CRNs Exposed: A Method for the Systematic Exploration of Chemical Reaction Networks

Marko Vasic
The University of Texas at Austin, TX, USA
https://marko-vasic.github.io/
vasic@utexas.edu

David Soloveichik
The University of Texas at Austin, TX, USA
http://users.ece.utexas.edu/~soloveichik/
david.soloveichik@utexas.edu

Sarfraz Khurshid
The University of Texas at Austin, TX, USA
https://users.ece.utexas.edu/~khurshid/
khurshid@utexas.edu

Abstract

Formal methods have enabled breakthroughs in many fields, such as in hardware verification, machine learning and biological systems. The key object of interest in systems biology, synthetic biology, and molecular programming is *chemical reaction networks* (CRNs) which formalizes *coupled chemical reactions* in a well-mixed solution. 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 seesaw 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 $X \to Y + Y$ 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*, *minmax*, *abs* and *ReLU* (rectified linear unit) functions in a natural subclass of rate-independent CRNs where rate-independence follows from structural network properties.

2012 ACM Subject Classification Theory of computation

Keywords and phrases molecular programming, formal methods

Digital Object Identifier 10.4230/LIPIcs.DNA.2020.4

Supplementary Material We release the source code of our tool at `https://github.com/marko-vasic/crnsExposed` to enable others make use of it, and extend it further.

Acknowledgements This work was supported in part by NSF grants CCF-1901025 to DS and CCF-1718903 to SK.

1 Introduction

Formal methods have enabled breakthroughs in many fields, e.g., in hardware verification [15], machine learning [23, 32], and biological systems [5, 24, 29, 40, 61]. 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 [60],

distributed computing [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 [6, 12, 53, 55, 57].

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] was later identified with a variety of biological networks [7]. 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 [33]. The Alloy tool performs *scope-bounded* analysis [35]. 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 [20] to analyze the model. Alloy is used in a wide range of areas in software engineering, including software design [21, 34], analysis [19, 22, 36, 38], testing [44], and security [37]. 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 [42, 48]. 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 "bipartite" in the sense that a species is either always a catalyst or never a catalyst. *Autocatalytic networks* are another interesting subclass of catalytic networks in which the (auto)catalyst generates another copy of itself. Autocatalysis is useful for exponential amplification and oscillation.

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 [64]. 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 [13, 49, 50] that makes use of a reversible strand displacement reaction. We designed an Alloy model 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 [11]. Consider the reaction $X \to Y + Y$, 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 $X_1 + X_2 \to Y$ computes the

$$
\begin{aligned}
A &\longrightarrow Z_1 + Y \\
B &\longrightarrow Z_2 + Y \\
Z_1 + Z_2 &\longrightarrow K \\
Y + K &\longrightarrow \emptyset
\end{aligned}
$$

Figure 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.)

"minimum" function since 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: if a species is consumed in a reaction then it cannot appear as a reactant somewhere else. Such constraints on the structure of the network can be easily encoded in the Alloy specification. We also require each reaction to consume at least one species (boundedness condition). We show in Appendix A that these conditions ensure that the CRN is rate-independent.

Focusing on the class of feed-forward, non-competitive CRNs, we search for the smallest reaction networks implementing *max*, *minmax*, *abs*, and *ReLU* (rectified linear unit) functions. As an example of the kind of computation we achieve, consider the *max* computing CRN shown in Figure 1. This CRN was previously studied [10, 11]; 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 [9, 11]. The *ReLU* function was first introduced due to the biological motivations explaining functioning of neurons in the brain cortex [27]. Since then, it was used with great success in the machine learning community, particularly in deep learning [25, 41] for realizing artificial neural networks. The simplicity of its implementation suggests that CRNs can naturally realize neural computation [58]. To our knowledge, the smallest implementations of *abs* (absolute value), and *minmax* (a two output function computing both minimum and maximum of two inputs) that we find are novel and have not been previously published.

[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.

Listing 1 General Alloy model of CRNs. "−−" indicate start of a comment.

```
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 }

pred ContainsAsReactant[r: Reaction, s: Species] { s in Int.(r.reactants) }
pred ContainsAsProduct[r: Reaction, s: Species] { s in Int.(r.products) }
```

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 [51], programming logic gates [43], mapping discrete, control flow, algorithms [31], and a molecular programming language translating high-level specifications to chemical reactions [59]. However the complexity of these reaction systems can be infeasible, asking for novel techniques that answer what is the natural way to compute "in reactions". To help answer this question we can take a different, bottom-up approach, and explore what small CRNs naturally do. 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. (See Appendix B for additional background on Alloy.) We first introduce a general model to represent the broadest 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*, *autocatalytic*, and *feed-forward non-competitive* reactions. Next, we present models that encode abstract molecular structure, including *strands and gates* model and a *seesaw* model built on top of it. 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 reactions than 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 arbitrarily many 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. Listing 1 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 signature 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` $\times$ `Int` $\times$ `Species`. In a case of reaction $R0 : X \rightarrow Y + Y$, the value of products relation would be the set: $\{R0 \times 0 \times Y, R0 \times 1 \times Y\}$. Note that we model reactants and products with *seq* instead of *set* to support repetition of a species as a reactant or product, as in the above reaction.

