate that when in state α move left or right respectively, without reading from or writing to the tape. The reverse of rule $\alpha\ x \longrightarrow \beta\ y$ is $\beta\ y \longrightarrow \alpha\ x$. The reverse of rule $\alpha\ / \longrightarrow \beta-$ is $\beta\ / \longrightarrow \alpha+$ and vice versa.

We represent the tape using two stacks. Everything to the left of the head is on stack 1 with the current symbol on top. Everything to the right of the head is on stack 2 with the symbol to the right of the head on top. The infinity of 0's past the rightmost 1 on the Turing machine tape is implicitly represented such that the topmost symbol on stack 2 can only be 1, if the stack is not empty. We convert Turing machine transition rules to stack machine transition rules as follows, where $x, y \in \{0, 1\}$:

1. $\alpha\ x \longrightarrow \beta\ y \quad \Rightarrow$

$$\alpha\ x\ 1 \longrightarrow \beta\ y\ 1$$

2. $\alpha\ / \longrightarrow \beta -\quad \Rightarrow$

$$\begin{aligned}
\alpha\ \ 0\ 1 &\longrightarrow \sigma_1\ 0\ 1\\
\alpha\ \ 1\ 1 &\longrightarrow \beta\ \ 1\ 2\\
\sigma_1\ \lambda\ 2 &\longrightarrow \sigma_2\ \lambda\ 2\\
\sigma_1\ 0\ 2 &\longrightarrow \sigma_4\ 0\ 2\\
\sigma_1\ 1\ 2 &\longrightarrow \sigma_4\ 1\ 2\\
\sigma_2\ 0\ 1 &\longrightarrow \sigma_3\\
\sigma_3\ \lambda\ 2 &\longrightarrow \beta\ \ \lambda\ 2\\
\sigma_4\ 0\ 1 &\longrightarrow \beta\ \ 0\ 2
\end{aligned}$$

3. $\alpha\ / \longrightarrow \beta +\quad \Rightarrow$

$$\begin{aligned}
\alpha\ \ 0\ 2 &\longrightarrow \sigma_4\ 0\ 1\\
\alpha\ \ 1\ 2 &\longrightarrow \beta\ \ 1\ 1\\
\alpha\ \ \lambda\ 2 &\longrightarrow \sigma_3\ \lambda\ 2\\
\sigma_4\ 0\ 2 &\longrightarrow \sigma_1\ 0\ 2\\
\sigma_4\ 1\ 2 &\longrightarrow \sigma_1\ 1\ 2\\
\sigma_3\ \ \ \ &\longrightarrow \sigma_2\ 0\ 1\\
\sigma_2\ \lambda\ 2 &\longrightarrow \sigma_1\ \lambda\ 2\\
\sigma_1\ 0\ 1 &\longrightarrow \beta\ \ 0\ 1
\end{aligned}$$

where σ_1 — σ_4 are states unique to the given Turing machine transition rule. Note that (3) is the reverse of (2). The hard work involved in moving left and right comes from the requirement to maintain a consistent and unique implicit representation of the infinite background of zeros on the right.

It is enough to prove two things: that the forward direction is deterministic, and that the forward direction correctly simulates the Turing machine transitions. Then no point can have multiple predecessors because simulating the reverse Turing machine transition crosses that point in the opposite direction. (The fact that the forward and backward stack machine paths within a single Turing machine transition must be the same follows from the fact that for the reverse path we could have just applied the forward rules in reverse order.)

Forward determinism follows because in any stack machine state we are reading at most one stack. It is also easy to verify that (2) and (3) correctly simulate the Turing machine transition in the forward direction by following all the branches. For that, note that (2) guarantees that the bottom symbol in stack 2, if any, is a 1. (I.e.: no unnecessary "blanks". We also assume this is true of the initial state.) Thus in (3), stack 2 cannot be empty when we get to state σ_4.

More efficient reversible Turing machines with multiple tapes and large alphabets can be simulated in a similar manner, as a straightforward generalization of the given construction. This is important because whereas 1-tape, 2-symbol reversible Turing machines are indeed universal, multi-tape Turing machines are essential for Bennett's theorems showing that the time and space requirements for logically reversible Turing machine computation are no more than slightly worse than linear with respect to irreversible Turing machine computation [5,7].

