CRNs Exposed: Systematic Exploration of Chemical Reaction Networks

Marko Vasic, David Soloveichik, and Sarfraz Khurshid

University of Texas at Austin, Austin TX 78712, USA
{vasic,david.soloveichik,khurshid}@utexas.edu

Abstract. Formal methods have enabled breakthroughs in many fields, such as in hardware verification, machine learning and biological systems. Our focus is on systems and synthetic biology, where a key object of interest is coupled chemical reactions in a well-mixed solution formalized as chemical reaction networks (CRNs). CRNs are pivotal for our understanding of biological regulatory and metabolic networks, as well as for programming engineered molecular behavior. Although it is clear that small CRNs are capable of complex dynamics and computational behavior, it remains difficult to explore the space of CRNs in search for desired functionality. We use Alloy, a tool for expressing structural constraints and behavior in software systems, to enumerate CRNs with declaratively specified properties. We show how this framework can enumerate CRNs with a variety of structural constraints including biologically motivated catalytic networks and metabolic networks, and see-saw networks motivated by DNA nanotechnology. We also use the framework to explore analog function computation in rate-independent CRNs. By computing the desired output value with stoichiometry rather than with reaction rates (in the sense that $X \rightarrow Y + Y$ computes multiplication by 2), such CRNs are completely robust to the choice of reaction rates or rate law. We find the smallest CRNs computing the max, abs, and ReLU (rectified linear unit) functions in a natural subclass of rate-independent CRNs where rate-independence follows from structural network properties.

1 Introduction

Formal methods have enabled breakthroughs in many fields, e.g., in hardware verification [13], machine learning [20\cite{29}], and biological systems [5\cite{21}, 26\cite{36}, 52\cite{36}]. In this paper we apply formal methods to Chemical Reaction Networks (CRNs), which have been objects of intense study in systems and synthetic biology. CRNs are widely used in modeling biological regulatory networks, and essentially identical models are also widely used in ecology [51\cite{51}], distributed computing [2\cite{2}], and other fields. More recently, CRNs have been directly used as a programming language for engineering molecules obeying prescribed interaction rules via DNA strand displacement cascades [11\cite{11}, 47\cite{47}, 49\cite{49}].

It is clear that small CRNs can exhibit very complex behavior. Dynamical systems, e.g., oscillatory, chaotic, and bistable systems, typically contain only a few reactions. Small CRNs also exhibit interesting computational behavior. For example, the approximate majority population protocol studied in distributed computing [1\cite{1}] was later identified with a variety of biological regulatory networks [6\cite{6}]. Can we systematically explore the power of small reaction networks?
We present a method that exhaustively enumerates small CRNs in different classes that are relevant for biology and for synthetic engineering systems. The enumeration is performed using Alloy, a powerful tool for modeling structural constraints and behavior in software systems using first-order logic with transitive closure [30]. The Alloy tool performs scope-bounded analysis [32]. Given an Alloy model and a scope, i.e., a bound on the universe of discourse, the analyzer translates the Alloy model to a propositional satisfiability (SAT) formula and invokes an off-the-shelf SAT solver [17] to analyze the model. Alloy is used in a wide range of areas in software engineering, including software design [18,31], analysis [16,19,33,35], testing [40], and security [34]. We show how Alloy can be used to conveniently model interesting classes of CRNs for biology and bioengineering, and we use the Alloy analyzer to search for CRNs with specific desired functionality.

As examples of the method we first focus on a number of classes: elementary, catalytic, metabolic. We say elementary reactions are CRNs with at most two reactants and products. (We allow reactions to be irreversible; reversible reactions are represented by two irreversible reactions.) Catalytic networks are those elementary CRNs in which the reactants and products are not disjoint; i.e., the reaction is catalyzed by some species that is not consumed in the reaction. Catalytic networks (e.g., transcriptional, phosphorylation, etc.) regulate many aspects of the cell’s behavior [38,43]. In general protein-protein interactions, proteins can catalytically modify other proteins, which in turn can be catalysts in other interactions. An important subclass of catalytic networks are metabolic networks, where the enzymes are proteins while the substrates are small molecules; these catalytic CRNs are “bi-partite” in the sense that a species is either always a catalyst or never a catalyst.

We then turn our attention to classes of CRNs especially relevant for synthetic reaction networks, showing how abstract molecular structure can be modeled in Alloy. In particular, we focus on DNA strand displacement cascades, which have proved to be a uniquely programmable technology for cell-free DNA-only systems [55]. Strand displacement interactions correspond to reactions between two types of molecules: “gates” and “strands”, where the reacting strand displaces the strand previously sequestered in the gate complex. A simple, yet very scalable, class of strand displacement circuits uses a simple motif called seesaw gates [12,44,45] that makes use of a reversible strand displacement reaction. We designed an Alloy program to enumerate such strand displacement reactions, showing that abstract molecular structure can be incorporated into the Alloy modeling formalism.

In the second part of the paper, we use our enumeration framework to search for specific desired functionality in a class of CRNs. In particular, we focus on the class of rate-independent CRNs [10]. Consider the reaction $X \rightarrow Y + Y$, and think of the concentrations of species $X$ and $Y$ as input and output respectively. This reaction computes the function of “multiplication by 2” since in the limit of time going to infinity it produces two units of $Y$ for every unit of $X$ initially present. Similarly the reaction $X_1 + X_2 \rightarrow Y$ computes the “minimum” function since
\[ A \rightarrow Z_1 + Y \]
\[ B \rightarrow Z_2 + Y \]
\[ Z_1 + Z_2 \rightarrow K \]
\[ Y + K \rightarrow \emptyset \]

Fig. 1: CRN computing Max. We think of the initial amount of \( A \) and \( B \) as inputs, and the converging amount of \( Y \) as the output. The amount of \( Y \) eventually produced in reactions 1 and 2 is the sum of the initial amounts of \( A \) and \( B \). The amount of \( K \) eventually produced in reaction 3 is the minimum of the initial amounts of \( A \) and \( B \). Reaction 4 subtracts the minimum from the sum, yielding the maximum. (The 4th reaction generates waste species, which are not named.)

