Computing Threshold Circuits with Bimolecular Void Reactions in Step Chemical Reaction Networks

Rachel Anderson[1], Bin Fu[1], Aiden Massie[1], Gourab Mukhopadhyay[1], Adrian Salinas[1], Robert Schweller[1], Evan Tomai[2], and Tim Wylie[1(✉)]

[1] University of Texas Rio Grande Valley, Edinburg, TX 78539-2999, USA
timothy.wylie@utrgv.edu
[2] University of Texas Dallas, Richardson, TX 75080-3021, USA

Abstract. Step Chemical Reaction Networks (step CRNs) are an augmentation of the Chemical Reaction Network (CRN) model where additional species may be introduced to the system in a sequence of "steps." We study step CRN systems using a weak subset of reaction rules, *void* rules, in which molecular species can only be deleted. We demonstrate that step CRNs with only void rules of size (2,0) can simulate threshold formulas (TFs) under linear resources. These limited systems can also simulate threshold *circuits* (TCs) by modifying the volume of the system to be exponential. We then prove a matching exponential lower bound on the required volume for simulating threshold circuits in a step CRN with (2,0)-size rules under a restricted *gate-wise* simulation, thus showing our construction is optimal for simulating circuits in this way.

1 Introduction

Chemical Reaction Networks (CRNs) are a well-established model of chemistry. In this model, chemical interactions are modeled as molecular species that react to create products according to a set of reaction rules. CRNs have been extensively studied since their standard formulation in the 1960s [6,7]. Several equivalent models were also introduced around the same time with Vector Addition Systems (VASs) [20] and Petri-nets [24]. Further, Population Protocols [3] are a restricted form of the model focused on bimolecular reactions.

Step CRNs. These models all assume a discrete starting number of species or elements and reaction rules that dictate how they can interact. Thus, any change in species numbers is only through these interactions. Motivated by standard laboratory procedures where additional chemical species may be added to an initial container of species after a set of reactions has passed, we utilize an extension to the CRN model known as the Step Chemical Reaction Network model (step CRN) first introduced in [2]. The step CRN model adds a sequence

This research was supported in part by National Science Foundation Grant CCF-2329918.

D.-J. Cho and J. Kim (Eds.): UCNC 2024, LNCS 14776, pp. 253–268, 2024.
https://doi.org/10.1007/978-3-031-63742-1_18

Table 1. Results in the paper related to computing circuits with $(2, 0)$ void rules in step CRN systems. D is the circuit's depth, G is the number of gates in a circuit, and F_{out} is the maximum fan-out of a gate in the circuit.

Function Computation						
Rules	Species	Steps	Volume	Simulation	Family	Ref
$(2, 0)$	$O(G)$	$O(D)$	$O(G)$	Gate-Wise	**TF** Formulas	Theorem 1
$(2, 0)$	$O(G)$	$O(D)$	$O(G{F_{out}}^{D})$	Gate-Wise	**TC** Circuits	Theorem 2
$(2, 0)$	-	-	$2^{\Omega(D)}$	Gate-Wise	**TC** Circuits	Theorem 3

of discrete steps where additional species can be added to the existing CRN configuration after waiting for all possible reaction rules to occur in the system.

Bimolecular Void Rules. In this paper, we use the terms *reaction* and *rule* interchangeably to denote a reaction rule. We focus on step CRN systems that include only *bimolecular void rules* ($(2, 0)$ rules) which are CRN rules that have two reactants and no products, and thus can only delete existing species. Void rules of size $(2, 0)$ and $(3, 0)$ were first studied within the standard CRN model in the context of the reachability problem [1]. While standard CRN rules are powerful thanks to their ability to replace, delete or create new species, (2,0) rules have extremely limited power to compute even simple functions in a standard CRN [2]. In contrast, we show $(2, 0)$ void rules become capable of computing Threshold Formulas (TF) and Threshold Circuits (TC) in the step CRN model. Threshold circuits are a computationally universal class of Boolean circuits that have practical application in deep learning, and consist of any Boolean circuit made of AND, OR, NOT, and MAJORITY gates.

Computation in Chemical Reaction Networks. Computation within Chemical Reaction Networks is a well-studied topic. Within stochastic Chemical Reaction Networks with some possibility for error, the systems are Turing-complete [27]. In contrast, error-free stochastic CRNs are capable of computing semilinear functions [5,12]. Further, molecules themselves have long been studied as a method of information storage and Boolean logic computation. In particular, CRNs and similar models have been extensively studied in these areas [8,9,11,13,15,19,21,25,26]. Logic gates such as AND [11,14,22,26,28,30], OR [11,14,26,28], NOT [11], XOR [11,30], NAND [11,13,15,29], NOR [11], Parity [16–18] and Majority [4,10,23] have also been explored.

Our Contributions. Table 1 gives an overview of our results, and the paper is formatted to introduce the general techniques and then expand into the necessary details to prove these results. Section 3.2 shows how a step CRN, using (2,0) void rules, computes individual logic gates. We then show how these gates can be combined to build a general construction of threshold formulas in Sect. 3.4. Theorem 1 shows how a step CRN with only (2, 0) rules is capable of computing threshold formulas with $O(G)$ species, $O(D)$ steps, and $O(G)$ volume, where G is the number of gates in a circuit and D is the depth of a circuit.

In Sect. 4, we modify this construction to compute threshold circuits with $O(G)$ species, $O(D)$ steps, and $O(G{F_{out}}^D)$ volume, where F_{out} is the maximum fan-out of the circuit. Finally, in Sect. 5 we show that the volume lower bound for simulating a circuit using *gate-wise* simulation in a step CRN with (2,0) rules is $2^{\Omega(D)}$. This lower bound is of note in that it shows the exponential volume utilized by the positive result is needed for this style of computation, and it shows a provable change in power from the polynomial volume achievable with $(3, 0)$ void rules [2].