6 Conclusions

Our paper contributes to the art of designing molecular interactions using strand displacement cascades by proposing a direct implementation of arbitrary coupled reversible reactions, as well as a way to add and remove end monomers to and from a DNA polymer. The new construction for reversible reactions is more efficient than implementing them as two separate irreversible reactions [25], both in terms of the complexity of the scheme as well as the amounts of fuel reagents required. By adjusting and maintaining fuel concentrations, reactions can be biased forward or backward or balanced arbitrarily close to equilibrium. Based on these reaction mechanisms, we developed a novel method of embedding computation in biochemical and biological systems by showing an efficient autonomous stack machine simulation. This simulation can be made reversible to attain low energy consumption.

Different architectures for molecular computing such as algorithmic self-assembly, circuits implemented with CRNs, Turing machines implemented with CRNs, and polymer CRNs embody different tradeoffs between time, volume, energy and uniformity. Our construction is exponentially more efficient in terms of the required molecular counts and volume than geometry-free Turing-universal computation using strand displacement reactions (combination of refs. [25] and [24]), and also polynomially faster. Moreover, unlike the geometry-free computation of ref. [24], our polymer CRN construction in theory yields the correct computation output with probability 1.

Lastly, using our implementation of reversible CRNs, we proposed a logically-reversible stack machine construction that maintains error-free computation using physically reversible reactions. We showed that these reversible stack machines can reversibly simulate a reversible Turing machine, establishing their Turing universality and the applicability of results in the existing literature. Our constructions can be viewed as steps toward a DNA implementation of Bennett's thought experiment [6] in which computation was shown to require arbitrarily little thermodynamic energy per step.

There is still room for improvement in our constructions. First, the fact that the machine state is stored within multiple free-floating molecules results in the requirement for slow bimolecular reactions, unlike Bennett's hypothetical scheme. A second drawback of our scheme is that preparing reactions with a single copy of each state-bearing molecule would be difficult experimentally, but our system will not correctly simulate stack machines if run with multiple copies of state or stack molecules, or if run with mass action. Further, unlike Bennett's scheme, ours cannot run an arbitrary number of parallel machines in the same reaction chamber. This limitation prevents the use of our construction for fast solutions to parallel search problems [1].

Finally, our construction lacks the attractive feature of material recycling: taking any irreversible Turing machine, applying the transform described in ref. [5] to make it reversible, and implementing it with Bennett's hypothesized molecular construction, yields a molecular computation that recycles all material requirements except for the molecules used in writing out the output. However, in our

scheme, different fuel molecules would be used in the "compute" and "retrace" phases of the transformed Turing machine computation, and would not be regenerated. Indeed, every computation converts fuels of forward reactions to the fuels of reverse reactions in an amount proportional to the length of the computation.

The polymer reactions we introduce are likely instances of a wider class of polymer modification reactions that can be implemented with strand displacement. It would be exciting to implement a polymer reaction class capable of exhibiting the richness of cytoskeletal networks that are responsible for cellular reorganization and coordinated movement.

Acknowledgments

We thank Ho-Lin Chen for insightful discussions and suggestions. Our development of the history-free CRN scheme grew out of extensive discussions with Luca Cardelli. We thank Anne Condon for clarifying discussions. This work was supported by the Molecular Programming Project under NSF grant 0832824 and an NSF CIFellows Award to DS.