The amount of \( Y \) eventually produced will be the minimum of the initial amounts of \( X_1 \) and \( X_2 \). Note that such computation makes no assumption on the rate law, such as whether the reaction obeys mass-action kinetics\(^1\) or not, allowing the computation to be correct in a wide variety of chemical contexts. (We use the continuous CRN model where concentrations are real-valued quantities.)

A natural subclass of CRNs whose structure enforces rate independence are those that satisfy two constraints: feed-forward, and non-competitive\(^2\). Intuitively, the first condition ensures that the CRN converges to a static equilibrium where no reaction can occur. The second condition ensures that no matter what the rates are, the system converges to the same static equilibrium. More precisely, we define feed-forward as follows: there exists a total ordering on the reactions such that no reaction consumes\(^3\) a species produced by a reaction later in the ordering. We define non-competitive as follows: Every species is consumed by at most one reaction. Such constraints on the structure of the network can be easily encoded in the Alloy specification.

Focusing on the class of feed-forward, non-competitive CRNs, we search for the smallest reaction networks implementing \( \text{max} \), \( \text{abs} \), and \( \text{ReLU} \) (rectified linear unit) functions. As an example of the kind of computation we achieve, consider the \( \text{max} \) computing CRN shown in Fig. 1. This CRN was previously studied\(^9\)^{10}; our result shows that it is indeed the smallest. The maximum function serves an important role in rate-independent computation since together with minimum, multiplication and division by a constant it forms a complete basis set\(^8\)^{10}. To our knowledge, the smallest implementations of \( \text{abs} \), and \( \text{ReLU} \) that we find are novel and have not been previously published. The \( \text{ReLU} \) function was first introduced due to the biological motivations explaining functioning of neurons in the brain cortex\(^{24}\). Since then, it was used with a tremendous

\(^1\)"Mass-action" kinetics refers to the best-studied case where the reaction rate is proportional to the product of the concentration of the reactants.

\(^2\)Feed-forward and non-competitive conditions are sufficient for rate-independence, but are not necessary. However, most known examples of rate independent computation satisfy these conditions.

\(^3\)We say a reaction produces (resp. consumes) a species \( S \) if there is net stoichiometric gain (resp. loss) of \( S \). Thus a catalyst in a reaction is neither consumed nor produced.
success in the machine learning community, particularly in deep learning [22,37] for realizing artificial neural networks. The simplicity of its implementation suggests that CRNs can naturally realize neural computation.

Much ongoing work explores the computational power of CRNs. Previous work showed the implementation of numerous complex behaviors, such as mapping polynomials to chemical reactions [46], programming logic gates [39], mapping discrete, control flow, algorithms [28], and a molecular programming language translating high-level specifications to chemical reactions [50]. However, the complexity of these reaction systems can be infeasible, asking for novel techniques that answers what the natural way to compute “in reactions”. To help answer this question we can take a different, ‘bottom up’ approach, and explore the design space of reactions. We hope the bottom-up approach will give insight of how that high-level language should look like at the first place, and prevent potential incompatibility and complexity of translation from the language to chemical world. We believe that insight we get from exploring reactions will help in design of higher-level primitives that naturally map to reactions, and will provide knowledge for more efficient design of high-level languages.

2 Modeling CRNs in Alloy

This section describes our approach to modeling chemical reaction networks (CRNs) in Alloy. We first introduce a general model that can represent the most broad class of CRNs (allowing arbitrary number of reactants and products), and next show specializations of the model for different classes such as elementary, catalytic, metabolic, and feed-forward non-competitive reactions. In addition, we present models that encode molecular structure, such as strands and gates and more fine-grained seesaw networks model. Our approach naturally admits a hierarchical structuring of models where a model builds on and specializes another model, e.g., metabolic reactions are structurally more constrained set of reactions than the elementary. This allows a systematic exploration of the design space of models as this section illustrates.

General model. Our general model captures CRNs consisting of reactions with arbitrary number of reactants and products. To model this in Alloy we define a set of species, a set of reactions, two relations that characterize the reactants and products, and logical constraints that define the basic structural requirements for well-formed CRNs. Figure 2 specifies the general model in Alloy. The keyword module allows naming the model, which can be imported in other models. The keyword sig declares a basic type and introduces a set of indivisible atoms that do not have any internal structure. The model declares two sets: a set of species (Species) and a set of reactions (Reaction). The sig declaration of Reaction introduces two fields, reactants and products, each of type sequence (seq) of Species. Alloy models a sequence as a binary relation from (non-negative) integer indices to atoms. Thus, each of these field declarations introduces a ternary relation of type: Reaction × Int × Species. In a case of reaction R0 : X → Y + Y, the value of products relation would be the set: \{R0 × 0 × Y, R0 × 1 × Y\}. Note that we model reactants and products with
module crn

abstract sig Species {}
abstract sig Reaction { reactants, products: seq Species }

-- Basic semantic constraints -- for all CRNs
fact AtLeastOneReactant { -- each reaction has >=1 reactant
    all r: Reaction | some r.reactants }

fact UniqueReactions { -- each reaction is unique
    all disj r1, r2 : Reaction | ReactionsDifferent[r1, r2] }

pred ReactionsDifferent[r1, r2: Reaction] {
    SpeciesSeqDifferent[r1.reactants, r2.reactants]
    or SpeciesSeqDifferent[r1.products, r2.products] }

pred SpeciesSeqDifferent[seq1, seq2: seq Species] {
    some s : Species | #indsOf[seq1, s] != #indsOf[seq2, s] }

fact ReactantsDifferentThanProducts {
    all r: Reaction | SpeciesSeqDifferent[r.reactants, r.products] }

fact AllSpeciesUsed { -- each species is used in some reaction
    Int.(Reaction.(reactants + products)) = Species }
module elementary
open crn
pred Elementary() { MaxReactantsNum[2] and MaxProductsNum[2] }
pred MaxReactantsNum[num: Int] { all r: Reaction | lte[#r.reactants, num] }
pred MaxProductsNum[num: Int] { all r: Reaction | lte[#r.products, num] }

Fig. 3: Elementary reactions.

The predicate SpeciesSeqDifferent is true if the two sequences of species are different. It uses existential quantification (some). The operator ‘#’ represents set cardinality. The Alloy library function indsOf represents the set of indices where the atom argument (e.g., s) appears in the sequence argument (e.g., seq1). Intuitively, this predicate compares number of appearances of species in two sequences, and returns true if exists a species that appears different number of times in the two sequences.

