Towards temporal logic computation using DNA strand displacement reactions

Matthew R. Lakin[1,2,3] and Darko Stefanovic[2,3]

[1] Department of Chemical & Biological Engineering, University of New Mexico, NM, USA
[2] Department of Computer Science, University of New Mexico, NM, USA
[3] Center for Biomedical Engineering, University of New Mexico, NM, USA

{mlakin,darko}@cs.unm.edu

Abstract. Time-varying signals are ubiquitous throughout science, and studying the high-level temporal structure of such processes is of significant practical importance. In this context, techniques from computer science such as temporal logic are a powerful tool. Temporal logic allows one to describe temporal properties of time-varying processes, e.g., the order in which particular events occur. In this paper, we show that DNA strand displacement reaction networks can be used to implement computations that check certain temporal relationships within time-varying input signals. A key aspect of this work is the development of DNA circuits that incorporate a primitive memory, so that their behavior is influenced not just by the current observed chemical environment, but also by environments observed in the past. We formalize our circuit designs in the DSD programming language and use simulation results to confirm that they function as intended. This work opens up the possibility of developing DNA circuits capable of long-term monitoring of processes such as cellular function, and points to possible designs of future DNA circuits that can decide more sophisticated temporal logics.

1 Introduction

Dynamic processes that produce time-varying signals are found throughout nature. In molecular biology, for example, changes in levels of protein expression over time are a cornerstone of cellular regulatory systems. In this context, a molecular computing system able to analyze both the current state of the protein expression levels as well as the "historical record" of previously observed protein expression levels would be able to make sophisticated decisions about the cell state by observing protein expression over an extended period of time.

A fundamental goal of research into molecular computing and synthetic biology is to *produce* time-varying signals, an early example being the "repressilator" oscillatory network produced by a ring of three mutually inhibiting transcription factors [1]. However, there has been relatively little work on using molecular computers or engineered bacteria to *analyze* time-varying signals. This is because published research on molecular circuit designs has focused in large part on analyzing the input signals present in the chemical environment *at a particular point in time*. Examples include previously published DNA circuits that implement digital logic circuits [2], analog neural networks [3],

and population protocols for approximate majority voting [4]. The most notable examples of synthetic biomolecular circuits designed for processing temporal signals are designs for DNA strand displacement circuits to carry out discrete-time signal processing tasks using a combination of "fast" and "slow" reactions [5], and prior experimental work on using recombinase enzymes to integrate expressed single-stranded DNA into the genomes of engineered bacteria, as a record of events experienced in the past [6]. More tangentially related to the current topic are studies of learning and adaptation in engineered biochemical circuits [7, 8] and abstract chemical reaction networks [9–11], including DNA strand displacement learning circuits [9, 12] designed using buffered strand displacement gates [13]. The concept of memory in DNA reaction networks has also been explored indirectly via a postulated DNA implementation of a "reservoir computing" system [14], as well as by proposals for chemical memories implemented using bistable switches [15] and delay lines [16].

In this paper we broaden the focus of research in molecular circuit design to produce systems that can analyze the current chemical environment not just in isolation, but rather in the context of previous states of the chemical environment observed by the system. We will present designs for DNA strand displacement circuits that can analyze the temporal structure of time-varying input signals modeled as a sequence of additions of input strands that are subsequently degraded. (This could be realized in an experimental system by using RNA inputs and RNAse-containing media [17].) The structures of our networks will be designed such that the reactions triggered by the additions of the input strands at different points in time activate strand displacement gates whose outputs act as a "memory", so that the state of the network effectively stores information about its past experience. By cascading multiple such gates together, we will design systems in which the cascade only executes to completion (and thus produces an output signal) if the input signals are presented in an order that satisfies the temporal relationships that are encoded in the network structure, and we will verify correct operation of our circuit designs using computational simulations of an ordinary differential equation (ODE) model of the circuit kinetics. This work therefore demonstrates a path toward molecular computing systems that can analyze non-trivial temporal properties of time-varying signals, with potential applications in the analysis of biochemical systems and in the diagnosis and treatment of disease.

The remainder of this paper is structured as follows. We introduce a basic logic of temporal relationships for sequential signals in Section 2 and present designs for DNA strand displacement circuits that test whether a sequential presentation of input signals satisfies a particular formula in Section 3. We present results from simulations of example circuits in Section 4 and conclude with a discussion in Section 5.

2 A logic of temporal relationships for sequential signals

In this section we present a logic for expressing simple temporal relationships within sequential input sequences. We begin by specifying the well-formed formulae φ of our logic, which are as follows:

$$\begin{aligned}\varphi ::= & A \sqsubset B \sqsubset \cdots \sqsubset Z \\ \mid & \ \varphi_1 \wedge \varphi_2 \\ \mid & \ \varphi_1 \vee \varphi_2\end{aligned}$$

The formula $A \sqsubset B$ should be read as "A before B", and its intended meaning is that an occurrence of input A is observed in the sequence of input signals before an occurrence of B is observed.