2 Preliminaries

2.1 Chemical Reaction Networks

Basics. Let $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_{|\Lambda|}\}$ denote some ordered alphabet of *species*. A configuration over Λ is a length-$|\Lambda|$ vector of non-negative integers that denotes the number of copies of each present species. A *rule* or *reaction* has two multisets, the first containing one or more *reactant* (species), used for creating resulting *product* (species) contained in the second multiset. Each rule is represented as an ordered pair of configuration vectors $R = (R_r, R_p)$. R_r contains the minimum counts of each reactant species necessary for reaction R to occur, where reactant species are either *consumed* by the rule in some count or leveraged as *catalysts* (not consumed); in some cases a combination of the two. The product vector R_p has the count of each species *produced* by the *application* of rule R, effectively replacing vector R_r. The species corresponding to the non-zero elements of R_r and R_p are termed *reactants* and *products* of R, respectively.

The *application* vector of R is $R_a = R_p - R_r$, which shows the net change in species counts after applying rule R once. For a configuration C and rule R, we say R is applicable to C if $C[i] \geq R_r[i]$ for all $1 \leq i \leq |\Lambda|$, and we define the *application* of R to C as the configuration $C' = C + R_a$. For a set of rules Γ, a configuration C, and rule $R \in \Gamma$ applicable to C that produces $C' = C + R_a$, we say $C \rightarrow^1_\Gamma C'$, a relation denoting that C can transition to C' by way of a single rule application from Γ. We further use the notation $C \rightsquigarrow_\Gamma C'$ to signify the transitive closure of $\rightarrow^1_\Gamma$ and say C' is *reachable* from C under Γ, i.e., C' can be reached by applying a sequence of applicable rules from Γ to initial configuration C. Here, we use the following notation to depict a rule $R = (R_r, R_p)$: $\sum_{i=1}^{|\Lambda|} R_r[i]s_i \rightarrow \sum_{i=1}^{|\Lambda|} R_p[i]s_i$.

Using this notation, a rule turning two copies of species H and one copy of species O into one copy of species W would be written as $2H + O \rightarrow W$.

Definition 1 (Discrete Chemical Reaction Network). *A discrete chemical reaction network (CRN) is an ordered pair (Λ, Γ) where Λ is an ordered alphabet of species, and Γ is a set of rules over Λ.*

We denote the set of reachable configurations for a CRN (Λ, Γ) from initial configuration I as $\text{REACH}_{I,\Lambda,\Gamma}$. A configuration is called *terminal* with respect to a CRN (Λ, Γ) if no rule $R \in \Gamma$ can be applied to it. We define the subset

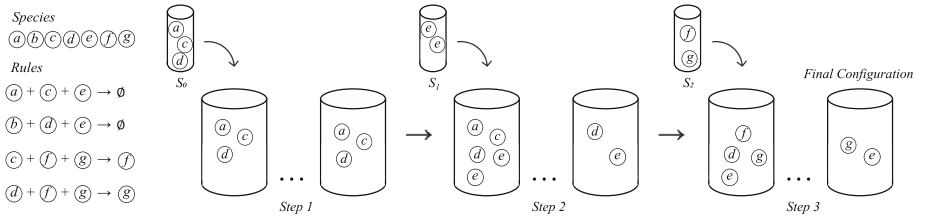


Fig. 1. An example step CRN system. The test tubes show the species added at each step and the system with those elements added. The CRN species and void rule-set are shown on the left.

of reachable configurations that are terminal as $\text{TERM}_{I,\Lambda,\Gamma}$. For an initial configuration I, a CRN (Λ, Γ) is said to be *bounded* if a terminal configuration is guaranteed to be reached within some finite number of rule applications starting from configuration I.

2.2 Void Rules

Definition 2 (Void and Autogenesis Rules). *A rule $R = (R_r, R_p)$ is a* void *rule if $R_a = R_p - R_r$ has no positive entries and at least one negative entry. A rule is an* autogenesis *rule if R_a has no negative values and at least one positive value. If the reactants and products of a rule are each multisets, a void rule is a rule whose product multiset is a strict submultiset of the reactants, and an autogenesis rule one where the reactants are a strict submultiset of the products. There are two classes of void rules,* catalytic *and* true void*. In catalytic void rules, one or more reactants remain after the rule is applied. In true void rules, such as $(2, 0)$ and $(3, 0)$ rules, there are no products remaining.*

Definition 3. *The* size/volume *of a configuration vector C is* $\texttt{volume}(C) = \sum C[i]$.

Definition 4 (Size-(i, j) Rules). *A rule $R = (R_r, R_p)$ is said to be a size-(i, j) rule if $(i, j) = (\texttt{volume}(R_r), \texttt{volume}(R_p))$. A reaction is bimolecular if $i = 2$.*

2.3 Step Chemical Reaction Networks

A step CRN is an augmentation of a basic CRN in which additional copies of some system species are added after each of a sequence of steps. Formally, a step CRN of k steps is a ordered pair $((\Lambda, \Gamma), (s_0, s_1, s_2, \ldots s_{k-1}))$, where the first element of the pair is a normal CRN (Λ, Γ), and the second is a sequence of length-$|\Lambda|$ vectors of non-negative integers denoting how many copies of each species type to add after each step.

Given a step CRN, we define the set of reachable configurations after each sequential step. To start off, let REACH_1 be the set of reachable configurations

of (Λ, Γ) with initial configuration s_0, which we refer to as the set of configurations reachable *after step 1*. Let TERM_1 be the subset of configurations in REACH_1 that are terminal. Note that after just a single step we have a normal CRN, i.e., 1-step CRNs are just normal CRNs with initial configuration s_0. For the second step, we consider any configuration in TERM_1 combined with s_1 as a possible starting configuration and define $REACH_2$ to be the union of all reachable configurations from each possible starting configuration attained by adding s_1 to a configuration in $TERM_1$. We then define TERM_2 as the subset of configurations in REACH_2 that are terminal. Similarly, define $REACH_i$ to be the union of all reachable sets attained by using initial configuration c_{i-1} at step s_{i-1} plus any element of $TERM_{i-1}$, and let $TERM_i$ denote the subset of these configurations that are terminal.