The fact ReactantsDifferentThanProducts requires each reaction to have non-identical reactants and products. Finally, the fact AllSpeciesUsed states that all species must be a part of some reaction, enforcing that instances do not have isolated species which is more meaningful. Int represents the set of integers.

Illustrating General Model. To illustrate using the Alloy analyzer, consider generating an instance of the constraints modeled. The following run command, labeled Generate, instructs the analyzer to create an instance with respect to a universe that contains exactly 2 reactions and 2 species, and 2-bit integers, and conforms to all the facts in the model:

Generate: run {} for exactly 2 Reaction, exactly 2 Species, 2 int

Executing the command Generate and enumerating the first three instances creates the following CRNs where S0 and S1 are species, and ∅ are waste species:

\[
\begin{align*}
S_1 &\rightarrow S_0 \\
S_0 &\rightarrow S_1 \\
S_1 &\rightarrow ∅ \\
S_0 &\rightarrow S_1 \\
\end{align*}
\]

While quite small, these three instances exhibit interesting properties, CRN in (a) models a reversible reaction \(S_1 \leftrightarrow S_0\); CRN in (b) is rate-dependent, where amount of \(S_1\) in a limit of time going to infinity is 0, but amount of \(S_0\) is dependent on reaction rates; and CRN in (c) is rate-independent, where concentrations of both \(S_0\) and \(S_1\) converge to 0.

Elementary reactions. Elementary reactions have at most 2 reactants and at most 2 products. In general, it is unlikely that 3 (or more) molecules interact at the same time, and thus elementary reactions are the ones commonly occurring in nature. In addition, reactions with more than 2 reactants can be represented with elementary reactions; e.g. reaction \(A + B + C \rightarrow D\) can be constructed with two elementary reactions: \(A + B \rightarrow T\) and \(T + C \rightarrow D\).

Figure 3 shows the Alloy model of elementary reactions, which specializes (restricts) the general CRN model crn. The Alloy model elementary imports

---

Alloy shows each instance as a valuation to the sets and relations declared in the model, and also supports visualizing the instances as graphs. We write the reactions here using their natural representation for clarity.
module catalytic
open elementary
pred Catalytic() { all r: Reaction | CatalyticReaction[r] }
pred CatalyticReaction(r: Reaction) { some elems[r.reactants] & elems[r.products] }
run { Catalytic and Elementary } for 2

Fig. 4: Catalytic reactions.

module metabolic
open catalytic
pred Metabolic[] { Catalytic[] and all s: Species | (some r: Reaction | IsCatalyst[s, r]) implies all x: Reaction | Contains[x, s] implies IsCatalyst[s, x] }
pred IsCatalyst[s: Species, r: Reaction] { s in Int.(r.reactants) & Int.(r.products) }
pred Contains[r: Reaction, s: Species] { ContainsAsReactant[r, s] or ContainsAsProduct[r, s] }
pred ContainsAsReactant[r: Reaction, s: Species] { s in Int.(r.reactants) }
pred ContainsAsProduct[r: Reaction, s: Species] { s in Int.(r.products) }

Fig. 5: Metabolic reactions.

(open) the crn model and defines the predicate Elementary, which uses the conjunction (and) of two helper predicates MaxReactantsNum and MaxProductsNum to characterize elementary reactions. The predicate lte is a standard Alloy utility predicate and represents the ≤ comparison.

Catalytic reactions. Next, we model catalytic reactions (Figure 4). The predicate Catalytic uses the helper predicate CatalyticReaction to require each reaction to be catalytic, i.e., have some species that is both a reactant and a product in that reaction. The Alloy utility function elems represents the set of elements in its argument sequence; the operator ‘&’ represents set intersection. The run command instructs the analyzer to create an instance that is both a catalytic and an elementary reaction within a scope of 2, i.e., at most 2 atoms in each sig. An example instance created by executing the command is:

\[ S_0 + S_1 \rightarrow S_0 + S_0 \]
\[ S_0 + S_1 \rightarrow S_1 + S_1 \]

Metabolic reactions. In metabolic networks catalysts are proteins that act upon substrates that are small molecules. Thus metabolic reactions are a form of catalytic reactions in which if a species appears as a catalyst in a reaction, then it has to be a catalyst in all reactions in which the species occurs. The predicate Metabolic in Figure 5 specifies metabolic reactions.

Strands and gates. We next model synthetic CRNs which use DNA strand displacement cascades for its implementation. Strand displacement interactions correspond to reactions between two types of molecules: “gates” and “strands”, where the reacting strand displaces the strand previously sequestered in the gate complex. We first capture the bipartite nature of the reactions: Figure 6 declares strands and gates as disjoint subsets (extends) that partition species. The predicate StrandsAndGates requires that each reaction has exactly 2 reactants and 2
module strandsandgates
open crn

sig Strand, Gate extends Species {}
fact { Strand + Gate = Species } -- strands and gates partition species

pred StrandsAndGates() {
    ExactReactantsNum[2] and ExactProductsNum[2] and
    all r: Reaction {
        some Int.(r.reactants) & Strand and some Int.(r.reactants) & Gate
        some Int.(r.products) & Strand and some Int.(r.products) & Gate }
}

pred ExactReactantsNum[num: Int] { all r: Reaction | eq[#r.reactants, num] }
pred ExactProductsNum[num: Int] { all r: Reaction | eq[#r.products, num] }

Fig. 6: Strands and gates.

Fig. 7: DNA strand displacement reaction with the seesaw gate motif. There are two reactants (a strand and a gate) and two products (a strand and a gate). A gate consists of two strands bound together. (For simplicity the usual helical structure of DNA is not shown.) Labels show binding sites (domains); a star indicates Watson-Crick complement such that domain $x$ binds $x^*$. In order for the reaction to happen, the complementary domains must match as shown. Such reactions can be cascaded since the strands $<a,t,b>$ and $<b,t,c>$ can react with other seesaw gates.

Products, and moreover has a strand and a gate as a reactant, and a strand and a gate as a product.