Let σ range over finite input sequences $[A_1 \cdots A_n]$. These finite input sequences will serve as models for our formulae. We now define satisfaction of a formula by an input sequence, written $\sigma \vDash \varphi$, by recursion on the structure of formulae, as follows:

$$\begin{aligned}\sigma \vDash A \sqsubset B \sqsubset \cdots \sqsubset Z \quad &\Longleftrightarrow \quad \exists \sigma_0, \sigma_1, \ldots, \sigma_n.\ \sigma = [\sigma_0 A \sigma_1 B \sigma_2 \cdots \sigma_{n-1} Z \sigma_n] \\ \sigma \vDash \varphi_1 \wedge \varphi_2 \quad &\Longleftrightarrow \quad \sigma \vDash \varphi_1 \wedge \sigma \vDash \varphi_2 \\ \sigma \vDash \varphi_1 \vee \varphi_2 \quad &\Longleftrightarrow \quad \sigma \vDash \varphi_1 \vee \sigma \vDash \varphi_2\end{aligned}$$

The semantics of conjunction and disjunction formulae are standard. A BEFORE formula $A \sqsubset B \sqsubset \cdots \sqsubset Z$ is satisfied by an input sequence σ if there exist subsequences $\sigma_0, \sigma_1, \ldots, \sigma_n$ such that the input sequence σ can be expressed as the concatenation $[\sigma_0 A \sigma_1 B \sigma_2 \cdots \sigma_{n-1} Z \sigma_n]$. In other words, we require that there exist occurrences of $\sigma_0, \sigma_1, \ldots, \sigma_n$ that appear in the input sequence in the correct order. Since we do not place any restrictions on the number of times a particular input may appear in the sequence, there may be multiple different decompositions of this form, but we do not distinguish this in the semantics.

For example, consider the formula $\varphi = (A \sqsubset B) \wedge (A \sqsubset C)$. The following both hold:

$$[ABC] \vDash (A \sqsubset B) \wedge (A \sqsubset C)$$

$$[ACB] \vDash (A \sqsubset B) \wedge (A \sqsubset C),$$

but, on the other hand,

$$[CAB] \nvDash (A \sqsubset B) \wedge (A \sqsubset C)$$

because A does not occur before C in the input sequence $[CAB]$.

In the following section, we will define a translation of these formulae into chemical reaction networks realized using DNA strand displacement reactions. Viewed through the prism of the definitions presented above, the DNA reaction networks that we define will each embody a formula φ, and we will challenge the network by a sequence of input additions that correspond to a particular input sequence σ. Then, the goal for our network will be to respond (by producing a "high" concentration of an output species) iff the input sequence satisfies the implemented formulae, i.e., iff $\sigma \vDash \varphi$ holds.

We note that, if all signals mentioned in the subformula are unique, we can define the $A \sqsubset B \sqsubset \cdots \sqsubset Z$ construct in terms of the two-input case, as follows:

$$A_1 \sqsubset A_2 \sqsubset \cdots \sqsubset A_n = \bigwedge_{i \in \{1,\ldots,n\}} \bigwedge_{j \in \{i+1,\ldots,n\}} (A_i \sqsubset A_j)$$

However, for the purposes of producing a DNA implementation it is simpler and far more compact to implement the extended version using a single gate cascade than it is

to add a large number of additional AND gates. In defining this expansion, we consider two formulae φ_1 and φ_2 to be equivalent iff they they are satisfied by the same set of input sequences, i.e., iff $\{\sigma \mid \sigma \vDash \varphi_1\} = \{\sigma \mid \sigma \vDash \varphi_2\}$.

3 DNA circuits for analyzing temporal relationships

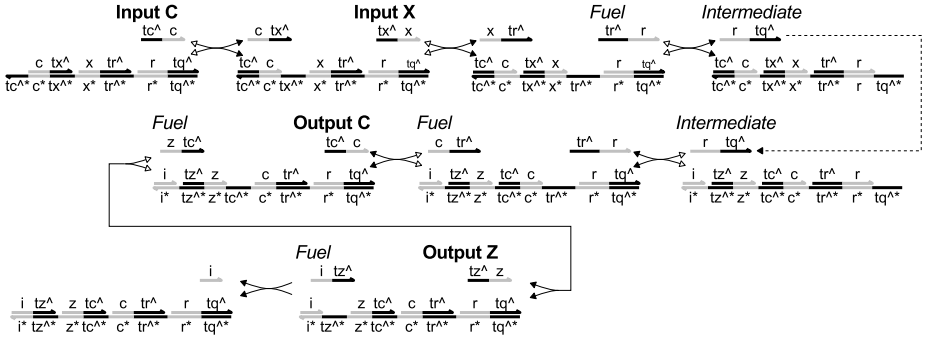
In this section we present our designs for DNA circuits that carry out temporal analysis tasks. Our chemical framework of choice is DNA strand displacement [18]. Strand displacement is a scheme for implementing reaction networks in DNA in which "signals" are represented by single strands of DNA in solution that interact with structures known as "gates" that consume certain input strands from solution and release output strands, with different sequences, back into solution. These interactions take place via a two-step process: the incoming strand first binds reversibly to the gate via a short complementary domain known as a "toehold", which positions the incoming strand to initiate the process of "branch migration", whereby it competes with the neighboring incumbent strand to bind to the gate. When the branch migration process completes, the end result is that the input strand is bound to the gate and the incumbent strand is released into solution. By designing structures so that multiple strand displacement reactions proceed in a pre-defined sequence, possibly with the assistance of other "fuel" molecules in solution, strand displacement gates can implement a range of computational tasks. Here we focus in particular on two-domain DNA strand displacement [19], a simplified form of strand displacement that has proven itself amenable to experimental implementation [4].

