Functional Analysis of Large-scale DNA Strand Displacement Circuits

Boyan Yordanov, Christoph M. Wintersteiger, Youssef Hamadi,
Andrew Phillips, and Hillel Kugler

Microsoft Research, Cambridge UK
{yordanov,cwinter,youssefh,aphillip,hkugler}@microsoft.com
http://research.microsoft.com/z3-4biology

Abstract. We present a method for the analysis of functional properties of large-scale DNA strand displacement (DSD) circuits based on Satisfiability Modulo Theories that enables us to prove the functional correctness of DNA circuit designs for arbitrary inputs, and provides significantly improved scalability and expressivity over existing methods. We implement this method as an extension to the Visual DSD tool, and use it to formalize the behavior of a 4-bit square root circuit, together with the components used for its construction. We show that our method successfully verifies that certain designs function as required and identifies erroneous computations in others, even when millions of copies of a circuit are interacting with each other in parallel. Our method is also applicable in the verification of properties for more general chemical reaction networks.

1 Introduction

The engineering of nanoscale devices from DNA has emerged as a powerful technology, with potential applications in nanomedicine and nanomaterials. More recently, DNA strand displacement (DSD) has attracted attention as a promising approach for engineering molecular devices with complex dynamics [24], and has been shown to scale to large circuits [17]. In spite of this potential, many challenges remain before the design of DSD circuits with predictable, robust behavior becomes routine. In addition to the experimental difficulties of synthesis, assembly, and elimination of cross-talk, the massive parallelism and complexity of DSD circuits make their manual design challenging and error-prone.

A number of computational methods and tools have been developed to facilitate the design process. In particular, the Visual DSD tool [14] computes the set of all possible strand displacement reactions generated from an initial collection of DNA species, and simulates these reactions over time. Methods have also been developed for proving that a set of strand displacement reactions is equivalent to a reduced set of reactions [18, 7]. However, further work is needed to be able to state and prove properties about the function that these reactions perform. To help address this, methods based on probabilistic model checking have been developed to prove properties about the states that a strand displacement circuit

traverses, together with the expected time and probability of failure [13]. So far however, these methods do not scale to realistic numbers of molecules. To help improve scalability, a symbolic method called Z34Bio for analyzing large and potentially infinite state spaces based on Satisfiability Modulo Theories (SMT) was developed [21, 22]. This technique has been applied to study structural properties of DNA circuits, such as the presence of exposed DNA sequences.

In this paper we present a method that allows the desired properties of a DNA strand displacement circuit to be formalized as a high-level functional specification, and formally verified for realistic numbers of molecules. The method extends the use of SMT-solvers for analyzing chemical reaction networks presented in [21], and is implemented within the Visual DSD tool using the Z34Bio framework, which is based on the Z3 theorem prover and SMT-solver [6]. To illustrate this approach, we study a model of the 4-bit square root circuit described in [2], which was originally developed as a localized circuit in contrast to the design from [17]. We formalize and analyze functional properties of the individual components used to construct this system, and show that a modified version of this design functions correctly, even when millions of copies interact with each other in parallel. We illustrate how our method helps identify design errors at both the component and circuit level. Although the method has been tailored specifically for DNA strand displacement systems, it is also more generally applicable for the analysis of chemical reaction networks.

2 SMT Analysis of Chemical Reaction Networks

This section summarizes the SMT-based method for analyzing Chemical Reaction Networks (CRNs) presented in [21], which will be used in the remainder of the paper. We denote a finite set as $S = \{s_0, \ldots, s_N\}$, where $|S| = N + 1$ is the number of elements in S. We use $S = \{(s_0, n_0), \ldots, (s_N, n_N)\}$ to denote a finite multiset where each pair (s_i, n_i) denotes an element s_i and its multiplicity n_i, with $n_i > 0$. Given a multiset S we use $s \in S$ for $\exists n \,.\, (s, n) \in S$ and $S(s) = n$ when $(s, n) \in S$ and $S(s) = 0$ otherwise. We define a CRN as a pair $(\mathcal{S}, \mathcal{R})$, where $\mathcal{S}$ is a finite set of species and $\mathcal{R}$ is a finite set of possible reactions. A reaction $r \in \mathcal{R}$ is defined as a pair of multisets $r = (R_r, P_r)$ denoting the reactants and products of r, respectively. For $(s, n) \in R_r$ (respectively P_r), $s \in \mathcal{S}$ is a species and n is the stoichiometry indicating how many molecules of s are consumed (respectively produced) when reaction r takes place.

To study the dynamics of a CRN with single-molecule resolution, we formalize its behavior as the transition system $\mathcal{T} = (Q, q_0, T)$, where Q is the set of states, $q_0 \in Q$ is the uniquely defined initial state, and $T \subseteq Q \times Q$ is the transition relation. Each state $q \in Q$ is a multiset of species and $q(s)$ indicates how many molecules of s are present in state q. A reaction r is *enabled* in q if there are enough molecules of each of its reactants for it to trigger; *i.e.,* $enabled(r, q) \leftrightarrow \bigwedge_{s \in \mathcal{S}} q(s) \geq R_r(s)$. A state q is *terminal* if it has no enabled reactions *i.e.*

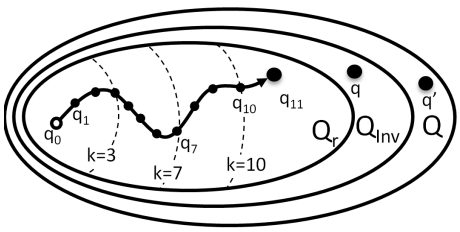

Fig. 1. SMT-based Analysis. All trajectories explored by Bounded Model Checking represent valid computations from q_0 but Q_r^k (*e.g.* $k = 3, 7, 10$) generally under-approximates the reachable states Q_r. The over-approximation Q_{Inv} allows proving that state q' is unreachable but spurious states (*e.g.* q) may be included.