**Seesaw networks.** A simple yet powerful subclass of DNA strand displacement reactions is the “seesaw” reaction. This reaction has been used to create some of the largest synthetic biochemical reaction networks, including logic circuits and neural networks [12,44]. The molecular structure schematic for a seesaw reaction is shown in Fig. 7. Figure 8 model seesaw reactions by specializing the model of strands and gates (Figure 6), capturing the abstract structure in an Alloy model. The sig Domain models the binding domains. The sig DNASpecies is a subset (in) of species, and left and right are binary relations that map DNASpecies to their left and right domains respectively. The keyword lone constraints the relations to be partial functions. The fact UseAll requires all species to be DNA species, and requires all domains to be a part of some species. The sigs RightGate and LeftGate partition gates. The fact Function requires strands and gates to have exactly one left and exactly one right domain. The predicate ReactStrandAndLeftGate is true if inputs (reactants and products) conform to the
open strandsandgates

sig Domain {} 

sig DNAspecies in Species { left, right: lone Domain } 

fact UseAll { DNAspecies = Species and DNAspecies.(left + right) = Domain } 

sig RightGate, LeftGate extends Gate {} 

fact Function { all s: Strand + LeftGate + RightGate | one s.left and one s.right } 

pred ReactStrandAndLeftGate[s: Strand, lg: LeftGate, s':Strand, rg': RightGate] { 
  (s in Strand and lg in LeftGate and s' in Strand and rg' in RightGate 
   and s.right = lg.left and s'.left = lg.left and s'.right = lg.right 
   and rg'.left = s.left and rg'.right = s.right) } 

pred ReactStrandAndRightGate[s: Strand, rg: RightGate, s': Strand, lg': LeftGate] { 
  (s in Strand and rg in RightGate and s' in Strand and lg' in LeftGate 
   and s.left = rg.right and s'.left = rg.left and s'.right = rg.right 
   and lg'.left = s.left and lg'.right = s.right) } 

pred Seesaw { 
  StrandsAndGates[] and all r: Reaction { 
    let s = 0.(r.reactants), g = 1.(r.reactants), 
    s' = 0.(r.products), g' = 1.(r.products) { 
      ReactStrandAndLeftGate[s, g, s', g'] or ReactStrandAndRightGate[s, g, s', g'] } 
  } } 

GenSeesaw: run Seesaw for exactly 1 Reaction, exactly 3 Domain, exactly 4 Species 

Fig. 8: Seesaw gates.

interaction rules of a strand and a left gate, specifically s and lg interact, i.e., the 
right domain of s matches the left domain of lg, and produce s' and rg' where 
the left and right domains of s' match those of lg, and left and right domains of 
rg' match those of s; likewise, ReactStrandAndRightGate specifies the interaction 
of a strand and a right gate. The predicate Seesaw specifies each reaction to be 
a ReactStrandAndLeftGate or ReactStrandAndRightGate.

The second instance generated by Alloy running the predicate with com-
mand GenSeesaw is S₀(a, b) + LG(b, c) → S₁(b, c) + RG(a, b), where S₀ and S₁ 
are strands, LG left gate, RG right gate, and domains a and b are shown in parenthesis. Note that this reaction is equivalent to the one shown in Fig. 7.

Feed-forward, non-competitive CRNs. Figure 9 models feed-forward, 
non-competitive CRNs. Recall, we define feed-forward as: there exists a total 
ordering on the species such that every reaction which produces a species S 
must consume some species earlier in the ordering than S. Also, we define non-
competitive as: every species is consumed by at most one reaction.

To model feed-forward constraints, one approach is to directly enforce a total 
ordering on the species with respect to the feed-forward property. Observe that
open elementary

one sig Graph { edges: Reaction -> Reaction }
{ all r1, r2: Reaction | r1->r2 in edges implies some s: Species |
  NetProduce[r1, s] and NetConsumes[r2, s]
}
{ all s: Species | all r1, r2: Reaction |
  NetProduce[r1, s] and NetConsumes[r2, s] implies r1->r2 in edges }

pred DAG[] { all r: Reaction | r !in r.^(Graph.edges) }

pred MustConsume[] { all r: Reaction | r !in r.^(Graph.edges) }

pred NetProduce[r: Reaction, s: Species] { -- r net produces s
  lt[#indsOf[r.reactants, s], #indsOf[r.products, s]] }

pred NetConsume[r: Reaction, s: Species] { -- r net consumes s
  gt[#indsOf[r.reactants, s], #indsOf[r.products, s]] }

pred NonCompetitive[] { all r1, r2: Reaction | all s: Species |
  (NetProduce[r1, s] and NetProduce[r2, s]) implies r1 = r2 }

pred Feedforward[] { Elementary[] and DAG[] and NonCompetitive[] and MustConsume[] }

Fig. 9: Feed-forward, non-competitive CRNs in Alloy.

there can be multiple valid total orderings of species for the same feed-forward CRN, which means that when enumerating instances for the resulting model, multiple unique instances are created for the same CRN. This is useful when finding all total orderings that exist for a CRN. However, in our case, the goal is to search for the CRN exhibiting desired functionality. So, we aim to enumerate each CRN once and not multiple times, and as quickly as possible.

Our modeling of feed-forward constraints introduces a new singleton (one) sig, termed Graph, to model a dependency relation, termed edges, between reactions. The constraint paragraph that immediately follows the sig declaration implicitly introduces a fact that defines the edges. Specifically, there is an edge from reaction r1 to reaction r2 if and only if there is some species s such that r1 produces s and r2 consumes s. Total ordering is achieved by the predicate DAG that requires the graph to be directed-acyclic. The operator ‘^’ is transitive closure and r.^ (Graph.edges) represents the set of all reactions that are reachable from r. The predicate Feedforward defines elementary, feed-forward, and non-competitive reactions where each reaction must consume some species.

3 Searching CRNs

This section describes our technique for finding a smallest CRN computing a desired function f. The section is divided in three parts: 1) a method for checking if a given CRN computes f (Section 3.1), 2) a bounded exhaustive search
algorithm enumerating all CRNs in a given class and within a given bounds (Section 3.2), and 3) presentation of new results discovered using the proposed techniques (Section 3.3).

3.1 Method to determine if CRN computes $f$

In this section we describe our algorithm for checking if a CRN computes a function of interest ($f$).