The set of reachable configurations for a k-step CRN is the set REACH_k, and the set of terminal configurations is TERM_k. A classical CRN can be represented as a step CRN with $k = 1$ steps and an initial configuration of $I = s_0$.

2.4 Computing Functions in Step CRNs

Here, we define what it means for a step CRN to compute a function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ that maps n-bit strings to m-bit strings. For each input bit, we denote two separate species types, one representing bit 0, and the other bit 1. An input configuration to represent a desired n-bit input string is constructed by selecting to add copies of *either* the 0 species or the 1 species for each bit in the target bit-string. Similarly, each output bit has two species representatives (for 0 and 1), and we say the step CRN computes function f if for any given n-bit input $x_1, \ldots, x_n$, the system obtained by adding the species for the string $x_1, \ldots, x_n$ to the initial configuration of this system in step 1 results in a final configuration whose output species encode the string $f(x_1, \ldots, x_n)$. Note that for a fixed function f, the species s_i added at each step are fixed to disallow outside computation. We now provide a more detailed formalization of this concept.

Input-Strict Step CRN Computing. Given a Boolean function $f(x_1, \ldots, x_n) = (y_1, \ldots, y_m)$ that maps a string of n bits to a string of m bits, we define the computation of f with a step CRN. An input-strict step CRN computer is a tuple $C_s = (S, X, Y)$ where $S = ((\Lambda, \Gamma), (s_0, s_1, \ldots, s_{k-1}))$ is a step CRN, and $X = ((x_1^F, x_1^T), \ldots, (x_n^F, x_n^T))$ and $Y = ((y_1^F, y_1^T), \ldots, (y_m^F, y_m^T))$ are sequences of ordered-pairs with each $x_i^F, x_i^T, y_j^F, y_j^T \in \Lambda$. Given an n-input bit string $b = b_1, \ldots, b_n$, configuration $X(b)$ is defined as the configuration over Λ obtained by including one copy of x_i^F only if $b_i = 0$ and one copy of x_i^T only if $b_i = 1$ for each bit b_i. We consider this representation of the input to be strict, as opposed to allowing multiple copies of each input bit species. The corresponding step CRN $(\Lambda, \Gamma, (s_0 + X(b), \ldots, s_{k-1}))$ is obtained by adding $X(b)$ to s_0 in the first step, which conceptually represents the system programmed with input b.

An input-strict step CRN computer *computes* function f if, for all n-bit strings b and for all terminal configurations of $(\Lambda, \Gamma, (s_0 + X(b), \ldots, s_{k-1}))$, the terminal configuration contains at least 1 copy of y_j^F and 0 copies of y_j^T if the

j^{th} bit of $f(b)$ is 0, and at least 1 copy of y_j^T and 0 copies of y_j^F if the j^{th} bit of $f(b)$ is 1, for all j from 1 to m.

We use the term *strict* to denote requiring exactly one copy of each bit species. While previous work has focused on strict computation [2], the focus here is a relaxation of this requirement in which multiple copies of each input species are permitted.

Multiple Input Relaxation. In this paper, we focus on a relaxation to strict computing that allows for multiple copies of each input bit species, i.e., we modify the strict definition by allowing some number greater or equal to 1 of each x_i^F or x_i^T species in the initial configuration (but still requiring 0 copies of the alternate choice species). A system that computes a function under this relaxation is said to be *multiple input relaxed*.

Gate-Wise Simulation. In this paper, we utilize a method of simulation we term *gate-wise* simulation, where the output of each gate is represented by a unique species and the gates are computed in the order of their depth level. In other words, multiple gates cannot use the exact same species to represent their output and a gate can only be computed once all gates in the previous depth levels have been computed. We use this method when simulating formulas and circuits with a step CRN in Sects. 3 and 4. A formal definition of gate-wise simulation is provided in Sect. 5.

2.5 Boolean and Threshold Circuits

A Boolean circuit on n variables $x_1, x_2, \ldots, x_n$ is a directed, acyclic multi-graph. The vertices of the graph are referred to as *gates*. The in-degree and out-degree of a gate are called the *fan-in* and *fan-out* of the gate, respectively. The fan-in 0 gates (*source* gates) are labeled from $x_1, x_2, \ldots, x_n$, or from the constants 0 or 1. Each non-source gate is labeled with a function name, such as AND, OR, or NOT. Fan-out 0 gates may or may not be labeled as output gates. Given an assignment of Boolean values to variables $x_1, x_2, \ldots, x_n$, each gate in the circuit can be assigned a value by first assigning all source vertices the value matching the labeled constant or labeled variable value, and subsequently assigning each gate the value computed by its labeled function on the values of its children. Given a fixed ordering on the output gates, the sequence of bits assigned to the output gates denotes the value computed by the circuit on the given input.

The *depth* of a circuit is the longest path from a source vertex to an output vertex. A circuit is a called a *formula* if all non-source vertices have fan-out 1, and there is exactly 1 output gate. Here, we focus on circuits that consist of AND, OR, NOT, and MAJORITY gates with arbitrary fan-in. We refer to circuits that use these gates as *Threshold Circuits* (TC).

Notation. When discussing a Boolean circuit, the following variables are used to denote the properties of the circuit: G denotes the number of gates in the circuit, D the circuit's depth, and F_{out} the maximum fan-out of any gate in the circuit.

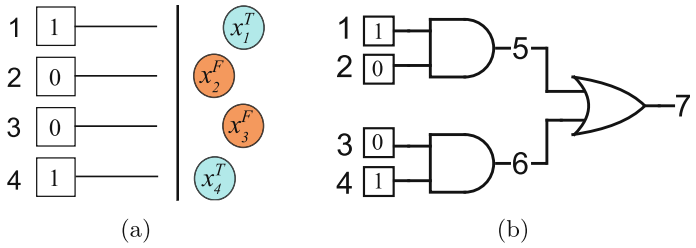


Fig. 2. (a) The input bits of a threshold formula and their representation as species.(b) An indexed threshold formula with the input species shown in Fig. 2a.