$terminal(q) \leftrightarrow \bigwedge_{r \in \mathcal{R}} \neg enabled(r, q)$. The transition relation T is defined as

$$T(q, q') \leftrightarrow \bigvee_{r \in \mathcal{R}} (enabled(r, q) \wedge \bigwedge_{s \in S} q'(s) = q(s) - R_r(s) + P_r(s)). \quad (1)$$

Once a CRN $(\mathcal{S}, \mathcal{R})$ is encoded as a transition system $\mathcal{T} = (Q, q_0, T)$ with some initial state $q_0 \in Q$, a number of analysis questions are expressible as logical formulas and resolvable using model checking methods and SMT solvers such as Z3 [6]. In the following, we focus on safety properties [16] *i.e.* a given state predicate $P : Q \to \mathbb{B}$ holds for all reachable states. Let $Q_r \subseteq Q$ denote all reachable states of $\mathcal{T}$ and let $Q_r^k \subseteq Q$ denote the set of states reachable in up to k transitions from q_0 (note that $Q_r^k \subseteq Q_r^{k+1} \subseteq \cdots \subseteq Q_r$).

We check the reachability of a state $q \in Q$ such that $\neg P(q)$ holds by using an SMT solver to decide whether the formula $\exists q_1, \ldots, q_k \;.\; \bigwedge_{i=0}^{k-1} T(q_i, q_{i+1}) \wedge \bigvee_{i=0}^{k} \neg P(q_i)$ is satisfiable. The formula represents the *unrolling* of the transition relation for k transitions from q_0 (see Fig. 1 for an example). This type of approach is usually called Bounded Model Checking (BMC) [1] and is particularly good at finding bugs in hardware and software, with the added advantage of producing an explicit computation trace which demonstrates the behavior that leads to the violation of P. However, since only bounded executions are considered and Q_r^k generally under-approximates Q_r for feasible choices of k (see [4] for a discussion on the length of computation traces), this technique only serves to prove that no errors are encountered in a finite number of transitions.

As a complementary approach based on inductive invariants, a state invariant Q_{Inv} such that $Q_r \subseteq Q_{Inv} \subseteq Q$, allows us to prove that all reachable states satisfy P because

$$\nexists q \in Q_{Inv} \;.\; \neg P(q) \;\rightarrow\; \nexists q \in Q_r \;.\; \neg P(q) \;\rightarrow\; \forall q \in Q_r \;.\; P(q) \;.$$

However, due to the over-approximation, this is not sufficient to prove the existence of reachable states violating the given property (see Fig. 1), since we have

$$\exists q \in Q_{Inv} \;.\; \neg P(q) \;\not\rightarrow\; \exists q \in Q_r \;.\; \neg P(q) \;.$$

An invariant Q_{Inv} is computable through strategies developed for the analysis of Petri nets [11], metabolic networks [9], and DNA circuits [21], where it captures constraints such as mass-conservation (various techniques from hardware and software analysis, *e.g.,* abstract interpretation [5] also apply).

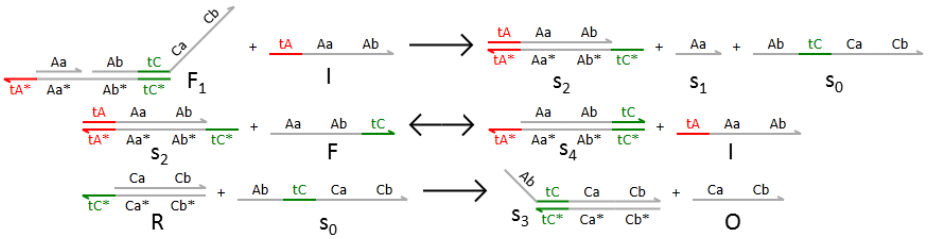

Fig. 2. `FANOUT` Gate.

To analyze chemical reaction networks and DNA circuits, we combine the BMC and state invariant approaches, where we prove that states with given properties are unreachable in any number of steps using Q_{Inv} but use BMC to guarantee the reachability of states and identify finite computation traces with specific behavior.

3 SMT Analysis of DNA Strand Displacement Circuits

In the following, we focus on circuits constructed as DNA strand displacement (DSD) devices [24], which is the DNA computing paradigm supported in Visual DSD [14]. In [21] we utilized the known structure of DNA species in these systems to develop an approach for the computation of constraints that was specific to DSD circuits. Intuitively, the individual DNA strands from which all species in the system are composed are preserved and their total amounts remain unchanged. A set of constraints was computed to capture this *conservation of strands* property, which allowed the computation of a state invariant Q_{Inv} (Sec. 2) based on the numbers of strands present initially (in state q_0). In Fig. 3 we illustrate this computation for the `FANOUT` gate shown in Fig. 2 (a component of the circuit we study in Sec. 4), which produces multiple copies of the output species O through a reporter R for a given input I, where the degree of "fanout" is controlled through the amount of species F.

Throughout the rest of this section, we present new strategies extending the approach and enabling the application of SMT-based methods to the analysis of large scale DSD circuits.