**Conservation Equations.** We first construct a set of conservation equations for the CRN which describe concentrations of species in terms of their initial concentrations and reaction fluxes. A reaction flux is equal to the total “flow of material” through the reaction. We associate a flux variable to each reaction, where $\text{flux}_i$ represents the flux of the reaction $i$. Now, concentration of a species $S$ can be expressed in terms of its initial concentration $S_0$ and reaction fluxes:

$$s = s_0 + \sum_{i=1}^{N} \text{netGain}(\text{rxn}_i, S) \cdot \text{flux}_i \quad (1)$$

where $\text{netGain}(\text{rxn}_i, S)$ is the net stoichiometric gain of species $S$ in the reaction $i$ (negative in the case of loss), and $N$ is the number of reactions in the CRN. For example, the CRN from Fig. 1 generates the equations shown in 2. The variables on the left side of equations represent concentrations of species, variables with suffixes 0 represent initial concentrations of species (e.g., $z_{10}$ is initial concentration of species $z_1$), and finally $\text{flux}_i$ variables represent fluxes of reactions.

$$a = a_0 - \text{flux}_1, b = b_0 - \text{flux}_2$$
$$z_1 = z_{10} + \text{flux}_1 - \text{flux}_3, z_2 = z_{20} + \text{flux}_2 - \text{flux}_3$$
$$k = k_0 + \text{flux}_3 - \text{flux}_4, y = y_0 + \text{flux}_1 + \text{flux}_2 - \text{flux}_4 \quad (2)$$

**Equilibrium Condition.** We next use the above conservation equations to find equilibria. Since we focus on rate-independent computation, we search for static equilibria only (none of the reactions is occurring). A static equilibrium corresponds to every reaction having at least one reactant in zero concentration. Thus, we create multiple systems of equations from the conservation equations, where each system corresponds to setting a set of species in conservation equations to zero, and the set contains a reactant from each reaction. The solution of each such constructed system of equations represents concentrations of species at an equilibrium. Different equilibria will be reached from different initial conditions.

As an example, consider again the CRN shown in Fig. 1. All combinations of species containing a reactant from each reaction are: $(A, B, Z_1, Y)$, $(A, B, Z_2, Y)$, $(A, B, Z_1, K)$, $(A, B, Z_2, K)$. For each combination we set species concentrations

---

$^5$In chemical kinetics, **static** equilibrium refers to an equilibrium where none of the reactions occur. In contrast, in **dynamic** equilibria, concentrations don’t change over time because the effects of the different reactions cancel out. Note that dynamic equilibria are not rate-independent since changing a reaction rate affects the equilibrium concentrations of the species involved in that reaction.
to zero and solve the system. This results in 4 solutions shown in (we do not show solutions for flux variables due to the space limits).

\[ a = 0, b = 0, k = -b_0 + k_0 - y_0 + z_{10}, y = 0, z_1 = 0, z_2 = -a_0 + b_0 - z_{10} + z_{20} \]
\[ a = 0, b = 0, k = -a_0 + k_0 - y_0 + z_{20}, y = 0, z_1 = a_0 - b_0 + z_{10} - z_{20}, z_2 = 0 \]
\[ a = 0, b = 0, k = 0, y = b_0 - k_0 + y_0 - z_{10}, z_1 = 0, z_2 = -a_0 + b_0 - z_{10} + z_{20} \]
\[ a = 0, b = 0, k = 0, y = a_0 - k_0 + y_0 - z_{20}, z_1 = a_0 - b_0 + z_{10} - z_{20}, z_2 = 0 \]

Although there are 4 solutions, for any particular initial concentrations of the species only one of the solutions is non-negative, and thus feasible.

**Check whether CRN computes \( f \).** We then check if the equilibrium solutions are equivalent to \( f \). In general, we do not know which species correspond to the input and which to the output, and thus we need to check for all possible combinations of the input and the output species. First, we construct all input \( n \)-tuples without repeating elements from a set of species (where \( n \) is the number of the inputs to \( f \)). Second, for all species that are not in the input tuple we set initial concentrations to zero. Third, for the output species we try any of the remaining species. Fourth, for a given set of input and output species, we construct a piecewise function, where each solution is valid if concentrations of species are non-negative. Finally, we use Mathematica’s constraint solving procedure `FindInstance` to check if the constructed piecewise function differs from function \( f \).

To illustrate on our example, consider setting input species to \( A \) and \( B \), and output to \( Y \). System of equations (3) reduces to the system (4):

\[ a = 0, b = 0, k = -b_0, y = 0, z_1 = 0, z_2 = -a_0 + b_0 \]
\[ a = 0, b = 0, k = -a_0, y = 0, z_1 = a_0 - b_0, z_2 = 0 \]
\[ a = 0, b = 0, k = 0, y = b_0, z_1 = 0, z_2 = -a_0 + b_0 \]
\[ a = 0, b = 0, k = 0, y = a_0, z_1 = a_0 - b_0, z_2 = 0 \]

The first two solutions are infeasible since they result in species \( k \) having negative concentration, \(-b_0\) and \(-a_0\). More precisely they are feasible only in the trivial case where \( a_0 = 0 \land b_0 = 0 \), but we can ignore that case. The third solution is feasible when \( b_0 \geq a_0 \), in which case \( y = b_0 \); while fourth solution is feasible when \( a_0 \geq b_0 \), in which case \( y = a_0 \). Thus, we can construct the piecewise function unifying multiple equilibrium solutions into a single function:

\[
y = \begin{cases} 
  b_0 & b_0 \geq a_0 \\
  a_0 & a_0 \geq b_0 
\end{cases}
\]

Next, once we constructed the equilibrium piecewise function \( (y(a_0, b_0)) \) we invoke the Mathematica’s constraint solving procedure `FindInstance` to find an assignment of inputs \((a_0, b_0)\) for which \( y \) differs from \( f \), with additional condition that initial concentrations are non-negative \((a_0 \geq 0 \land b_0 \geq 0)\). If no counterexample is found, then the CRN computes \( f \) and we have finished our search. On the other hand, if a counterexample is found, then we repeat the
Algorithm 1 ComputesF

| Input: CRN crn, Function f, Number of inputs N. |
| Output: True if crn computes f; false otherwise. |