We will model our systems using the DSD programming language, which provides a text-based syntax for representing strand displacement gate structures and processes that represent the combination of multiple different gates and strands in a dilute, well-mixed solution. The semantics of the DSD language which specifies a formally-defined translation of those structural models into a kinetic model, by enumerating all possible interactions between the DNA components that could possibly occur within the system. For reasons of brevity, we do not provide a full exposition of the DSD language here, rather, we refer the reader to previous work that formally defines the syntax and semantics of the DSD language [13].

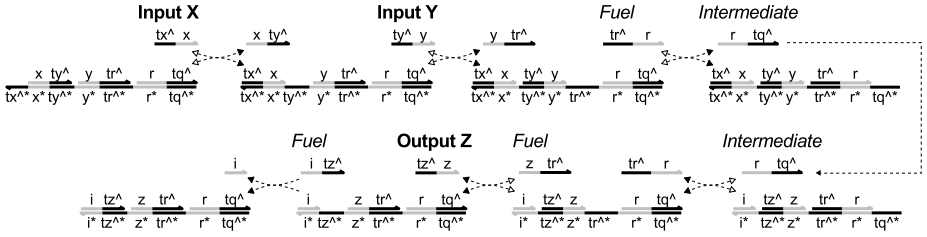
In the DSD syntax, each two-domain signal A in our circuits will be represented by a DSD module `Signal`(A) that just consists of a single two-domain DNA strand $<ta\hat{}\ a>$. Furthermore, we will model the input signals that appear in temporal formulae as degrading over time (with standard exponential decay kinetics) when they are free in solution. This approximates a real-world temporal analysis scenario where the DNA circuit is monitoring the occurrence of environmental markers that may also be consumed by other downstream chemical reactions that are taking place simultaneously.

We will implement our DNA reaction networks using three different kinds of strand displacement gates: "catalyst" reaction gates that implement abstract reactions of the form $C+X \rightarrow C+Z$, "AND" logic gates that compute the logical conjunction of two inputs, and "OR" logic gates that compute the logical disjunction of two inputs. Reac-

(a) `CatalystGate`(C,X,Z) implements $C+X \rightarrow C+Z$



(b) `AndGate`(X,Y,Z) implements $Z = X \wedge Y$



(c) `OrGate`(X,Y,Z) implements $Z = X \vee Y$

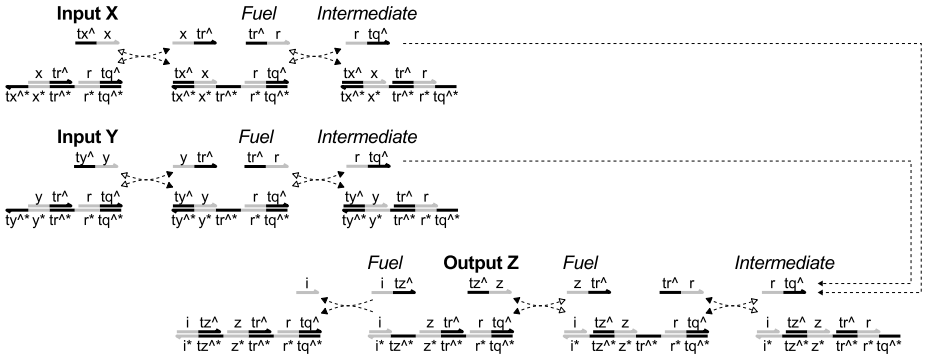


Fig. 1. Strand displacement reactions that implement (a) the abstract catalytic reaction $C+X \rightarrow C+Z$, (b) the "AND" logic gate $Z = X \wedge Y$, and (c) the "OR" logic gate $Z = X \vee Y$.

$$
\begin{array}{lll}
[\![A \sqsubset B \sqsubset \cdots \sqsubset Y]\!] = (P,Z) & \text{where } P = \texttt{Signal}(X_1) & \\
& \quad \mid \texttt{CatalystGate}(A,X_1,X_2) & \#\, A + X_1 \to A + X_2 \\
& \quad \mid \texttt{CatalystGate}(B,X_2,X_3) & \#\, B + X_2 \to B + X_3 \\
& \quad \mid \cdots & \#\, \cdots \\
& \quad \mid \texttt{CatalystGate}(Y,X_n,Z) & \#\, Y + X_n \to Y + Z \\
& \text{and } X_1, X_2, \ldots, X_n, Z \text{ are fresh species} & \\
[\![\varphi_1 \wedge \varphi_2]\!] = (P,Z) & \text{where } P = P_1 \mid P_2 \mid \texttt{AndGate}(X_1,X_2,Z) & \#\, Z = X_1 \wedge X_2 \\
& \text{and } [\![\varphi_1]\!] = (P_1,X_1) & \\
& \text{and } [\![\varphi_2]\!] = (P_2,X_2) & \\
& \text{and } Z \text{ is a fresh species} & \\
[\![\varphi_1 \vee \varphi_2]\!] = (P,Z) & \text{where } P = P_1 \mid P_2 \mid \texttt{OrGate}(X_1,X_2,Z) & \#\, Z = X_1 \vee X_2 \\
& \text{and } [\![\varphi_1]\!] = (P_1,X_1) & \\
& \text{and } [\![\varphi_2]\!] = (P_2,X_2) & \\
& \text{and } Z \text{ is a fresh species} &
\end{array}
$$

Fig. 2. Definition of the translation $[\![\varphi]\!]$ of formulae φ into a DSD process P and an output species Z. The "comments" on the right-hand side provide informal descriptions of the meaning of the DSD modules, for clarity. We note that, in the first case of the translation, the execution time is proportional to the number of compared signals.

tion schemes for each of these gate types are presented in Figure 1.⋆ The basic pattern is that the input strands bind to an input-accepting gate in a pre-programmed order, and with the help of a fuel strand enable the release of an intermediate strand that initiates a similar cascade of reactions on an output-releasing gate, which require additional fuel strands to be present and which release the output strands from the gate into solution. The function implemented by each gate is dependent on the patterns of input and output signals, so, for example, the "AND" gate has two input strands that must both be consumed in order to enable the release of a single output strand.