3 Computation of Threshold Formulas with (2, 0) Rules

Section 3.1 and Sect. 3.2 introduce how a step CRN using only (2,0) rules represents bits and logic gates of a Threshold Formula (TFs), respectively. An example construction of a formula is provided in Sect. 3.3. Section 3.4 then shows the general construction of building TFs, and we prove how the system computes TFs with $O(G)$ species, $O(D)$ steps, and $O(G)$ volume in Theorem 1.

3.1 Bit Representation

Here, we show how the bits of a TF are represented in our model. We first demonstrate a system for indexing the TF's wires before introducing the species used to represent bits.

Indexing. Every wire of the TF has a unique numerical index. The input and output bits that traverse these wires also use the wire's index. If a wire fans out, the fanned-out wires share the same index as the original. This indexing ensures that the bit species only use the rules that compute their respective gates. Note that gates may also be denoted with an index, where its index is that of its output wire.

Bits. Every input bit of a binary gate is represented by the species x_n^b, where $n \in \mathbb{N}$ and $b \in \{T, F\}$. n represents the bit's index and b represents its truth value. An example of these species is shown in Fig. 2a. Every output bit of a binary gate is represented by the species y_n^b or $y_{j\to i}^b$, where j represents the input bit's (x^b) index and i represents the output bit's (y^b) index.

3.2 Logic Gate Computation

We now show how the logic gates of a TF are computed. Let f_i^{in} be the set of all the indexes of the inputs fanning into a gate at index i.

AND Gate. To compute an AND gate such as the one shown in Fig. 3a, a single true output species y_i^T and $|f_i^{in}|$ copies of the false output species $y_{j\to i}^F$ are added in. In one step, the true input and false output species delete each other. Additionally, if at least one false input species exists, it deletes the lone true

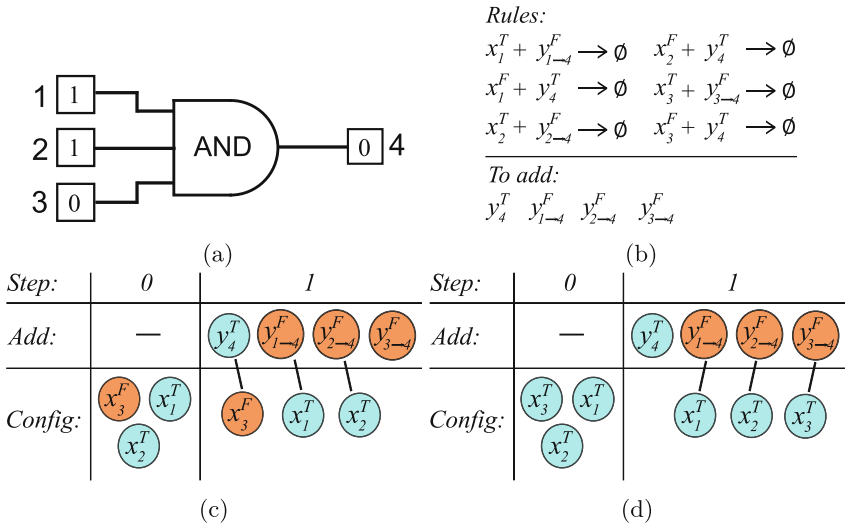


Fig. 3. (a) A threshold formula consisting of a single three-input AND gate. (b) Reaction rules and added species for the step CRN that compute the formula in Fig. 3a. (c) The step CRN computing the formula in Fig. 3a. The black lines connecting species represents a reaction applied to them. (d) The step CRN computing the circuit in Fig. 3a, but with three true inputs.

output species along with itself, guaranteeing a false output as shown in the example steps in Fig. 3c. The only output species remaining after the step are those whose truth value matches the intended output.

AND Gate Example. Consider an AND gate with index 4 and a fan-in of three; the first two inputs are true and the last is false. For computing this gate with our model, the system's initial configuration consists of x_1^T, x_2^T, and x_3^F. We then add y_4^T, representing a true output and $y_{1\to4}^F$, $y_{2\to4}^F$, and $y_{3\to4}^F$, representing false outputs. The rules $x_1^T + y_{1\to4}^F \to \emptyset$, $x_2^T + y_{2\to4}^F \to \emptyset$, and $x_3^F + x_1^T \to \emptyset$ are then applied to the system, removing all reactant species in these rules. The remaining species is $y_{3\to4}^F$, indicating a false output.

OR Gate. To compute an OR gate, a single false output species y_i^F and $|f_i^{in}|$ copies of the true output species $y_{j\to i}^T$ are added in. In one step, the false input and true output species delete each other. Additionally, if at least one true input species exists, it deletes the lone false output species along with itself, guaranteeing a true output. The only output species remaining after the step are those whose truth value matches the intended output.

NOT Gate. To compute a NOT gate, a single copy of the true and false output species are added in. In a single step, the input and output species that share the same truth value b delete each other, leaving the complement of the input species as the remaining output species (Table 2).

Table 2. (2, 0) rules and steps for computing AND, OR, and NOT gates.

Gate Type	Step	Relevant Rules	Description
AND	*Add* y_i^T $\forall j \in f_i^{in}$: $y_{j\to i}^F$	$x_j^T + y_{j\to i}^F \to \emptyset$ $x_j^F + y_i^T \to \emptyset$	An input species with a certain truth value deletes the complement output species
OR	*Add* y_i^F $\forall j \in f_i^{in}$: $y_{j\to i}^T$	$x_j^T + y_i^F \to \emptyset$ $x_j^F + y_{j\to i}^T \to \emptyset$	An input species with a certain truth value deletes the complement output species
NOT	*Add* y_i^T y_i^F	$x_j^T + y_i^T \to \emptyset$ $x_j^F + y_i^F \to \emptyset$	The input and output species that share the same truth value delete each other

Table 3. (2, 0) rules and steps for computing majority gates.