1: procedure ComputesF
2:   conservationEquations ← constructConservationEquations(crn)
3:   equilibriumSolutions ← ∅
4:   for each speciesSet ∈ getAllReactantCombinations(crn) do
5:       equilibriumEquations ← setConcToZero(conservationEquations, speciesSet)
6:       solution ← solve(equilibriumEquations)
7:       equilibriumSolutions.add(solution)
8:   end for
9:   for each \{x\_1, x\_2, ..., x\_N, y\} ∈ getInputOutputSpecies(crn, N) do
10:      nonInputSpecies ← getOtherSpecies(crn, \{x\_1, x\_2, ..., x\_N\})
11:      newSols ← setInitialConcToZero(equilibriumSolutions, nonInputSpecies)
12:      pwF ← constructPiecewise(newSols, y)
13:      counterExample ← FindInstance(pwF \neq f(x\_1, x\_2, ..., x\_N))
14:      if counterExample = null then return true
15:   end for
16:   return false
17: end procedure

procedure for the next combination of input and output species. When the list of input and output combinations is exhausted we can conclude that the CRN does not compute f.

**Algorithm.** We implement this functionality in Mathematica by defining ComputesF function described in Algorithm 1. In step 2, conservation equations are constructed, while in step 3 we initialize a set of equilibrium solutions equilibriumSolutions to the empty set. In steps 4–8, we iterate over all existing sets of species containing at least one reactant from each reaction. Specifically, function getAllReactantCombinations computes Cartesian product over sets of reactants from different reactions; and removes elements with the same sets of species. In step 5 we update the conservation equations by setting speciesSet concentrations to zero, and save the linear system in equilibriumEquations. In steps 6–7 we solve the system of linear equations and add it to the list of equilibrium solutions (note that since we are focused on feed-forward non-competitive reactions, a unique solution will always exist). Next, we iterate over all combinations of input and output species \{x\_1, x\_2, ..., x\_N, y\}, where x\_1, x\_2, ..., x\_N represent input species, and y output species. In step 10 we get all the species that are not in the input species set. In step 11 we modify the equilibrium solutions by setting initial concentrations of nonInputSpecies to zero, and we save the result in newSols. In step 12 we construct a piecewise function pwF out of newSols. Finally, in step 13 we invoke the FindInstance method to find input values for which pwF is different then f. If such solution is not found then counterExample is null, and constructed pwF is implementing f; in which case procedure returns true. If counterexample is found then the same steps are repeated for different set of input and output species. Finally, if all combinations are exhausted procedure returns false.
Algorithm 2 Search Algorithm

Input: Model (model), Generation bounds (scope), Function (f), Inputs (N).
Output: CRN that computes f if found; otherwise, null.

1: procedure ExhaustiveSearch
2: for each instance ∈ Alloy.findAllInstances(model, scope) do
3:    crn ← translate(instance)
4:    if ComputesF(crn, f, N) then return crn
5: end for
6: return null
7: end procedure

3.2 Exhaustive Search

In this section we describe our algorithm (shown in Algorithm 2) that performs exhaustive search of space of CRNs respecting properties defined by an Alloy model, to find the CRN implementing desired function.

Inputs to the algorithm are Alloy model, size of CRNs (e.g., number of reactions and species) defined by the scope, searched for function f, and number of inputs to the function N. Function findAllInstances accepts Alloy model definition and scope, and enumerates all possible instances that satisfy the Alloy model. Each Alloy instance is translated to CRN (step 3). Then, in step 4 we invoke the Algorithm 1 to check if CRN computes f. If CRN is found then we return it (step 4). If after checking all instances no satisfying CRN is found then the procedure returns null.

Bounded exhaustive search. To find the smallest CRN computing f we conduct a bounded exhaustive search. Our goal is to find a smallest (in terms of numbers of species and reactions) feed-forward, non-competitive CRN that computes f. We use iterative deepening [23, 25, 27] where we start from a small scope and iteratively increase it to a larger scope until a desired CRN is found, where for each scope we invoke Algorithm 2.

3.3 New Results

In this section we present new discoveries made using the proposed techniques. We focus on the class of feed-forward, non-competitive CRNs since they are always rate-independent. We first consider the max function, confirming that the CRN shown in Fig. 1 is the smallest max-computing CRN. Next, we turn the smallest CRNs implementing ReLU and abs functions.

Smallest max CRN. We perform bounded exhaustive search for 1–4 reactions, and 1–6 species, starting with smaller number of species and reactions, and iteratively increasing the scope until the max is found. Table I shows the number of enumerated CRNs and Alloy enumeration time for different scope sizes. Note that while we perform isomorphic breaking[7] not all isomorphic cases are pruned, and thus number of non-isomorphic instances may be less then numbers reported in Table I. We perform (not perfect) isomorphic breaking in Alloy by requiring lexicographic ordering on reactions among other things, which we do

---

7Alloy can generate isomorphic instances, i.e., two instances that are distinct but there exists a permutation on atoms, which maps one instance to the other
| 1 Reaction | 2 Reactions | 3 Reactions | 4 Reactions |
|------------|------------|------------|------------|
| 1 Species  | 3 00:00:00 | 0 00:00:00 | 0 00:00:00 | 0 00:00:00 |
| 2 Species  | 10 00:00:00 | 40 00:00:00 | 0 00:00:00 | 0 00:00:00 |
| 3 Species  | 6 00:00:00 | 281 00:00:01 | 1,060 00:00:02 | 0 00:00:00 |
| 4 Species  | 1 00:00:00 | 479 00:00:01 | 11,082 00:00:17 | 43,550 00:01:12 |
| 5 Species  | 0 00:00:00 | 326 00:00:01 | 31,929 00:00:43 | 590,891 00:50:57 |
| 6 Species  | 0 00:00:00 | 104 00:00:00 | 41,199 00:00:57 | 2,394,480 09:52:11 |

Table 1: Number of enumerated feed-forward, non-competitive CRNs and wall-clock times (hh:mm:ss) for the enumeration procedure.

\[
\begin{align*}
X^+ &\rightarrow M + Y^+ \\
M + X^- &\rightarrow Y^- \\
X^+ &\rightarrow Y^+ + C \\
X^- &\rightarrow Y^+ + E \\
C + E &\rightarrow 2Y^-
\end{align*}
\]

Fig. 10: Minimal ReLU CRN (left) and abs CRN (right).

not show here due to the space constraints. The first occurrence of \(\text{max}\) is found in the scope of 4 reactions and 6 species, and it was the 635, 250th instance Alloy enumerated in that scope. The CRN discovered is equivalent to the one shown in Fig. 1 modulo reaction and species ordering.