We can now define translation of the language of formulae from Section 2 into DNA strand displacement systems. For a formula φ, the translation $[\![\varphi]\!]$ returns a DSD process P (which is just a collection of parallel DSD species) and an output species Z. The output species is the one whose concentration will indicate the output of the computation: if it goes high then the input signal sequence *satisfies* the formula encoded in the network, and if it stays low then the input signal sequence *does not satisfy* the formula encoded in the network.

The definition of the translation is presented in Figure 2. The key case is the one for the formulae with actual temporal meaning, that is, the formulae of the form $A \sqsubset B \sqsubset \cdots \sqsubset Y$. Temporal formulae such as this are encoded using a cascade of strand displacement catalyst gates, catalyzed by the input signals $A, B, \ldots, Y$. These reactions catalyze conversion of a "substrate" species X_1 to X_2, then to X_3, and so on, until the final catalyst gate produces the overall output species Z. (The DSD process produced by this case of the translation also includes the initial substrate species X_1.) Crucially,

⋆ See the Supporting Information (available from the first author's web page) for full DSD code listings for each system simulated in this paper, including full definitions of the modules.

the input signals $A, B, \ldots, Y$ catalyze this cascade of reactions *in the same order as they appear in the temporal formula*, ordered from earliest to latest. This means that, if the input signals are actually observed in this order, then these catalyst reactions will all be activated in turn, leading to the eventual release of the output species. However, if one or more of the input signals is never observed, or is observed out of the required sequence, then one or more of the catalyst reactions will not be activated, and thus the output sequence will not be produced at the end of the cascade. Hence, presence or absence of the output species corresponds to whether the input signals were observed in the correct temporal ordering, and hence to satisfaction of the temporal formula. The key to our circuit design is that the conversion of the substrate species, catalyzed by the input signals, serves as a "memory" that records the past experience of the networks interactions with the observed input species. This ensures that each input signal will be (almost) entirely removed from the system before the next input signal is presented, which prevents unwanted circuit responses being generated by overlapping input signals.

The remaining two cases of the definition of the translation, for "AND" and "OR" formulae, are comparatively straightforward. In each of these cases, the processes and output species for the two subformulae are defined recursively, and these processes are then returned in parallel with a new logic gate whose input species are the output species from the translations of the two subformulae and whose output species is a freshly generated signal.

4 Simulation results

DNA strand displacement reaction networks that carry out temporal analysis tasks were compiled and simulated using Visual DSD [20], using the "Infinite" semantics. In particular, we used the "beta" version of DSD [21] that supports scheduled additions of inputs via mixing events as well as the inclusion of user-defined reactions.

Briefly, the simulation conditions were as follows: we use a 1000 nM initial concentration of strand displacement gates and fuel strands, with the exception of the output part of the "OR" gates, of which we use 10 nM (so that the output signal strength of the "OR" gate is the same whether one or two positive input signals is present). The input signal sequence was implemented by adding a 10 nM concentration of each input signal in a pre-programmed order, with a wait time between addition of input signals of 30 000 s. The degradation rate of those input signals is $0.0005\,\mathrm{s}^{-1}$, that is, for each input signal A a unimolecular degradation reaction $A \rightarrow \varnothing$ with rate constant $k = 0.0005\,\mathrm{s}^{-1}$ was explicitly added to the model. We kept this rate constant between simulations for consistency, however, our circuits could be adapted to different degradation rates by modifying the rates of the other DNA reactions, e.g., by lengthening toehold domains or by increasing fuel concentrations. Additionally, for each input-consuming gate, a 10 nM concentration of the strand that is displaced by the binding of the first input was also included, to provide a degree of "backpressure" that presents inputs from being sequestered by binding to the gates, which allows them to be released back into solu-

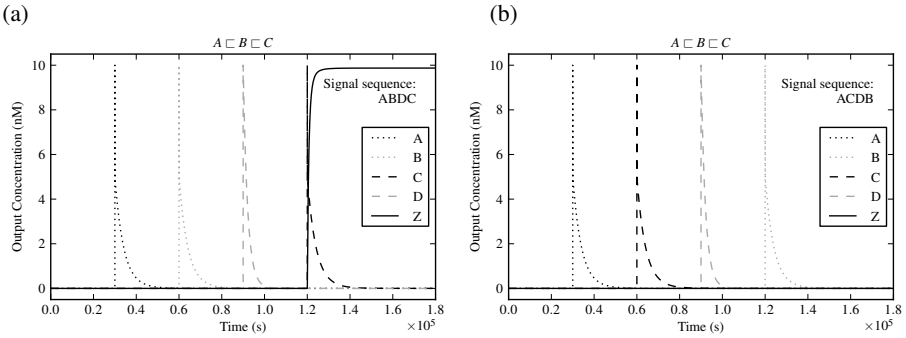


Fig. 3. Concentration time courses of selected species for the formula $A \sqsubset B \sqsubset C$, for input signal sequences (a) $[ABDC]$ and (b) $[ACDB]$.