Steps		Relevant Rules	Description
1	*Add* $\lvert f_i^{in}\rvert \cdot a_i^T$ $\lvert f_i^{in}\rvert \cdot a_i^F$	$\forall j \in f_i^{in}$: $x_j^T + a_i^F \to \emptyset$ $x_j^F + a_i^T \to \emptyset$	$\forall j \in f_i^{in}$, convert x_j^b input species into a_i^b species
2	*Add* $\lfloor \lvert f_i^{in}\rvert /2 \rfloor \cdot b_i^T$ $\lfloor \lvert f_i^{in}\rvert /2 \rfloor \cdot b_i^F$	$a_i^T + b_i^F \to \emptyset$ $a_i^F + b_i^T \to \emptyset$	Adding $\lfloor \lvert f_i^{in}\rvert /2 \rfloor$ amounts of b_i^T and b_i^F species will delete all of the minority species, leaving some amount of the majority species remaining
3	*Add* y_i^T y_i^F	$a_i^T + y_i^F \to \emptyset$ $a_i^F + y_i^T \to \emptyset$	Convert a_i^b into the proper output species (y_i^b)

Majority Gate. To compute a majority gate, all input species are first converted into a new species a_i^b (Step 1). These species retain the same index and truth value of their original inputs. If the number of bits inputted into a majority gate is even, then an extra *false* input species should be added in. The species b_i^b is then added (Step 2). This species performs the majority operation across all a_i^b species. Any species that represents the minority inputs are deleted. The remaining species are then converted into the matching output species (Step 3) (Table 3).

3.3 Formula Computation Example

We demonstrate a simple example of computing a threshold formula under our constructions. The formula is the same as in Fig. 2b. It has four inputs: x_1, x_2, x_3, and x_4. At the first depth level, x_1 and x_2 fan into an AND gate, as does

x_3 and x_4. Both gate outputs are then fanned into an OR gate, whose output represents the final value of the formula.

The initial configuration consists of bit species that correlate to the input values for the formula. Step 1 converts the species into input species for the first depth level. Step 2 then performs all gate operations at the first depth level. Step 3 converts the output species of the gates into input species for the next and final depth level. Step 4 computes the last gate. Finally, Step 5 coverts the gate;s output into an input species that represents the final output of the formula. A more detailed explanation of computing the formula is in Table 4a.

3.4 Threshold Formula Computation

We now introduce another step with rules that convert the output species of one depth level into input species for the next level, enabling the complete computation of a TF. We then prove the computational complexity of computing TFs within our system.

Depth Traversal. To enable the traversal of every gates' bits at a specific depth level to the next level, every output species is converted into an input species in one step. The same truth value and index is retained between the output and input species. Table 4b shows how to compute depth level traversal for an output bit with index i.

Table 4. (a) (2, 0) rules and steps for computing the formula in Fig. 2b. (b) (2, 0) rules and step for converting outputs to inputs per depth level. Add species for that represent true and false inputs. Delete the species that are the complement of the output. Only the correct input species remains.

Initial Configuration: $y_1^T, y_2^F, y_3^F, y_4^T$			
Step			**Relevant Rules**
1	*Add*	$x_1^T, x_2^T, x_3^T, x_4^T$ $x_1^F, x_2^F, x_3^F, x_4^F$	$y_1^T + x_1^F \rightarrow \emptyset$ $y_2^F + x_2^T \rightarrow \emptyset$ $y_3^F + x_3^T \rightarrow \emptyset$ $y_4^T + x_4^F \rightarrow \emptyset$
2	*Add*	$y_5^T, y_{1\rightarrow 5}^F, y_{2\rightarrow 5}^F$ $y_6^T, y_{3\rightarrow 6}^F, y_{4\rightarrow 6}^F$	$x_1^T + y_{1\rightarrow 5}^F \rightarrow \emptyset$ $x_2^F + y_5^T \rightarrow \emptyset$ $x_3^F + y_6^T \rightarrow \emptyset$ $x_4^T + y_{4\rightarrow 6}^F \rightarrow \emptyset$
3	*Add*	x_5^T, x_6^T x_5^F, x_6^F	$y_{2\rightarrow 5}^F + x_5^T \rightarrow \emptyset$ $y_{3\rightarrow 6}^F + x_6^T \rightarrow \emptyset$
4	*Add*	$y_{5\rightarrow 7}^T, y_{6\rightarrow 7}^T, y_7^F$	$y_{5\rightarrow 7}^F + x_5^T \rightarrow \emptyset$ $y_{6\rightarrow 7}^F + x_6^T \rightarrow \emptyset$
5	*Add*	x_7^T, x_7^F	$y_7^F + x_7^T \rightarrow \emptyset$

(a)

Step		**Relevant Rules**
Add	x_i^T x_i^F	$y_i^T + x_i^F \rightarrow \emptyset$ $y_i^F + x_i^T \rightarrow \emptyset$ $y_{j\rightarrow i}^T + x_i^F \rightarrow \emptyset$ $y_{j\rightarrow i}^F + x_i^T \rightarrow \emptyset$

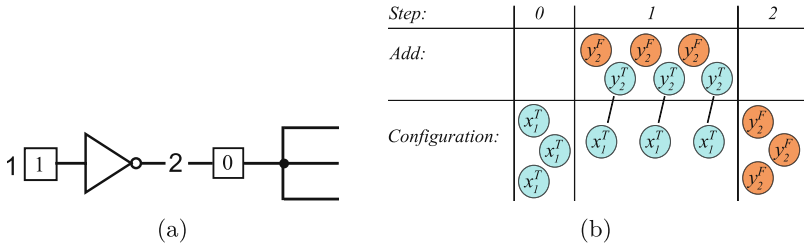
(b)

Theorem 1. *Threshold formulas (TF) can be computed with multiple-input relaxation by a step CRN with only* $(2, 0)$ *rules with upper bounds of* $O(G)$ *species,* $O(D)$ *steps, and* $O(G)$ *volume.*

Proof. Each gate of a TF is represented by a constant number of species, resulting in $O(G)$ unique species. All gates at a given depth level are computed simultaneously in constant steps. Computing a TF therefore requires $O(D)$ steps.