3.1 Identification of Inactive Reactions

As discussed in Sec. 2, our analysis strategy based on state invariants (*e.g.* computed using the method from [21] as in Fig. 3) is conservative, leading to the possible identification of spurious (unreachable) states. We exemplify this[1] on the `FANOUT` gate (Fig. 2), for which the following constraints are derived when

[1] Note that this example is similar to the one from Fig. 3, with the exception of the input $q_0(I) = 0$

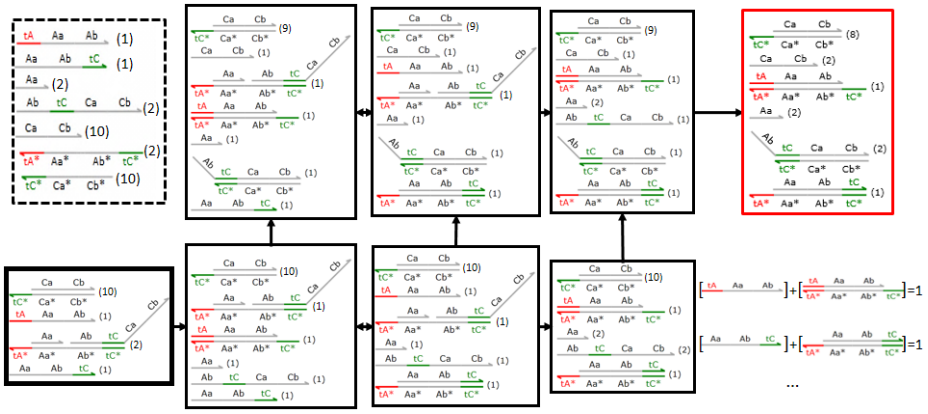

Fig. 3. Strand conservation along a computation of the `FANOUT` gate (Fig. 2). The system is initialized in $q_0 = \{(R, 10), (I, 1), (F_1, 2), (F, 1)\}$ (thick black border) and terminates in the state shown with a red border. The total number of DNA strands (top left box) remains unchanged in each state, which is captured using a set of constraints (*e.g.* bottom right). Each constraint captures information about a DNA strand shared between species (*i.e.* in each state, the sum of the molecule numbers of all species sharing strand s is equal to the number of s present in the system), which defines a state invariant Q_{Inv}. See Eqn.(2) for a related example and [21] for additional details.

the system is initialized in state $q_0 = \{(R, 10), (F_1, 2), (F, 1)\}$:

$$\begin{aligned} q \in Q_{Inv} \leftrightarrow q \in Q \wedge \\ [q(s_2) + q(I) = 0] \wedge [q(s_1) + q(F_1) = 2] \wedge [q(s_4) + q(F) = 1] \wedge \\ [q(s_3) + q(s_0) + q(F_1) = 2] \wedge [q(O) + q(R) = 10] \wedge \\ [q(s_4) + q(s_2) + q(F_1) = 2] \wedge [q(s_3) + q(R) = 10] \ . \end{aligned} \tag{2}$$

Given these constraints, only two terminal states $q_1, q_2 \in Q_{Inv}$ are possible where $q_1 = q_0$ and

$$q_2 = \{(R, 9), (O, 1), (F_1, 1), (s_1, 1), (s_3, 1), (s_4, 1)\} \ .$$

While q_1 satisfies the expected behavior of the circuit (no output is produced without input) state q_2 violates it. However, due to the over-approximation of Q_{Inv} (see Sec. 2), this does not directly imply that the `FANOUT` gate is flawed.

A closer inspection of this example reveals that, when initialized in state q_0, no reactions are enabled for this system but such information is not captured in the derived constraints and therefore spurious terminal states are identified. To decrease this conservativeness, we use the available constraints Q_{Inv} to identify reactions that are disabled for any reachable state of the system (this might require a call to the SMT solver for each $r \in \mathcal{R}$ but does not involve deep reasoning for most). Then, we use this information to identify species which are never produced (resp. consumed) and constrain their abundances to only decrease (resp. increase) from their initial values. The procedure is repeated iteratively until no additional constraints are derived (see Alg. 1).

Algorithm 1 Given a DSD circuit $(\mathcal{S}, \mathcal{R})$ encoded as $\mathcal{T} = (Q, q_0, T)$ and an invariant Q_{Inv}, derive additional constraints to produce $Q'_{Inv} \subseteq Q_{Inv}$

Initialize $Q'_{Inv} := Q_{Inv}$
repeat
  $\mathcal{R}_e := \{r \in \mathcal{R} \mid \exists q \in Q'_{Inv} \,.\, \mathit{enabled}(r, q)\}$ {possibly enabled reactions}
  $\mathcal{S}_p := \bigcup_{r \in \mathcal{R}_e} \{s \in \mathcal{S} \mid s \in P_r\}$ {producible species}
  $\mathcal{S}_c := \bigcup_{r \in \mathcal{R}_e} \{s \in \mathcal{S} \mid s \in R_r\}$ {consumable species}
  $Q_{tmp} := \{q \in Q \mid \bigwedge_{s \in (\mathcal{S} \setminus \mathcal{S}_p)} q(s) \leq q_0(s) \wedge \bigwedge_{s \in (\mathcal{S} \setminus \mathcal{S}_c)} q(s) \geq q_0(s)\}$
  done := $(Q'_{Inv} = Q'_{Inv} \setminus Q_{tmp})$
  $Q'_{Inv} := Q'_{Inv} \setminus Q_{tmp}$
until done
return Q'_{Inv}

Applying Alg. 1 to the `FANOUT` gate produces the following additional constraints, which are sufficient to eliminate the spurious terminal state q_2:

$$\begin{aligned} q \in Q'_{Inv} \leftrightarrow q \in Q_{Inv} \wedge \\ [q(F) \geq 1] \wedge [q(R) \leq 10] \wedge [q(I) \leq 0] \wedge [q(F) \leq 1] \wedge [q(s_2) \leq 0] \wedge \\ [q(s_1) \leq 0] \wedge [q(s_0) \leq 0] \wedge [q(s_4) \leq 0] \wedge [q(F_1) \leq 2] \wedge [q(F_1) \geq 2] \,. \end{aligned}$$

The use of Alg. 1 is not guaranteed to eliminate all unreachable states captured in Q_{Inv}. In other words, even though $Q'_{Inv} \subseteq Q_{Inv}$, the invariant still over-approximates the reachable states (*i.e.* in general, $Q_r \subseteq Q'_{Inv}$) and, therefore, unreachable states $q \notin Q_r$ such that $q \in Q'_{Inv}$ might still exist. Even so, the invariant strengthening strategy implemented in Alg. 1 is useful, particularly for the analysis of DNA circuits as the ones discussed in Sec. 4. For such designs, system inputs are encoded using the availability of chemical species where, for specific input values, certain species are not supplied. In these cases, Alg. 1 identifies reactions that are never enabled and restricts Q'_{Inv} accordingly.