**Dual-rail convention.** Concentrations of species are always non-negative, making it impossible to represent negative values directly. However, there is a natural way to extend computation semantics to negative values. Instead of using a single species to represent a value, in dual-rail convention a value is represented by a difference between two species (e.g., the output value is equal to the concentration of species \(Y^+\) minus the concentration of \(Y^-\)). Note that we slightly modify our Algorithm 1 to search for dual-rail computation. As an example in a case of functions with a single input and a single output, we select two input species (\(X^+\) and \(X^-\)) representing positive and negative parts, and two output species (\(Y^+\) and \(Y^-\)) representing positive and negative parts, and check if \(y^+ - y^-\) equals \(f(x^+ - x^-)\).

**Smallest ReLU CRN.** Using the above described procedure we run experiments for finding the smallest CRN computing ReLU (rectified linear unit) function, and we discover the smallest ReLU computing CRN shown in Fig. 10. The output is \(y = y^+ - y^-\), and input \(x = x^+ - x^-\), and \(y = \text{ReLU}(x)\). Note that CRNs were already enumerated when searching for \(\text{max}\), and that was no need to re-enumerate them as they were saved on disk.

**Smallest abs CRN.** We conducted a similar experiment for finding the smallest CRN computing absolute value function, and discovered the smallest abs computing CRN shown in Fig. 10.

### 4 Related Work

**CRN Enumeration.** Deckard et al. [15] developed an online library of reaction networks, which was extended [3] to catalog reactions of several classes. These approaches generate non-isomorphic bipartite graphs (two types of vertices for species and reactions) with undirected edges relying on Nauty library [41]. Each
such constructed graph is then reified as multiple CRN instances. Recent generalization of this work gives the first complete count of all 2-species bimolecular CRNs, and counts for other classes of CRNs such as mass-conserving and reversible [48]. Rather than focusing on removing all isomorphisms and generating exact counts of non-isomorphic CRNs in each class, our work allows the user to flexibly specify and analyze structural properties of CRNs of interest (enabling direct generation of CRNs following the structure). For example, it is not clear how to encode molecular structure (such as we do for seesaw networks) using graph-based models.

**Minimal Systems with Desired Behavior.** Complementary to CRN enumeration, previous work also tackled the problem of finding minimal CRNs respecting some desired properties or exhibiting certain behavior. Wilhelm [53] discovers the smallest elementary CRN with bistability. Wilhelm and Heinrich [54] similarly detect the smallest CRN with Hopf bifurcation. In comparison with this line of work, our paper presents a more general framework that allows specifying structure and properties, including different functions, of CRNs to be explored.

Recent work due to Murphy et al [42] is close to ours in spirit, but focuses on discrete systems (integer molecular counts of the species). Cardelli et al [7] take a program synthesis approach to generate CRNs that follow properties provided by a certain “sketch” language (i.e., a template) using SMT solvers on the back end [3][14].

**Computational power of CRNs.** Much ongoing work has explored computational power of CRNs [28,59,46,50]. It is shown how to map complex computation to CRNs, such as mapping polynomials to chemical reactions, mapping discrete algorithms, and even defining a high-level imperative languages that map to CRNs. We believe that by exploring CRNs bottom up, we may found answers of what the appropriate (more efficient) high-level primitives are to be used for implementing such high-level functionality.

5 Conclusion

We introduced the use of Alloy, a framework for modeling and analyzing structural constraints and behavior in software systems, to enumerate CRNs with declaratively specified properties. We showed how this framework can enumerate CRNs with a variety of structural constraints including biologically motivated catalytic networks and metabolic networks, and see-saw networks motivated by DNA nanotechnology. We also used the framework to explore analog function computation in rate-independent CRNs. We applied our approach in a case-study to find the smallest CRNs computing the \( \max \), ReLU and \( \text{abs} \) functions in a natural subclass of rate-independent CRNs where rate-independence follows from structural network properties.