It is possible not all species that are no longer needed after computing a specific gate are deleted. For example, computing an AND gate with three false inputs leaves two of those species in the configuration. While this does not cause computation errors, the volume will increase. Therefore, it is possible for only a fraction of the $O(G)$ species added throughout the simulation to be deleted, resulting in $O(G)$ volume. □

4 Computation of TCs with Exponential Volume



(a) (b)

Fig. 4. (a) A NOT gate with a fan-out of three. (b) Computing the NOT gate in Fig. 4a in (2, 0) rules.

In this section, we slightly alter the approach presented in Sect. 3 to enable computation of Threshold Circuits (TC). We show in Sect. 4.1 how modifying our volume to be exponentially-sized allows the system to account for unbounded fan-out outside of the first depth level, enabling the computation of TCs. Theorem 2 shows our system computes TCs with $O(G)$ species, $O(D)$ steps, and $O({GF_{out}}^{D})$ volume.

4.1 Threshold Circuit Computation

Here, we demonstrate how to account for unbounded fan-out when computing TCs, and show the computational complexity of computing TCs with our system.

Bits and Gates. Section 3.1 shows how input bits, output bits, and indexing are represented. Individual gates and depth traversal are computed using the same methods shown in Sects. 3.2 and 3.4, respectively.

Unbounded Fan-Out. To allow the output of a gate at index i to fan-out $k > 1$, such as in the NOT gate shown in Fig. 4a, the count of the species added for all

the gates whose output eventually fans into gate i should be multiplied by k. The gate computes all input species concurrently with each other, and result in the gate's output species being equivalent in quantity to the fan-out. Figure 4b shows an example of this process for a NOT gate with a fan-out of three.

Unbounded Fan-Out Example. Consider an AND gate with a fan-in and fan-out of two. Let the two input bits be true and false. To compute the gate with the correct amount of fan-out, our system's initial configuration consists of two copies of each input species (x_1^T, x_1^T, x_2^F, and x_2^F). We then apply the relevant rules to the configuration. Afterwards, we are left with two copies of $y_{2\to1}^F$.

Theorem 2. *Threshold circuits (TC) can be computed with multiple-input relaxation by a step CRN with only (2, 0) rules with upper bounds of $O(G)$ species, $O(D)$ steps, and $O({GF_{out}}^D)$ volume.*

5 Exponential Volume Lower Bound for Gate-Wise Simulation

In this section, we derive an exponential lower bound for the volume of a step CRN with $(2, 0)$ rules that simulates boolean circuits of depth D. Our lower bound almost matches the upper bound.

To prove the lower bound, we design a circuit that is able to be simulated by any step CRN using only $(2, 0)$ rules as follows. The circuit has D stages such that each stage of the circuit has $O(1)$ layers. We establish a recursive inequality for the CRN volume over three consecutive stages which implies an exponential lower bound for species in the input stage. We show a proof that the lower bound of volume in a step CRN that uses gate-wise simulation to simulate a boolean circuit with only $(2, 0)$ rules is $2^{\Omega(D)}$.

Definition 5. *A step CRN uses* gate-wise simulation *to simulate a circuit $V(\cdot)$ if each gate g is assigned $c_{1,g}$ copies of species 1_g, which represents output 1, and $c_{0,g}$ copies of 0_g, which represents output 0. When gate g computes an output, it will be either $c_{1,g}$ copies of species 1_g for the case 1, or $c_{0,g}$ copies of species 0_g for the case 0. We define $C(g) = c_{1,g} + c_{0,g}$ to be the number of species used for gate g. There is a special case that g is one of the input bits (source gates with fan-in 0) that satisfies $c_{1,g} = 0$ or $c_{0,g} = 0$, as each input bit is either 0 or 1. Let a step CRN have k steps $0, 1, \cdots, k-1$ as defined in Sect. 2.4. It also satisfies the conditions:*

- *Every gate g enters its complete state at exactly one step i, which is denoted by $complete(g) = i$. After step i, the system releases $c_{a,g}$ copies of species a_g and removes all copies of species $(1-a)_g$ to represent gate g having the output a. After step $complete(g)$, the system does not generate any additional copy of 1_g or 0_g (it may keep some existing copies of a_g). The output of gate g determines the simulation according to the logic of circuit.*
- *For two different gates g_1 and g_2, if there is a directed path from g_1 to g_2 (g_1's output may affect g_2's output) in the circuit $V(\cdot)$, then $complete(g_1) < complete(g_2)$.*

If a circuit $V(\cdot)$ computes a function $f(x_1, \cdots, x_n) = y_1, \cdots, y_m$ ($\{0,1\}^n \to \{0,1\}^m$) and $V(\cdot)$ is simulated using gate-wise simulation in a step CRN, define $1_{f(\cdot),i}$ to represent the case y_i to be 1 and $0_{f(\cdot),i}$ to represent the case y_i to be 0.

By Definition 5, when gate g outputs 1, all the $c_{0,g}$ copies of 0_g are removed and it has $c_{1,g}$ copies of 1_g to enter the next layer of a circuit. The step CRN given in Sect. 4 uses gate-wise simulation to simulate threshold circuits. Our lower bound result in this section shows that the exponential volume is required.