3.2 Encoding Generalization

To identify erroneous computations for large DSD circuits such as the ones from Sec. 3, a BMC strategy requires prohibitively long paths, since the transition relation from Eqn. (1) only captures the execution of a single reaction per step. In the following, we relax this requirement by abstracting the exact number of consecutive executions of a reaction. Given states $q, q' \in Q$

$$\mathrm{reach}(q, q', r, n) = \begin{bmatrix} 0 \leq n \leq min_{s \in R_r}\{\lfloor \frac{q(s)}{R_r(s)} \rfloor\} & \wedge \\ \bigwedge_{s \in S} q'(s) = q(s) - nR_r(s) + nP_r(s) & \end{bmatrix}$$

expresses the property that state q' is reachable from q through n consecutive executions of reaction $r \in \mathcal{R}$. The condition that the reaction is enabled is implicitly captured in the choice of n (*i.e.* if r is disabled, then $0 \leq n \leq 0$). The

transition relation

$$T(q, q') \leftrightarrow q' \in Q_{Inv} \wedge \bigvee_{r \in \mathcal{R}} \exists n \,.\, \text{reach}(q, q', r, n)$$

captures multiple executions of the same reaction in a given step and therefore allows us to consider shorter computation traces. Note that this does not influence the completeness of the approach as single-reaction steps are also allowed. Furthermore, available constraints (*e.g.* derived as in Sec. 3.1) are captured directly in the transition relation.

Besides re-defining the transition relation, we generalize the transition system representation of a DSD circuit to $\mathcal{T} = (T, Q_0, Q)$ where $Q_0 \subseteq Q$ is a (possibly infinite) set of initial states. While currently, a circuit is defined using a unique initial state (population of species) within Visual DSD, it is natural to reason about the behavior of certain systems under a range of possible inputs, encoded through the abundances of chemical species at the beginning of a computation, which we illustrate in Sec. 4. Note that an invariant computed as in [21] depends on the initial state and is therefore denoted as $Q_{Inv}(q), q \in Q_0$ in the following. While applying Alg. 1 to explicit initial states $q_0 \in Q_0$ was sufficient for the circuits we consider in Sec. 3, additional strategies are required for other systems.

3.3 Implementation of methods in Visual DSD

We developed a prototype implementation of the methods reviewed in Sec. 2, together with the extension described in this section, as part of Visual DSD. The implementation makes us of the Z3 theorem prover [6], which provides efficient decision procedures for several theories including bit-vectors [20]. This allows us to specify various system properties and automatically verify them during the DSD circuit design process. The experimental results presented in Sec. 4 were obtained on 2.5 GHz Intel L5420 CPU machines with a 2 GB memory limit where computation required under a minute per benchmark.

4 Functional Analysis of a 4-bit Square Root Circuit

In this section, we study the 4-bit square root circuit design from [2]. First, we formalize and analyze the functional behavior of the individual components used for the construction of this system and then apply our method to study properties of the full system, when multiple copies of the circuit are operating in parallel. For each component, we define a set of initial states $Q_0 \subset Q$ capturing the possible abundances of species (inputs, gates, etc) present at the beginning of a computation and a property $P(q_0, q)$ that describes the expected output for a given input, encoded as part of the initial state q_0. To prove the correctness of a circuit, we need to show that $\forall q_0 \in Q_0, q \in Q_r \,.\, \mathit{terminal}(q) \rightarrow P(q_0, q)$ (*i.e.* for all input values, the correct output is produced when the computation terminates, regardless of the initial abundances of other species). We compute the state invariant Q_{Inv} using the procedure from [21] illustrated in Fig. 3 (or

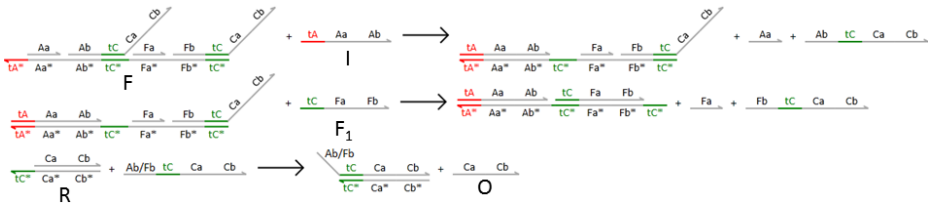

Fig. 4. Modified `FANOUT` Gate with fanout degree of two.

Q'_{Inv} extended through Alg. 1) and use it to prove that $terminal(q) \wedge \neg P(q_0, q)$ is not satisfiable for states $q_0 \in Q_0$ and $q \in Q_{Inv}(q_0)$ (*i.e.* no terminal state exists where incorrect output is produced). This formula is trivially unsatisfiable when no terminal states exist (*i.e.* $\forall q \in Q_{Inv} \; . \; \neg terminal(q)$) and, to conclude the proof, we show that this is not the case. When incorrect behavior is identified through this strategy (as is the case for one of the square root circuit designs we explore), we use BMC (see Sec. 2) to identify an error trace.