tion so that they can be degraded. This is crucial to prevent unwanted circuit responses triggered by input signals left over from earlier stages of the simulation.**

Figure 3 shows time courses of the concentrations of the input signals (A, B, C, and D) and the output species (Z) for two different input signal sequences, $[ABDC]$ and $[ACDB]$, when added to a network that tests satisfaction of the formula $A \sqsubset B \sqsubset C$. Hence, we expect that the response (i.e., the final concentration of the output species Z) should be high for the input sequence $[ABDC]$ (since A appears before B and B appears before C in $[ABDC]$) but should be low for the input sequence $[ACDB]$ (since B does not appear before C in $[ACDB]$). Indeed, the plots from Figure 3 confirm this, as the final concentration of Z is high in part (a) but low in part (b). Thus, in this case the circuit construction for testing satisfaction of temporal formulae functioned as intended.

We further investigated the correctness of our circuit designs for all possible permutations of the input signals A, B, C, and D, for three different formulae that collectively employ all three kinds of formula: $A \sqsubset B \sqsubset C$, $(A \sqsubset B) \wedge (C \sqsubset D)$, and $(A \sqsubset B) \vee (C \sqsubset D)$. The final concentrations of the output species in each case are presented in Figure 4(a)–(c), where the black bars are those where the output of the circuit is expected to be high, and the light grey bars are those where the output of the circuit is expected to be low. For clarity, the labels of those bars were typeset in boldface and italics, respectively. These results show that, in all cases, the circuit designs were able to correctly compute whether the corresponding formula was satisfied by the particular sequence of input signals, with a high ratio of signal to leakage (unwanted circuit activation). This indicates that our compilation from formulae into DNA circuits is functioning correctly.

Since our definitions do not require that the inputs in the input sequence are unique, we also used our circuits to test satisfaction of the temporal formula $A \sqsubset B \sqsubset A \sqsubset B$, which is satisfied by any input sequence in which A appears followed by B, followed again by A and then B. Figure 4(d) shows the final concentration of the output species from this circuit, when tested with all possible input signal sequences that contain two

** See the Supporting Information (available from the first author's web page) for full DSD code listings for each system simulated in this paper, including full definitions of the modules.

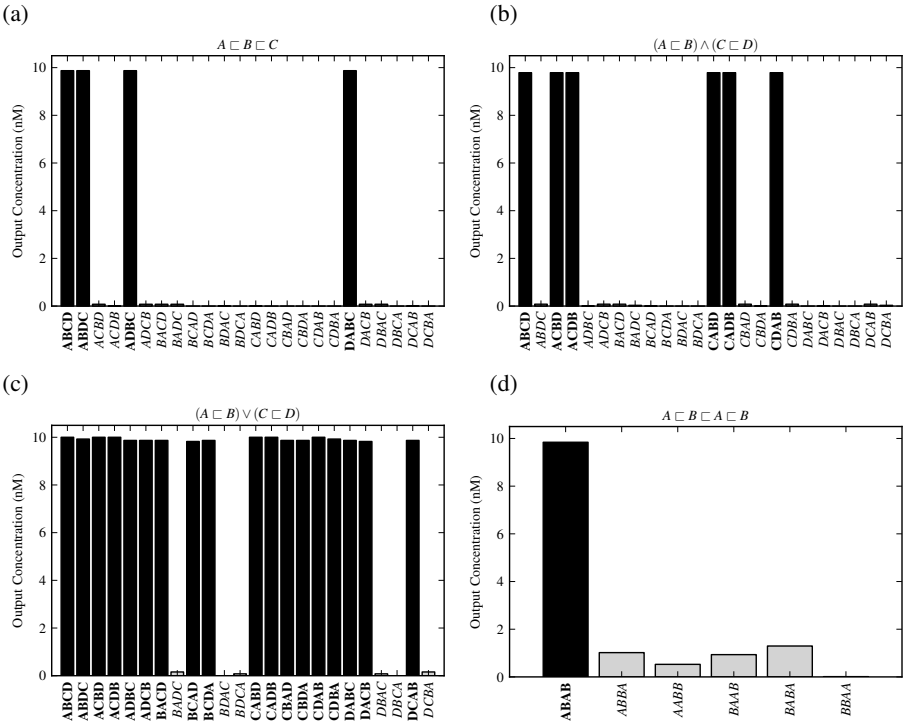


Fig. 4. Final concentrations of the output species for formulae (a) $A \sqsubset B \sqsubset C$, (b) $(A \sqsubset B) \wedge (C \sqsubset D)$, (c) $(A \sqsubset B) \vee (C \sqsubset D)$, and (d) $A \sqsubset B \sqsubset A \sqsubset B$. In parts (a)–(c), the bars represent the output for all possible permutations of the input signals A, B, C, and D, and in part (d) the bars represent the output for all possible permutations of the input signals A, A, B, and B. Black bars (with boldface labels) represent those simulations where the formula is satisfied by the corresponding input signal sequence, and light grey bars (with italic labels) represent those simulations where it is *not* satisfied.

occurrences of A and two occurrences of B. As the figure shows, the circuit only returned a high response for the input sequence $[ABAB]$, as expected. Thus, our DNA circuits could be used as a crude means of detecting switching, or oscillatory, input signals.