Lemma 1. *Assume that a step CRN with $(2,0)$ rules simulates a circuit $V(\cdot)$. Let $b \in \{1_g, 0_g\}$ be the output species. If one copy of a species is removed or added, it results in at most one difference in the number of copies of species b.*

Lemma 2. *Let $f(x_1, \cdots, x_n) = y_1, \cdots, y_m$ be a function $\{0,1\}^n \to \{0,1\}^m$ such that each variable x_i and y_j has a value in $\{0,1\}$. If a step CRN with $(2,0)$ rules computes $f(\cdot)$, and changing variable x_j to $1 - x_j$ will change $y_{i_1}, \cdots, y_{i_t}$ to $1 - y_{i_1}, \cdots, 1 - y_{i_t}$, respectively, then $C(x_j) \geq \sum_{k=1}^{t} C(y_{i_k})$.*

Definition 6. *A list of Boolean circuits $\{H_n(\cdot)\}$ is uniform if there is a Turing machine $M(\cdot)$ such that each $H_n(\cdot)$ can be generated by $M(1^n)$ in a polynomial $p(n)$ steps.*

Theorem 3. *There exist uniform Boolean circuits $\{V_D(x_1, x_2, x_3)\}_{D=1}^{+\infty}$ with each $V_D(x_1, x_2, x_3) : \{0,1\}^3 \to \{0,1\}^3$ s.t. each $V_D(x_1, x_2, x_3)$ has depth $O(D)$, size $O(D)$, 3 output bits, and requires $C(g) = 2^{\Omega(D)}$ for at least one input gate g in a step CRN using gate-wise simulation to simulate $V_D(\cdot)$ with $(2,0)$ rules.*

Proof. We construct a circuit that has $O(D)$ layers. It is built in D stages. Each stage has a circuit of depth $O(1)$ to compute function $s(x_1, x_2, x_3) = y_1 y_2 y_3$. The function $s(\cdot)$ has the properties:

$$s(1,1,1) = 111, \ (1) \qquad s(0,1,1) = 000, \ (2) \qquad s(1,0,1) = 011, \ (3)$$
$$s(1,1,0) = 101, \ (4) \qquad s(0,0,0) = 110 \ (5)$$

Define function $s^{(1)}(x_1, x_2, x_3) = s(x_1, x_2, x_3)$ and $s^{(k+1)}(x_1, x_2, x_3) = s(s^{(k)}(x_1, x_2, x_3))$ for all integers $k > 1$. The circuit $V(x_1, x_2, x_3) = s^{(D)}(x_1, x_2, x_3)$. We can also represent the circuit $V(x_1, x_2, x_3) = s^{(D)}(x_1, x_2, x_3) = s_{D-1} \circ s_{D-2} \circ \cdots \circ s_0(x_1, x_2, x_3)$, where $s_i(\cdot)$ represent the function $s(\cdot)$ at stage i. The circuit $V(x_1, x_2, x_3)$, which computes $s^{(D)}(x_1, x_2, x_3)$ links the D circuits $V_s(x_1, x_2, x_3)$ that compute the function $s(x_1, x_2, x_3)$. The three output bits for $s_k(\cdot)$ at stage k become three input bits of $s_{k-1}(\cdot)$ at stage $k-1$.

Let the output of the circuit be stage 0. The input stage has the largest stage index. Consider the general case. Let $C_k(u)$ be the number of copies of species u in stage k. If u is computed by a gate g, we let $C_k(u) = C(g)$. Let $v_{i,k}$ be the variable v_i at stage k. As we have three output bits $y_{1,0}, y_{2,0}, y_{3,0}$ in the last layer (layer D), each bit $y_{i,0}$ must have a copy of species to represent its 0, 1-value (see Definition 5). Thus,

$$C_D(y_{1,0}) \geq 1,\ C_D(y_{2,0}) \geq 1,\ C_D(y_{3,0}) \geq 1. \tag{6}$$

When $x_1x_2x_3$ is changed from 111 to 011 (by flipping x_1), the output $y_1y_2y_3$ is changed from 111 to 000. By Equations (1) and (2) and Lemma 2, we have

$$C_k(x_{1,k}) \geq C_k(y_{1,k}) + C_k(y_{2,k}) + C_k(y_{3,k}). \tag{7}$$

When $x_1x_2x_3$ is changed from 111 to 101 (by flipping x_2), the output $y_1y_2y_3$ is changed from 111 to 011. By Equations (1) and (3) and Lemma 2, we have Equation 8 below. When $x_1x_2x_3$ is changed from 111 to 110 (by flipping x_3), the output $y_1y_2y_3$ is changed from 111 to 101. By Equations (1) and (4) and Lemma 2, we have Equation 9 below.

$$C_k(x_{2,k}) \geq C_k(y_{1,k}) \ (8) \qquad C_k(x_{3,k}) \geq C_k(y_{2,k}) \ (9)$$

When $y_{i,k}$ is equal to $x_{i,k-1}$ as the output of $s_k(\cdot)$ becomes the input of $s_{k-1}(\cdot)$. We have

$$C_k(y_{i,k}) \geq C_{k-1}(x_{i,k-1}) \quad for\ i = 1, 2, 3. \tag{10}$$

In each stage, the input to the function $s(\cdot)$ can reach all cases 000, 011, 101, 110,111 by adjusting the 3 input bits of the circuit. When the input is 111, the function $s(\cdot)$ gives the same output 111 at all phases. Through a simple repetition of the above inequalities, we derive a $2^{\Omega(D)}$ volume lower bound.

$$\begin{aligned} C_k(x_{1,k}) &\geq C_{k-1}(x_{1,k-1}) + C_{k-1}(x_{2,k-1}) + C_{k-1}(x_{3,k-1}) (by\ (7), (10)) && (11)\\ &\geq C_{k-1}(x_{1,k-1}) + C_{k-1}(y_{1,k-1}) \quad (by\ inequality\ (8)) && (12)\\ &\geq C_{k-1}(x_{1,k-1}) + C_{k-2}(x_{1,k-2}). \quad (by\ inequality\ (10)) && (13) \end{aligned}$$

Let $a_0, a_1, \cdots$ be the Fibonacci series with $a_0 = a_1 = 1$ and recursion $a_k = a_{k-1} + a_{k-2}$ for all $k > 1$. By inequalities (6), (7), and the fact that every input bit affects the output bit in $s(\cdot)$, we have $C_0(x_{1,0}) \geq 1$ and $C_1(x_{1,1}) \geq 1$. This is because when the three input bits are 111, we need bit x_1 to make the output bits 111. By inequality (13), we have $C_k(x_{1,k}) \geq a_k$ for all $k \geq 0$. □

6 Conclusions and Open Problems

In this paper we show how bimolecular void rules, a subset of reaction rules with low power compared to traditional CRNs, become capable of computing threshold formulas and circuits in the step CRN model under gate-wise simulation. We also prove that simulating circuits under this technique requires an exponential lower bound volume that matches the upper bound of our construction methods.