`FANOUT` Gate The `FANOUT` gate (Fig. 2) introduced in Sec. 3.1 is intended to split a particular input species I into multiple copies of output O, where the degree of fanout is controlled through species F. We define the set $Q_0 = \{q \in Q'_{Inv} \mid q(s) > 0 \text{ if } s \in \{R, F_1\} \text{ and } q(s) = 0 \text{ if } s \notin \{R, F_1, I, F\}\}$ (*i.e* gates F_1, reporters R and possibly fanout F and input I species are present initially). We used our analysis approach to prove that, for all initial states $q_0 \in Q_0$ and terminal states $q \in Q'_{Inv}(q_0)$ the behavior of the component formalized as

$$P_{\texttt{FANOUT}}(q_0, q) \leftrightarrow q(O) = \begin{cases} q_0(I) + q_0(F) & \text{when } q_0(I) > 0 \\ 0 & \text{otherwise} \end{cases}$$

holds, as long as the additional conditions $q_0(R) \geq q_0(I) + q_0(F)$ and $q_0(F_1) \geq I$ are satisfied (*i.e.* there is an excess of reporters and gates). Note that this behavior holds regardless of the specific input $q_0(I)$ and fanout $q_0(F)$ settings. Thus, the component adds a constant to the input value (when input is present) but replicates the desired behavior $q(O) = m \cdot q_0(I)$ only when $q_0(F) = (m - 1) \cdot q_0(I)$ and, as a result, $q_0(F)$ must be precisely tuned for a specific input value. To obtain the correct behavior for arbitrary inputs, we redesign the gate for fanout $m = 2$ as in Fig. 4 and show that the expected behavior

$$P_{\texttt{FANOUT2}}(q_0, q) \leftrightarrow q(O) = 2 \cdot q_0(I)$$

is now satisfied when $q_0(F) \geq q_0(I)$, $q_0(R) \geq 2 \cdot q_0(I)$, and $q_0(F_1) \geq 2 \cdot q_0(F)$ (*i.e.* gates and reporters are in excess).

`AND` Gate The `AND` gate (Fig. 5) is a component designed to implement the corresponding logical operation. We define the set $Q_0 = \{q \in Q_{Inv} \mid q(s) > 0 \text{ if } s \in \{R, G\} \text{ and } q(s) = 0 \text{ if } s \notin \{R, G, I_A, I_B\}\}$ (*i.e* gates G, reporters R and

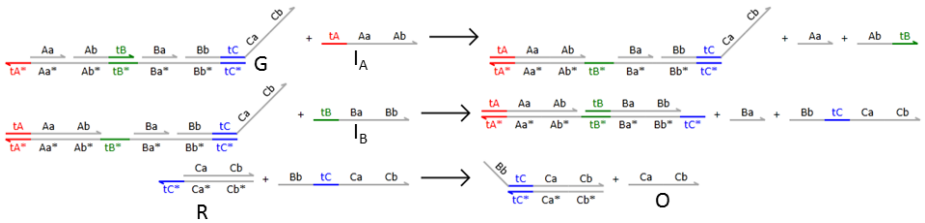

Fig. 5. AND Gate

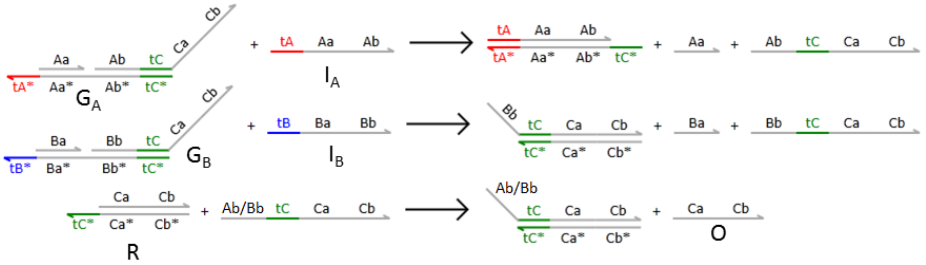

Fig. 6. OR Gate.

possibly inputs I_A, I_B are present initially). We prove that, for all initial states $q_0 \in Q_0$ and terminal states $q \in Q_{Inv}(q_0)$,

$$P_{\texttt{AND}}(q_0, q) \leftrightarrow q(O) = \min\{q_0(I_A), q_0(I_B)\} \tag{3}$$

holds, as long as $q_0(R) \geq q_0(I_A), q_0(R) \geq q_0(I_B), q_0(G) \geq q_0(I_A)$ and $q_0(G) \geq q_0(I_B)$. The logical behavior of the AND gate is formalized using a threshold θ where a signal represents the logical "true" if and only if the number of molecules is greater than θ *i.e.* $[q(O) > \theta] \leftrightarrow [q_0(I_A) > \theta] \wedge [q_0(I_B) > \theta]$, which implements the desired logical operation. This behavior follows directly from Eqn. (3) and was also verified using our approach for arbitrary values of θ.

OR Gate The desired behavior of the OR gate (Fig. 6) is defined similarly to the AND gate described above. We define the set $Q_0 = \{q \in Q_{Inv} \mid q(s) > 0$ if $s \in \{R, G_A, G_B\}$ and $q(s) = 0$ if $s \notin \{R, G_A, G_B, I_A, I_B\}\}$ (*i.e* gates G_A, G_B, reporters R and possibly inputs I_A, I_B are present initially). We prove that, for all initial states $q_0 \in Q_0$ and terminal states $q \in Q_{Inv}(q_0)$,

$$P_{\texttt{OR}}(q_0, q) \leftrightarrow q(O) = q_0(I_A) + q_0(I_B) \tag{4}$$

holds, as long as $q_0(G_A) \geq q_0(I_A)$, $q_0(G_B) \geq q_0(I_B)$ and $q_0(R) \geq q_0(I_A) + q_0(I_B)$. As before, the logical behavior of the OR gate is formalized through a threshold θ but here, a signal represents the logical "true" only if the number of molecules is greater than θ *i.e* $[q_0(I_A) > \theta] \vee [q_0(I_B) > \theta] \rightarrow [q(O) > \theta]$. This behavior follows from Eqn. (4) and was also verified using our approach for arbitrary values of θ. To avoid issues with the composition of multiple OR gates, the logical "false" is left undefined for this component, which is sufficient for the implementation of the square root circuit discussed next, where a dual-rail signal encoding is used.