Finally, we tested a larger example formula $(((A \sqsubset B) \vee (A \sqsubset C)) \wedge (C \sqsubset D \sqsubset E))$ that includes all three formula types in a single circuit, with a total of five input signals (A, B, C, D, and E). This gave a total of 120 distinct input sequences, and the final concentration of the output species for each of these is presented in Figure 5. Again, we see that the circuit responded correctly, with a high output concentration whenever the input signal sequence satisfied the encoded formula, and a low output concentration whenever it did not. This demonstrates that our approach can be scaled to larger circuits that implement larger formulae. This scalability is further demonstrated by Table 1, which presents the circuit sizes for all five example circuits from Figures 4 and 5. The

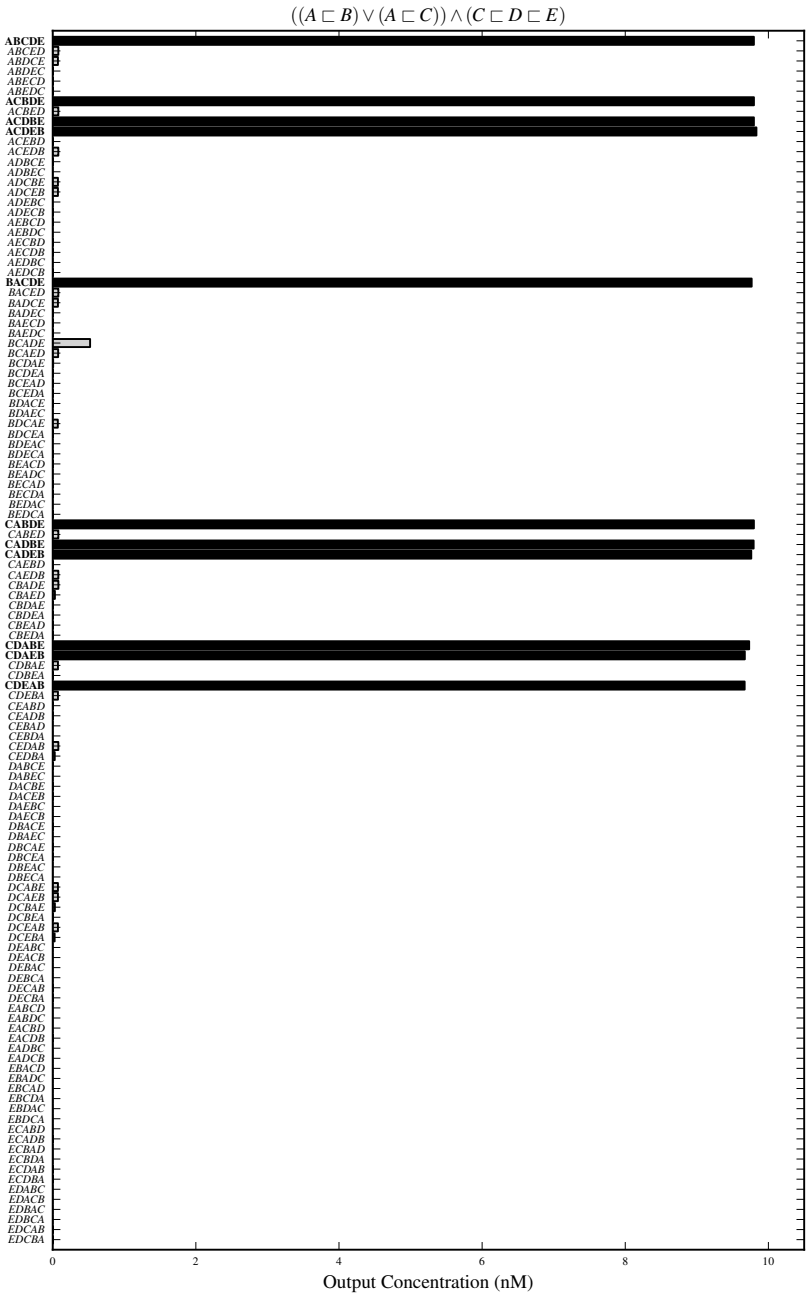


Fig. 5. Final concentrations of the output species for the formula $((A \sqsubset B) \vee (A \sqsubset C)) \wedge (C \sqsubset D \sqsubset E)$. The bars represent the output for all possible permutations of the input signals A, B, C, D, and E. Formatting of bars and labels is as explained in Figure 4.

Table 1. Circuit sizes for circuits simulated in Figures 4 and 5, expressed in terms of the number of species. The number of species in each case was calculated by considering all logic gates and fuel strands that must be present initially, as well as all signals that are either present initially or introduced during the course of the experiment.

Formula	Signals	Gate structures	Fuel strands	**Total**
$A \sqsubset B \sqsubset C$	5	6	15	**26**
$(A \sqsubset B) \wedge (C \sqsubset D)$	6	10	24	**40**
$(A \sqsubset B) \vee (C \sqsubset D)$	6	11	25	**42**
$A \sqsubset B \sqsubset A \sqsubset B$	3	8	20	**31**
$((A \sqsubset B) \vee (A \sqsubset C)) \wedge (C \sqsubset D \sqsubset E)$	8	19	44	**71**

largest of these circuits, the one from Figure 5, has a number of initial species roughly comparable to the largest strand displacement system implemented experimentally [2], which contained 74 initial DNA species, excluding inputs.