References

1. Angluin, D., Aspnes, J., Eisenstat, D.: A simple population protocol for fast robust approximate majority. Distributed Computing 21(2), 87–102 (2008)
2. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20(4), 279–304 (2007)
3. Banaji, M.: Counting chemical reaction networks with nauty. arXiv preprint arXiv:1705.10820 (2017)
4. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., Jovanović, D., King, T., Reynolds, A., Tinelli, C.: Cvc4. In: CAV (2011)
5. Bernot, G., Comet, J.P., Richard, A., Guespin, J.: Application of formal methods to biological regulatory networks: extending thomas asynchronous logical approach with temporal logic. Journal of theoretical biology (2004)
6. Cardelli, L.: Morphisms of reaction networks that couple structure to function. BMC systems biology 8(1), 84 (2014)
7. Cardelli, L., Češka, M., Fränzle, M., Kwiatkowska, M., Laurenti, L., Paoletti, N., Whitby, M.: Syntax-guided optimal synthesis for chemical reaction networks. In: CAV (2017)
8. Chalk, C., Kornerup, N., Reeves, W., Soloveichik, D.: Composable rate-independent computation in continuous chemical reaction networks. In: CMSB, pp. 256–273. Springer (2018)
9. Chen, H.L., Doty, D., Soloveichik, D.: Deterministic function computation with chemical reaction networks. Natural computing 13(4), 517–534 (2014)
10. Chen, H.L., Doty, D., Soloveichik, D.: Rate-independent computation in continuous chemical reaction networks. In: Proceedings of the 5th conference on Innovations in theoretical computer science. pp. 313–326. ACM (2014)
11. Chen, Y.J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from DNA. Nature nanotechnology 8(10), 755 (2013)
12. Cherry, K.M., Qian, L.: Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. Nature 559(7714), 370 (2018)
13. Clarke, E.M., Grumberg, O., Kroening, D., Peled, D., Veith, H.: Model Checking
14. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
15. Deckard, A.C., Bergmann, F.T., Sauro, H.M.: Enumeration and online library of mass-action reaction networks. arXiv preprint arXiv:0901.3067 (2009)
16. Dennis, G., Chang, F.S., Jackson, D.: Modular verification of code with SAT. In: ISSTA (2006)
17. Een, N., Sörensson, N.: An extensible SAT-solver. Santa Margherita Ligure, Italy (2003)
18. Frias, M.F., Galeotti, J.P., Pombo, C.G.L., Aguirre, N.M.: DynAlloy: Upgrading Alloy with actions. In: ICSE (2005)
19. Galeotti, J.P., Rosner, N., Pombo, C.G.L., Frias, M.F.: TACO: efficient SAT-based bounded verification using symmetry breaking and tight bounds. TSE (2013)
20. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.; Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP) (2018)
21. Giacobbe, M., Guet, C.C., Gupta, A., Henzinger, T.A., Paixão, T., Petrov, T.: Model checking gene regulatory networks. In: TACAS (2015)
22. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics (2011)
23. Godefroid, P.: Verisoft: A tool for the automatic analysis of concurrent reactive software. In: CAV. pp. 476–479. Springer (1997)
24. Hahnloser, R.H., Sarpeshkar, R., Mahowald, M.A., Douglas, R.J., Seung, H.S.: Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature (2000)
25. Havelund, K., Pressburger, T.: Model checking java programs using java pathfinder. International Journal on Software Tools for Technology Transfer 2(4), 366–381 (2000)
26. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyns, O.: Probabilistic model checking of complex biological pathways. Theoretical Computer Science (2008)
27. Holzmann, G.J.: The SPIN model checker: Primer and reference manual, vol. 1003. Addison-Wesley Reading (2004)
28. Huang, D.A., Jiang, J.H.R., Huang, R.Y., Cheng, C.Y.: Compiling program control flows into biochemical reactions. In: Proceedings of the International Conference on Computer-Aided Design. pp. 361–368 (2012)
29. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: CAV (2017)
30. Jackson, D.: Alloy: A lightweight object modelling notation. ACM Transactions on Software Engineering and Methodology (TOSEM) 11(2), 256–290 (2002)
31. Jackson, D., Fekete, A.: Lightweight analysis of object interactions. In: TACS (2001)
32. Jackson, D., Schechter, I., Shlyakhter, I.: ALCOA: The Alloy constraint analyzer. In: International Conference on Software Engineering, Limerick, Ireland (Jun 2000)
33. Jackson, D., Vaziri, M.: Finding bugs with a constraint solver. In: ISSTA (2000)
34. Kang, E., Milicevic, A., Jackson, D.: Multi-representational security analysis. In: FSE (2016)
35. Khurshid, S., Marinov, D., Jackson, D.: An analyzable annotation language. In: ACM SIGPLAN Notices. vol. 37, pp. 231–245. ACM (2002)
36. Lakin, M.R., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of DNA strand displacement devices using probabilistic model checking. Journal of the Royal Society Interface (2012)
37. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. nature (2015)
38. Lee, T.I., Rinaldi, N.J., Robert, F., Odom, D.T., Bar-Joseph, Z., Gerber, G.K., Hannett, N.M., Harbison, C.T., Thompson, C.M., Simon, I., et al.: Transcriptional regulatory networks in Saccharomyces cerevisiae. Science 298(5594), 799–804 (2002)
39. Magnasco, M.O.: Chemical kinetics is turing universal. Physical Review Letters 78(6), 1190 (1997)
40. Marinov, D., Khurshid, S.: TestEra: A novel framework for automated testing of java programs. In: ASE. pp. 22–31 (2001)
41. McKay, B.D., Piperno, A.: Practical graph isomorphism, (II). Journal of Symbolic Computation (2014)
42. Murphy, N., Petersen, R., Phillips, A., Yordanov, B., Dalchau, N.: Synthesizing and tuning stochastic chemical reaction networks with specified behaviours. Journal of The Royal Society Interface 15(145), 20180283 (2018)
43. Ptacek, J., Devgan, G., Michaud, G., Zhu, H., Zhu, X., Fasolo, J., Guo, H., Jona, G., Breitkreutz, A., Sopko, R., et al.: Global analysis of protein phosphorylation in yeast. Nature 438(7068), 679 (2005)
44. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science 332(6034), 1196–1201 (2011)
45. Qian, L., Winfree, E.: A simple DNA gate motif for synthesizing large-scale circuits. Journal of the Royal Society Interface 8(62), 1281–1297 (2011)
46. Salehi, S.A., Parhi, K.K., Riedel, M.D.: Chemical reaction networks for computing polynomials. ACS Synthetic Biology 6(1), 76–83 (2017)
47. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proceedings of the National Academy of Sciences 107(12), 5393–5398 (2010)
48. Spaccasassi, C., Yordanov, B., Phillips, A., Dalchau, N.: Fast enumeration of non-isomorphic chemical reaction networks. In: CMSB. pp. 224–247. Springer (2019)
49. Srinivas, N., Parkin, J., Seelig, G., Winfree, E., Soloveichik, D.: Enzyme-free nucleic acid dynamical systems. Science 358(6369), eaal2052 (2017)
50. Vasic, M., Soloveichik, D., Khurshid, S.: Crn++: molecular programming language. In: International Conference on DNA Computing and Molecular Programming. pp. 1–18. Springer (2018)
51. Volterra, V.: Variazioni e fluttuazioni del numero d’individui in specie animali conviventi. C. Ferrari (1927)
52. Wang, Q., Zuliani, P., Kong, S., Gao, S., Clarke, E.M.: Sreach: A probabilistic bounded delta-reachability analyzer for stochastic hybrid systems. In: CMSB (2015)
53. Wilhelm, T.: The smallest chemical reaction system with bistability. BMC systems biology 3(1), 90 (2009)
54. Wilhelm, T., Heinrich, R.: Smallest chemical reaction system with hopf bifurcation. Journal of mathematical chemistry 17(1), 1–14 (1995)
55. Zhang, D.Y., Seelig, G.: Dynamic dna nanotechnology using strand-displacement reactions. Nature chemistry 3(2), 103 (2011)