Full Square Root Circuit The square root circuit takes an input between 0 and 15 represented as a 4-bit binary number encoded using the concentrations of 8 input chemical species ($\text{STRAND}_0, \ldots, \text{STRAND}_7$) in dual-rail logic (see [2, 17] for details). The circuit computes the largest integer smaller than or equal to the square root of the input and represents this 2-bit output using the concentrations of 4 species (M_0, L_0, M_1, L_1). Following the design from [17], the circuit is separated into a number of logical blocks (composed of the `AND`, `OR`, and `FANOUT` gates studied above), each of which computes a separate part of the output. Here, we study three different implementations of this circuit inspired by the design from [2]. For one (referred to as $\texttt{SQRT}_1$), distinct DNA domains are used to prevent crosstalk between the logical blocks. However, this increases the total number of domains required, which potentially increases the cost of circuit construction. Motivated by this, we explore two simplified designs where domains are shared between logical blocks and either the original `FANOUT` gate design from Fig. 2 ($\texttt{SQRT}_2$) or the modified one from Fig. 4 ($\texttt{SQRT}_3$) is used. All three circuits are designed to implement the mathematical operation

$$P_{\texttt{SQRT}}(q_0, q) \leftrightarrow O(q) = \lfloor \sqrt{I(q_0)} \rfloor)$$

where $I(q_0) \in \{0, \ldots, 15\}$ for $q_0 \in Q_0$ and $O(q) \in \{0, \ldots, 3\}$ for $q \in Q$ denote the input and output of a circuit, specific to each design.

For $\texttt{SQRT}_1$, we assume that N copies of each functional block are operating in parallel. We use the existing Visual DSD model from [2] which defines the inputs I and outputs O to capture the requirement that each circuit copy computes the correct output independently (*e.g.* $O(q) = 0 \leftrightarrow [q(M_0) = N] \wedge [q(M_1) = 0] \wedge [q(L_0) = N] \wedge [q(L_1) = 0]$ and $O(q) = 3 \leftrightarrow [q(M_0) = 0] \wedge [q(M_1) = N] \wedge [q(L_0) = 0] \wedge [q(L_1) = N]$). The strategy from Sec. 3.1 allows us to prove that $\texttt{SQRT}_1$ implements this behavior correctly for $N = \{1, 10^2, 10^3, 10^6\}$. Note that this circuit is distinct from the original, localized setup from [2], where only a single copy of the circuit is considered in isolation.

To obtain requirements for a population-based design that are independent of the precise numbers of molecules used as inputs, we define thresholds θ_I and θ_O where $\theta_O \leq \theta_I$. An input bit is set to true by including more than θ_I molecules of the corresponding species which defines $I()$ (*e.g.* $I(q_0) = 0 \leftrightarrow [\text{STRAND}_0 > \theta_I] \wedge [\text{STRAND}_2 > \theta_I] \wedge [\text{STRAND}_4 > \theta_I] \wedge [\text{STRAND}_6 > \theta_I]$). Similarly, an output bit is considered true if θ_O or more molecules of an output species are present (*e.g.* $O(q) = 0 \leftrightarrow [q(M_0) \geq \theta_O] \wedge [q(M_1) = 0] \wedge [q(L_0) \geq \theta_O] \wedge [q(L_1) = 0]$). In practice, the numbers of molecules for each gate of a circuit cannot be set precisely at the beginning of a computation and therefore we consider thresholds G_l and G_u where, for each initial state $q_0 \in Q_0$, $G_l \leq q_0(s) \leq G_u$ for a gate $s \in \mathcal{S}$. This defines the set of initial states Q_0 for a dual-rail input encoding (*i.e.* where $\forall q_0 \in Q_0 \;.\; q_0(\text{STRAND}_i) = 0 \vee q_0(\text{STRAND}_{i+1}) = 0$ for $i = 0, 2, 4, 6$) where no species other than gates and inputs are present initially. Finally, we assume that the error with which gates are supplied $n = G_u - G_l$ is set to an arbitrary number, which simplifies the analysis but is not restrictive in practice.

We use the formulation described above to show that design $\texttt{SQRT}_2$ leads to erroneous behavior where, for some input combinations, no terminal states

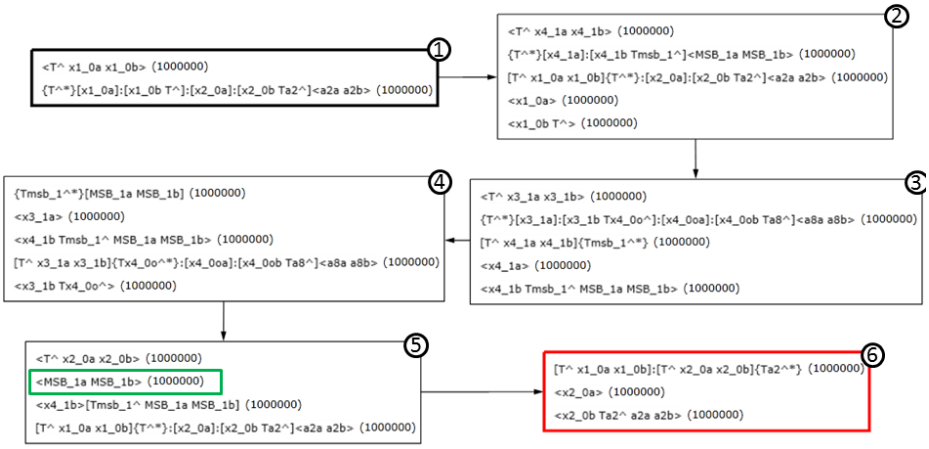

Fig. 7. For the $\texttt{SQRT}_3$ design with $\theta_O = 1, n = 0$ and $G_l = Gu = \theta_I = 10^6$ (*i.e.* one million copies of both the circuit and its inputs are present), initialized in state q_0 where $I(q_0) = 12$ (input is 12) a computation trace producing incorrect output was identified using the bounded model checking analysis method described in Sec. 2. In each state, only the species with nonzero abundances and participating in the reaction captured by the following transition are displayed. While part of the correct output is produced (step 5), the full output is incorrect in the terminal state (highlighted in red) and violates the dual-rail logical encoding (neither $L0$ nor $L1$ is produced). By considering multiple executions of a reaction per transition, short computation traces representing a large number of individual reactions are identified.

are ever reached. In other words, there exists an initial state $q_0 \in Q_0$ such that no state $q \in Q_{Inv}(q_0)$ is terminal, which violates a requirement that all computations eventually terminate.