References

1. Adleman, L.: Molecular Computation of Solutions to Combinatorial Problems. Science 266(5187), 1021–1024 (1994)
2. Beaver, D.: A Universal Molecular Computer. In: Lipton, R., Baum, E. (eds.) DNA Based Computers, pp. 29–36. AMS, Providence (1996)
3. Benenson, Y., Paz-Elizur, T., Adar, R., Keinan, E., Livneh, Z., Shapiro, E.: Programmable and autonomous computing machine made of biomolecules. Nature 414(6862), 430–434 (2001)
4. Benenson, Y., Shapiro, E.: Molecular computing machines. In: Encyclopedia of Nanoscience and Nanotechnology, pp. 2043–2056 (2004)
5. Bennett, C.: Logical reversibility of computation. IBM Journal of Research and Development 17(6), 525–532 (1973)
6. Bennett, C.: The thermodynamics of computation – a review. International Journal of Theoretical Physics 21(12), 905–940 (1982)
7. Bennett, C.: Time/space trade-offs for reversible computation. SIAM Journal on Computing 18, 766–776 (1989)
8. Blinov, M., Faeder, J., Goldstein, B., Hlavacek, W.: BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. Bioinformatics 20(17), 3289–3291 (2004)
9. Cardelli, L.: Strand algebras for DNA computing. In: Deaton, R., Suyama, A. (eds.) DNA 15. LNCS, vol. 5877, pp. 12–24. Springer, Heidelberg (2009)
10. Cardelli, L.: Two-Domain DNA Strand Displacement. In: Developments in Computational Models (DCM), pp. 33–47 (2010)
11. Cardelli, L., Zavattaro, G.: On the computational power of biochemistry. In: Horimoto, K., Regensburger, G., Rosenkranz, M., Yoshida, H. (eds.) AB 2008. LNCS, vol. 5147, pp. 65–80. Springer, Heidelberg (2008)
12. Chen, H., De, A., Goel, A.: Towards Programmable Molecular Machines. In: Foundations of Nanoscience (FNANO) (2008)

13. Danos, V., Feret, J., Fontana, W., Harmer, R., Krivine, J.: Rule-based modelling of cellular signalling. In: Caires, L., Li, L. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 17–41. Springer, Heidelberg (2007)
14. Kurtz, S., Mahaney, S., Royer, J., Simon, J.: Biological computing. In: Complexity Theory Retrospective II, pp. 179–195 (1997)
15. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development 5(3), 183–191 (1961)
16. Liekens, A.M.L., Fernando, C.T.: Turing complete catalytic particle computers. In: Almeida e Costa, F., Rocha, L.M., Costa, E., Harvey, I., Coutinho, A. (eds.) ECAL 2007. LNCS (LNAI), vol. 4648, pp. 1202–1211. Springer, Heidelberg (2007)
17. Minsky, M.L.: Computation: finite and infinite machines. Prentice-Hall, Englewood Cliffs (1967)
18. Morita, K., Shirasaki, A., Gono, Y.: A 1-tape 2-symbol reversible Turing machine. The Transactions of the IEICE E 72(3), 223–228 (1989)
19. Qian, L., Winfree, E.: A simple DNA gate motif for synthesizing large-scale circuits. In: Goel, A., Simmel, F.C., Sosík, P. (eds.) DNA 14. LNCS, vol. 5347, pp. 70–89. Springer, Heidelberg (2009)
20. Rothemund, P.: A DNA and restriction enzyme implementation of Turing machines. In: Lipton, R., Baum, E. (eds.) DNA Based Computers. DIMACS, vol. 27, pp. 75–119. AMS, Providence (1996)
21. Rothemund, P., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA sierpinski triangles. PLoS Biology 2(12), e424 (2004)
22. Seelig, G., Soloveichik, D., Zhang, D.Y., Winfree, E.: Enzyme-free nucleic acid logic circuits. Science 314(5805), 1585–1588 (2006)
23. Smith, W.: DNA computers in vitro and vivo. In: Lipton, R., Baum, E. (eds.) DNA Based Computers. DIMACS, vol. 27, pp. 121–185. AMS, Providence (1996)
24. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Natural Computing 7(4), 615–633 (2008)
25. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proceedings of the National Academy of Science 107(12), 5393–5398 (2010)
26. Winfree, E.: On the computational power of DNA annealing and ligation. In: Lipton, R., Baum, E. (eds.) DNA Based Computers. DIMACS, vol. 27, pp. 199–221. AMS, Providence (1996)
27. Winfree, E.: Simulations of computing by self-assembly. Technical Report CS-TR:1998.22, Caltech (1998)
28. Yin, P., Turberfield, A., Sahu, S., Reif, J.: Design of an autonomous DNA nanomechanical device capable of universal computation and universal translational motion. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 10. LNCS, vol. 3384, pp. 426–444. Springer, Heidelberg (2005)
29. Zhang, D., Turberfield, A., Yurke, B., Winfree, E.: Engineering entropy-driven reactions and networks catalyzed by DNA. Science 318(5853), 1121–1125 (2007)
30. Zhang, D., Winfree, E.: Control of DNA strand displacement kinetics using toehold exchange. Journal of the American Chemical Society 131(47), 17303–17314 (2009)