These results naturally lead to some promising future research directions. One approach is constructing another method for simulating threshold circuits under

only (2,0) rules. A more general simulation technique could have the benefit of computing circuits without the exponential-sized volume gate-wise simulation requires. Furthermore, our step CRN definition requires the system to reach a terminal configuration before moving to the next step. Relaxing this definition so that a system may reach a step without entering a terminal configuration can make the model more valuable to general CRNs, where reachability to a terminal configuration is not guaranteed.

References

1. Alaniz, R.M., et al.: Reachability in restricted chemical reaction networks. arXiv preprint arXiv:2211.12603 (2022)
2. Anderson, R., et al.: Computing threshold circuits with void reactions in step chemical reaction networks. arXiv preprint arXiv:2402.08220 (2024)
3. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distrib. Comput. **18**(4), 235–253 (2006)
4. Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. Distrib. Comput. **21**, 87–102 (2008)
5. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. (2007)
6. Aris, R.: Prolegomena to the rational analysis of systems of chemical reactions. Ration. Mech. Anal. **19**(2), 81–99 (1965)
7. Aris, R.: Prolegomena to the rational analysis of systems of chemical reactions II. Some addenda. Ration. Mech. Anal. **27**(5), 356–364 (1968)
8. Arkin, A., Ross, J.: Computational functions in biochemical reaction networks. Biophys. J . **67**(2), 560–578 (1994)
9. Beiki, Z., Dorabi, Z.Z., Jahanian, A.: Real parallel and constant delay logic circuit design methodology based on the dna model-of-computation. Microprocess. Microsyst. **61**, 217–226 (2018)
10. Cardelli, L., Csikász-Nagy, A.: The cell cycle switch computes approximate majority. Sci. Rep. **2**(1), 656 (2012)
11. Cardelli, L., Kwiatkowska, M., Whitby, M.: Chemical reaction network designs for asynchronous logic circuits. Nat. Comput. **17**, 109–130 (2018)
12. Chen, H.L., Doty, D., Soloveichik, D.: Deterministic function computation with chemical reaction networks. Nat. Comput. **13**(4), 517–534 (2014)
13. Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Algorithmic Bioprocesses, pp. 543–584 (2009)
14. Dalchau, N., Chandran, H., Gopalkrishnan, N., Phillips, A., Reif, J.: Probabilistic analysis of localized DNA hybridization circuits. ACS Synth. Biol. **4**(8), 898–913 (2015)
15. Ellis, S.J., Klinge, T.H., Lathrop, J.I.: Robust chemical circuits. Biosystems **186**, 103983 (2019)
16. Eshra, A., El-Sayed, A.: An odd parity checker prototype using dnazyme finite state machine. IEEE/ACM Trans. Comput. Biol. Bioinf. **11**(2), 316–324 (2013)
17. Fan, D., Fan, Y., Wang, E., Dong, S.: A simple, label-free, electrochemical DNA parity generator/checker for error detection during data transmission based on "aptamer-nanoclaw"-modulated protein steric hindrance. Chem. Sci. **9**(34), 6981–6987 (2018). https://doi.org/10.1039/C8SC02482K

18. Fan, D., Wang, J., Han, J., Wang, E., Dong, S.: Engineering DNA logic systems with non-canonical DNA-nanostructures: basic principles, recent developments and bio-applications. Sci. China Chem. **65**(2), 284–297 (2022)
19. Jiang, H., Riedel, M.D., Parhi, K.K.: Digital logic with molecular reactions. In: International Confernce on Computer-Aided Design (ICCAD) (ICCAD 2013), pp. 721–727 (2013)
20. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
21. Lin, Y.C., Jiang, J.H.R.: Mining biochemical circuits from enzyme databases via Boolean reasoning. In: 39th International Conference on Computer-Aided Design, pp. 1–9 (2020)
22. Magri, D.C.: A fluorescent and logic gate driven by electrons and protons. New J. Chem. **33**(3), 457–461 (2009)
23. Mailloux, S., Guz, N., Zakharchenko, A., Minko, S., Katz, E.: Majority and minority gates realized in enzyme-biocatalyzed systems integrated with logic networks and interfaced with bioelectronic systems. J. Phys. Chem. B **118**(24), 6775–6784 (2014). https://doi.org/10.1021/jp504057u
24. Petri, C.A.: Kommunikation mit Automaten. Ph.D. thesis, Rheinisch-Westfälischen Institutes für Instrumentelle Mathematik an der Universität Bonn (1962)
25. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science **332**(6034), 1196–1201 (2011)
26. Qian, L., Winfree, E.: A simple dna gate motif for synthesizing large-scale circuits. J. R. Soc. Interface **8**(62), 1281–1297 (2011)
27. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Nat. Comput. **7**(4), 615–633 (2008)
28. Thachuk, C., Winfree, E., Soloveichik, D.: Leakless DNA strand displacement systems. In: 21st International Conference on DNA Computing and Molecular Programming (DNA 2015), pp. 133–153. Springer (2015)
29. Winfree, E.: Chemical reaction networks and stochastic local search. In: 25th International Conference on DNA Computing and Molecular Programming (DNA 2019), pp. 1–20 (2019)
30. Xiao, W., Zhang, X., Zhang, Z., Chen, C., Shi, X.: Molecular full adder based on DNA strand displacement. IEEE Access **8**, 189796–189801 (2020)