For $\texttt{SQRT}_3$, we prove that the required behavior is satisfied for all input and gate settings, as long as (i) $G_l > \theta_O$, (ii) $\theta_I > 3G_u$, and (iii) $G_l > n\theta_O$. These additional constraints capture important properties of the circuit that ensure its correct operation. Constraint (i) captures the property that it is only possible to produce as much output as there are available reporter gates, (ii) ensures that enough of the input is supplied to be processed by each logical block of the circuit, and (iii) formalizes the accuracy with which the output must be measured as a function of the absolute number G_l and error n with which the circuit gates are supplied. Intuitively, to measure stronger output signal (*i.e.* where θ_O is higher) the lowest possible number of gates G_l must be increased while the error n is decreased, which might also require the addition of more input (if θ_I is higher). To confirm these requirements we used our method to find erroneous computation traces when $G_l = G_u = \theta_I$ (Fig. 7). Such behavior is also observed in stochastic simulations of a detailed models of the circuit (capturing the chemical kinetics) but becomes rare as the number of gates is increased and is easily missed (Fig. 8).

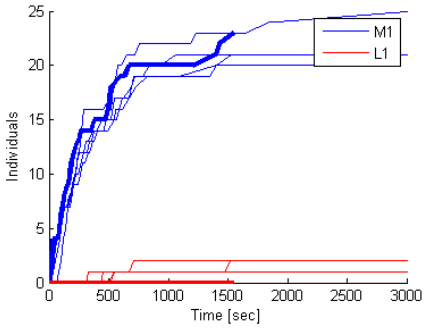

Fig. 8. For the $\texttt{SQRT}_3$ design with $\theta_O = 1, n = 0$ and $G_l = Gu = \theta_I = 20$ (*i.e.* 20 copies of both the circuit and its inputs are present), the correct outputs are not produced for one out of the five simulated trajectories when the circuit is initialized in state q_0 where $I(q_0) = 12$ (input is 12). Output species $M1$ and $L1$ are represented by blue and red lines, respectively (output species $M0$ and $L0$ remain absent throughout these computations). The trajectory capturing the erroneous computation (reaching a terminal state at around 1500s) is highlighted.

In the analysis described above, we show that if $\texttt{SQRT}_3$ terminates, the correct output is produced. To prove that the system terminates for any choice of inputs, we employ standard Petri Net theory techniques for the computation of T-invariants [11], which reveal that, in this particular design, no cycles are possible and, therefore, termination is guaranteed.

5 Discussion

Despite the theoretical complexity of DNA strand displacement analysis problems [19], we demonstrate that an SMT-based method enables the analysis of properties of large-scale DNA computing systems and is capable of handling the largest designs currently constructed in wet labs. Although we focus on strand displacement, using the power of SMT methods and their underlying solvers to address challenging analysis questions in other DNA computing paradigms remains an interesting topic for future research. Besides improved scalability, a major advantage of the method compared to previous analysis strategies (*e.g.* [15]) is that it enables us to formalize and prove the functional correctness of systems under arbitrary inputs or large numbers of copies operating in parallel.

There are several potential uses we envision for these methods in the field. First, formalizing functionality requirements during the circuit design process and invoking the method to prove the correctness of a model is crucial in medical and industrial applications of DNA computing. For flawed designs, our method allows the identification of erroneous computation traces. Such formal "bug hunting" has become indispensable for the design of software and hardware [12] where subtle errors are hard to detect using simulation alone. Enabling a designer to explicitly state the expected functionality and related assumptions, is often sufficient to identify potential problems when reusing components in larger designs. Furthermore, computation traces generated during analysis help identify specific inputs and conditions leading to a given behavior in the model, allowing the actual behavior of the circuit to be examined in the lab.

Second, analysis methods are useful as "compiler optimizations", for example by detecting reactions that will not be enabled for given inputs (as in our method) thereby speeding up simulation. A tighter integration with a compiler (*e.g.* Visual DSD) also benefits the analysis - our preliminary investigation suggest that, in specific cases, disabled reactions can safely be removed from the system during compilation rather than by using the more general procedure from Sec. 3.1. Incorporating analysis capabilities within DNA compilers also allows useful information to be provided to the designer and helps in understanding circuit behavior.

Despite all analysis and design efforts, our methods only allow us to gain confidence in the correctness of the available models, while the functionality of a DNA circuit is ultimately determined experimentally in the lab. Even so, models can be extended to include the additional complexities (*e.g.* unproductive reactions, leak rates, etc.) required to capture the behavior of a circuit more accurately. Furthermore, the ability to systematically analyze models has the potential to aid the construction of circuits in the lab by allowing observed experimental results that are also possible in the model (although potentially rare) to be distinguished from situations where the modeling assumptions fail.