5 Discussion

To summarize, we have shown that our simple logic of temporal relationships, which allows properties concerning the relative temporal occurrence of signals in a linear input sequence to be expressed, can be compiled systematically into DNA strand displacement reaction networks such that the output networks encode the semantics of the corresponding formulae. Then, when those networks are presented with a linear temporal sequence of input signals, each network produces a high concentration of its output species if the input signal sequence *satisfies* the encoded formula, and produces little or no output species if the input signal sequence *does not satisfy* the encoded formula. For simplicity, in our simulations we assumed that those input signals undergo exponential decay when free in solution. Our simulation results indicate that our circuit designs and compilation process function correctly for a range of input signals and different temporal formulae, and that our design approach can be scaled to larger formulae (subject to the usual limitations imposed by the DNA sequence space).

In practice, degrading inputs could be implemented by using RNA input strands (and RNA outputs from the catalyst gates) with nuclease-containing media [17] so that single-stranded RNA in solution is degraded. An alternative approach could be to include additional DNA circuit components that act as a sink for the input strands. We used two-domain strand displacement catalyst gates as the basis for our designs because of their highly modular and composable nature. However, an alternative framework could be the strand displacement catalyst system developed by Zhang et al [22]. A practical advantage of this system is that it would require fewer strands for an experimental implementation. Furthermore, that catalyst design actually recycles the original input strand back into solution, as opposed to our design based on two-domain strand displacement in which a distinct copy of the input strand is released back into solution. Thus, this approach might be more easily integrated with the RNA-based approach to implementing degradable inputs, as discussed above.

Another alternative circuit design might employ fork gates instead of catalyst gates, which would mean that the input signals, once used by a gate to modify the populations of substrate species, would not be released back into solution at all. This could be a simpler solution for the purposes of building an experimental system but would mean that the circuit would have a significant impact on the system that it was measuring—a key rationale for using catalytic reactions is that the recycled input strands could continue to undergo reactions elsewhere in the system, thereby allowing our circuits to be used for real-time monitoring of biochemical systems, such as cellular regulatory networks, without significantly perturbing the system under observation. An additional advantage of using simpler strand displacement gates to implement temporal sensing is that most designs for multi-input strand displacement logic gates impose an implicit ordering on the binding of multiple inputs to the logic gate [23], thereby providing another alternative mechanism for the experimental implementation of temporal sensing.

The logic that we defined in Section 2 does not include negation, which is in keeping with previous work that used dual rail expansions to eliminate negation from DNA logic circuits [2]. In our logic, however, such expansions are more challenging. It is tempting to think that we can achieve a similar effect by using de Morgan's laws to push negations through conjunctions and disjunctions, and by expanding BEFORE formulae when the negations reach them, e.g.:

$$\neg(A_1 \sqsubset A_2 \sqsubset \cdots \sqsubset A_n) = \bigvee_{i\in\{1,\ldots,n\}} \bigvee_{j\in\{i+1,\ldots,n\}} (A_j \sqsubset A_i)$$
$$\neg(\varphi_1 \wedge \varphi_2) = (\neg\varphi_1) \vee (\neg\varphi_2)$$
$$\neg(\varphi_1 \vee \varphi_2) = (\neg\varphi_1) \wedge (\neg\varphi_2)$$

However, even for simple examples such as $\neg(A \sqsubset B)$, which expands to $B \sqsubset A$, this expansion misbehaves when one or both of the mentioned input signals are absent, e.g., $[AC] \not\models A \sqsubset B$ and $[AC] \not\models B \sqsubset A$. More disturbingly, $\neg(A \sqsubset A)$ expands to $A \sqsubset A$. Clearly more work on the semantics of negation in logics such as this is required. Indeed, one can think of our logic as a quantifier-free, negation-free subset of first order logic where the temporal ordering between signals could be implemented as a ternary predicate over a signal sequence and the two signals in question, and in this view the implementation of negation is less problematic but would still require our DNA circuits to be able to detect the *absence* of a particular signal from the input sequence.

To simplify the presentation, in this paper we assumed that the times in which the different inputs are present in the system do not overlap. However, in practical applications the signals that we might want to analyze are unlikely to be so clear-cut and regimented. An obvious first step would be to relax the requirement that all inputs are non-overlapping, so there could be two or more input signals present in solution simultaneously. In this case, the $A \sqsubset B$ formula could be generalized to an $A \sqsubseteq B$ formula, which would be satisfied if A occurs before, or at the same time as, B, and the implementation would need to be generalized accordingly, e.g., by using cooperative hybridization to detect the simultaneous presence of input signals [24].

Another important generalization would be to handle input concentration profiles that change continuously over time, rather than being added at discrete points as in this paper. In this context, our circuits would likely need to discretize the incoming signals in

terms of their concentration, as well as in time. For the former, prior work on digital and analog DNA circuits implemented using "seesaw gates" [2, 3] employed a thresholding mechanism, which could be used to discretize concentrations of the signals in the input time course. To discretize signals in time, Jiang et al. [5] use both synchronous (via an oscillatory chemical "clock") and asynchronous (self-timed) approaches.