While the analysis of large-scale, single-molecule resolution models is the focus in this paper, SMT-based methods also enable the encoding and analysis of approximations such as (non-linear) ODEs, where species concentrations are described as continuous values but important system behavior is potentially missed. In our current work, we focused specifically on a class of CRNs where computations do not depend on reaction kinetics. Recently, the class of mathematical functions computable in chemical reactions networks with arbitrary kinetics was characterized [3] and practical advantages of such systems were highlighted (*e.g.* only a set of inputs is sufficient to initiate a computation [8]). Extending the method described here to probabilistic systems to capture the additional complexity of chemical kinetics (*e.g.* through the use of stochastic SMT [10]) is an ongoing effort. For instance, in the present work, we study several DSD-circuits, inspired by the localized square root circuit from [2]. Although this design is similar to the one from [17], it does not use *seesaw* gates as a basic logical component. The seesaw gate is capable of implementing either AND- or OR-type behavior, but relies on differential binding rates between certain species to do so and, as a result, there is a low probability that the circuit will compute the incorrect output. Since chemical kinetics are not currently considered in our representation, low probability computation traces (*e.g.* where an OR-gate behaves as an AND-gate) would be identified as erroneous using our approach, without taking into account their actual probability.

In this paper we consider DSD circuits where all species and reactions are generated *a priori* (which is often the case for circuits of practical interest, but with notable exceptions [15]) and the output is measured once a state is reached where no additional reactions are possible. For more general chemical systems and other DSD circuits, this is not always the case (*e.g.* when an output signal is computed but other auxiliary reactions are still enabled). Thus, DNA

computing circuits may be viewed as reactive systems that continually perform computation and react to external signals, rather than circuits that compute some output and terminate. Richer specifications (*e.g.* as captured in temporal logic) are useful for defining more general behavioral properties and are possible in the proposed framework (*e.g.* through standard encodings as in [1]). Besides capturing the functional properties discussed here, in [21] our methods proved useful for studying certain structural properties such as the presence of exposed DNA sequences in the transducer circuit designs from [15].

While termination is generally a challenging problem [23], it is possible to obtain termination proofs for many concrete models. Here, we obtain such a proof by adapting methods developed for Petri nets [11] to the particular circuit design we study in this paper. Extending this method and adapting other recent techniques to study termination in DSD circuits is a promising direction of future research. More generally, several of the properties we study are closely related to ones defined for Petri nets [11] and adapting techniques developed for their analysis is currently ongoing. Notably, the use of Petri net methods (instead of the strand-conservation strategy from [21]) to compute invariants for DSD circuits does not substitute the strengthening procedure from Sec. 3.1 for the examples we consider.

The iterative strengthening of inductive invariants (as in our strategy from Sec. 3.1) has been studied in the context of software and hardware verification, and the development of such methods for DNA circuits is being investigated within our framework. The application of such methods also provides a promising strategy for automatically uncovering important properties of circuit designs, such as the ones we defined for the square root circuit and its components. Finally, we study and prove the correctness of components of complex DNA circuits in isolation but cannot guarantee that this behavior is maintained when these components are used within larger systems - modularizing the analysis of DNA circuits is an auspicious future direction.

References

1. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *TACAS*, pages 193–207, 1999.
2. H. Chandran, N. Gopalkrishnan, A. Phillips, and J. Reif. Localized hybridization circuits. *DNA17*, pages 64–83, 2011.
3. H. L. Chen, D. Doty, and D. Soloveichik. Deterministic Function Computation with Chemical Reaction. In *DNA18*, volume 7433 of *LNCS*, pages 25–42, 2012.
4. A. Condon, B. Kirkpatrick, and J. Maňuch. Reachability Bounds for Chemical Reaction Networks and Strand Displacement Systems. In *DNA18*, volume 7433 of *LNCS*, pages 43–57, 2012.
5. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM, 1977.
6. L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

7. Q. Dong. A bisimulation approach to verification of molecular implementations of formal chemical reaction networks. Master's thesis, Stony Brook University, 2012.
8. D. Doty and M. Hajiaghayi. Leaderless deterministic chemical reaction networks. arXiv:1304.4519, 2013.
9. I. Famili and B. O. Palsson. The convex basis of the left null space of the stoichiometric matrix leads to the definition of metabolically meaningful pools. *Biophysical journal*, 85(1):16–26, 2003.
10. M. Fränzle, H. Hermanns, and T. Teige. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In *HSCC'08*, volume 4981 of *LNCS*, pages 172–186. Springer, 2008.
11. M. Heiner, D. Gilbert, and R. Donaldson. Petri Nets for Systems and Synthetic Biology. *Formal Methods for Computational Systems Biology*, 5016:215–264, 2008.
12. R. Kaivola, R. Ghughal, and N. Narasimhan. Replacing Testing with Formal Verification in Intel® CoreTM i7 Processor Execution Engine Validation. *Computer Aided Verification*, pages 414–429, 2009.
13. M. R. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society, Interface*, 9(72):1470–85, July 2012.
14. 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):3211–3, 2011.
15. R. Lakin, Matthew and A. Phillips. Modelling, simulating and verifying turing-powerful strand displacement systems. *DNA17*, pages 130–144, 2011.
16. Z. Manna and A. Pnueli. *Temporal verification of reactive systems: safety*, volume 2. Springer, 1995.
17. L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–201, 2011.
18. S. W. Shin. Compiling and verifying DNA-based chemical reaction network implementations. Master's thesis, California Institute of Technology, 2012.
19. C. Thachuk and A. Condon. Space and Energy Efficient Computation with DNA Strand Displacement Systems. In *DNA18*, volume 7433 of *LNCS*, pages 135–149, 2012.
20. C. M. Wintersteiger, Y. Hamadi, and L. de Moura. Efficiently solving quantified bit-vector formulas. *Formal Methods in System Design*, 42(1):3–23, 2013.
21. B. Yordanov, C. M. Wintersteiger, Y. Hamadi, and H. Kugler. SMT-based analysis of biological computation. In *NASA Formal Methods*, volume 7871, pages 78–92. Springer, 2013.
22. `Z34Bio` at `rise4fun` Software Engineering Tools from Microsoft Research, http://rise4fun.com/z34biology, 2013.
23. G. Zavattaro and L. Cardelli. Termination problems in chemical kinetics. In *CONCUR '08*, pages 477–491, 2008.
24. D. Y. Zhang and G. Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nat Chem*, 3(2):103–113, 2011.