Finally, the logic that we implemented in this paper is relatively straightforward, as it just allows statements about the order in which different input signals were observed in the linear input sequence. However, there are many more temporal logics in practical and industrial use, such as computation tree logic (CTL), linear temporal logic (LTL), and interval temporal logic (ITL). These discrete-time logics deal with branching time, infinite linear time, and finite linear time, respectively, and include more powerful logical primitives such as checking whether a proposition is globally true, or eventually true, or true until some other proposition becomes true. In the case of CTL, there are also logical primitives to deal with whether these properties hold along all branching paths in time, or just some. Clearly, these are much more powerful logics than that which we defined in this paper. A fruitful direction for future research would be to investigate designs for DNA-based circuits that can decide satisfaction of these more powerful logics, or to recognize strings drawn from regular languages (in which case, the DNA network would encode a regular expression). These recognition tasks are non-trivial because solving them would require far more information about the past states observed by the network to be stored, such as the length of time that has passed since a given signal was observed. Tackling this problem in an efficient and scalable manner would require us to be able to use the same input signal irrespective of its position in the temporal ordering, and we note that previous work on chemical memories [15, 16] provides a possible solution to these challenges associated with reusing inputs and deciding more sophisticated temporal logics.

Acknowledgments. This material is based upon work supported by the National Science Foundation under grants 1525553, 1518861, and 1318833.

References

1. M. B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338, 2000.
2. L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332:1196–1201, 2011.
3. L. Qian, E. Winfree, and J. Bruck. Neural network computation with DNA strand displacement cascades. *Nature*, 475:368–372, 2011.
4. Y.-J. Chen, N. Dalchau, N. Srinivas, A. Phillips, L. Cardelli, D. Soloveichik, and G. Seelig. Programmable chemical controllers made from DNA. *Nature Nanotechnology*, 8:755–762, 2013.
5. H. Jiang, S. A. Salehi, M. D. Riedel, and K. K. Parhi. Discrete-time signal processing with DNA. *ACS Synthetic Biology*, 2(5):245–254, 2013.
6. F. Farzadfard and T. K. Lu. Genomically encoded analog memory with precise in vivo DNA writing in living cell populations. *Science*, 346(6211):1256272, 2014.

7. C. T. Fernando, A. M. K. Liekens, L. E. H. Bingle, C. Beck, T. Lenser, D. J. Stekel, and J. E. Rowe. Molecular circuits for associative learning in single-celled organisms. *Journal of the Royal Society Interface*, 6:463–469, 2009.
8. S. McGregor, V. Vases, P. Husbands, and C. Fernando. Evolution of associative learning in chemical networks. *PLoS Computational Biology*, 8(11):e1002739, 2012.
9. M. R. Lakin, A. Minnich, T. Lane, and D. Stefanovic. Design of a biochemical circuit motif for learning linear functions. *Journal of the Royal Society Interface*, 11(101):20140902, 2014.
10. P. Banda, C. Teuscher, and M. R. Lakin. Online learning in a chemical perceptron. *Artificial Life*, 19(2):195–219, 2013.
11. P. Banda, C. Teuscher, and D. Stefanovic. Training an asymmetric signal perceptron through reinforcement in an artificial chemistry. *Journal of the Royal Society Interface*, 11:20131100, 2014.
12. M. R. Lakin and D. Stefanovic. Supervised learning in adaptive DNA strand displacement networks. *ACS Synthetic Biology*, 5(8):885–897, 2016.
13. M. R. Lakin, S. Youssef, L. Cardelli, and A. Phillips. Abstractions for DNA circuit design. *Journal of the Royal Society Interface*, 9(68):470–486, 2012.
14. A. Goudarzi, M. R. Lakin, and D. Stefanovic. DNA reservoir computing: a novel molecular computing approach. In D. Soloveichik and B. Yurke, editors, *Proceedings of DNA19*, volume 8141 of *LNCS*, pages 76–89. Springer-Verlag, 2013.
15. A. Padirac, T. Fujii, and Y. Rondelez. Bottom-up construction of in vitro switchable memories. *Proceedings of the National Academy of Sciences USA*, 109(47):E3212–E3220, 2012.
16. J. Moles, P. Banda, and C. Teuscher. Delay line as a chemical reaction network. *Parallel Processing Letters*, 21(1), 2015.
17. M. R. O'Steen, E. M. Cornett, and D. M. Kolpashchikov. Nuclease-containing media for resettable operation of DNA logic gates. *Chemical Communications*, 51:1429–1431, 2015.
18. D. Y. Zhang and G. Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature Chemistry*, 3(2):103–113, 2011.
19. L. Cardelli. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science*, 23:247–271, 2013.
20. M. R. Lakin, S. Youssef, F. Polo, S. Emmott, and A. Phillips. Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics*, 27(22):3211–3213, 2011.
21. B. Yordanov, J. Kim, R. L. Petersen, A. Shudy, V. V. Kulkarni, and A. Phillips. Computational design of nucleic acid feedback control circuits. *ACS Synthetic Biology*, 3(8):600–616, 2014.
22. D. Y. Zhang, A. J. Turberfield, B. Yurke, and E. Winfree. Engineering entropy-driven reactions and networks catalyzed by DNA. *Science*, 318:1121–1125, 2007.
23. D. Soloveichik, G. Seelig, and E. Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences USA*, 107(12):5393–5398, 2010.
24. D. Y. Zhang. Cooperative hybridization of oligonucleotides. *Journal of the American Chemical Society*, 133:1077–1086, 2011.