After defining the basic structure, we use Alloy *facts* to add constraints ensuring that enumerated CRNs are well-formed. A `fact` paragraph states a constraint that must always be satisfied, i.e., every solution found (CRN enumerated) must satisfy each fact (and may satisfy additional constraints as desired). For example, the fact `AtLeastOneReactant` requires that every reaction contains at least one reactant. We use universal quantification (`all`) to require that the reactants in each reaction form a non-empty sequence. The keyword `some` in formula "`some E`" for expression `E` constrains it to represent a non-empty set. The operator '`.`' is relational join; specifically, if `r` and `s` are binary relations where the domain of `r` is the same as co-domain of `s`, `r.s` is relational composition, and if `x` is a scalar and `t` is a binary relation where the type of `x` is the co-domain of `t`, `x.t` is relational image of `x` under `t`. Thus, `r.reactants` represents a sequence of reactants in a reaction r.

We ensure that there are no two identical reactions in a CRN using the fact `UniqueReactions`. For all distinct (`disj`) reactions we require that predicate `ReactionsDifferent` holds. A *predicate* (`pred`) paragraph is a named formula that may have parameters. The predicate `ReactionsDifferent` uses logical disjunction (`or`) and invokes `SpeciesSeqDifferent` to constrain its parameters (reactions) `r1` and `r2` to be different.

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 the number of appearances of species in two sequences, and returns true if exists a species that appears a 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. `Int` represents the set of integers.

The predicate `ContainsAsReactant` is true if a given reaction contains a given species as a reactant. Similar holds for `ContainsAsProduct` and reaction products.

Illustrating the General Model. To illustrate using the Alloy analyzer, consider generating an instance of the constraints modeled. The following `Generate` command 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
```

Listing 2 Elementary reactions.

```
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] }
```

Listing 3 Catalytic reactions.

```
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
```

Executing the command `Generate` and enumerating the first three instances creates the following CRNs where $S0$ and $S1$ are species, and $\emptyset$ are waste species [4]:

$S_1 \longrightarrow S_0$
$S_0 \longrightarrow S_1$
(a)

$S_1 \longrightarrow \emptyset$
$S_1 \longrightarrow S_0$
(b)

$S_1 \longrightarrow \emptyset$
$S_0 \longrightarrow S_1$
(c)

While quite small, these three instances exhibit interesting properties, CRN in (a) models a reversible reaction $S1 \longleftrightarrow S0$; CRN in (b) is rate-dependent, where amount of $S1$ in a limit of time going to infinity is 0, but amount of $S0$ is dependent on reaction rates; and CRN in (c) is rate-independent, where concentrations of both $S0$ and $S1$ converge to 0.

Elementary reactions. Elementary reactions have at most 2 reactants and at most 2 products. Elementary reactions are arguably the ones commonly occurring in nature, as it is unlikely that 3 (or more) molecules react or split at the same exact time. Also, reactions with more than 2 reactants can be represented with elementary reactions; e.g. reaction $A + B + C \to D$ can be constructed with two elementary reactions: $A + B \to T$ and $T + C \to D$. (Similarly for products.)

Listing 2 shows the Alloy model of *elementary* reactions, which specializes (restricts) the general CRN model `crn`. The Alloy model `elementary` imports (`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 $\leq$ comparison.

Catalytic reactions. Next, we model catalytic reactions (Listing 3). 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

[4] 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.

Listing 4 Metabolic 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] }
```

Listing 5 Strands and gates.

```
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] }
```

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:

$$S_0 + S_1 \to S_0 + S_0$$
$$S_0 + S_1 \to S_1 + S_1$$

We also model autocatalytic reactions shown in Appendix C.

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 Listing 4 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: Listing 5 declares strands and gates as disjoint subsets (`extends`) that partition species. The predicate `StrandsAndGates` requires that each reaction has exactly 2 reactants and 2 products, and moreover has a strand and a gate as a reactant, and a strand and a gate as a product.

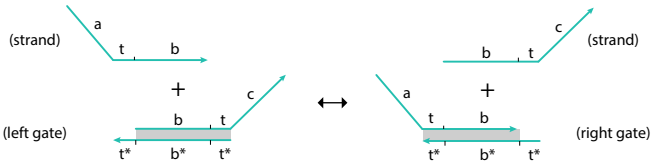

Figure 2 DNA strand displacement reaction with the seesaw gate motif. There are two reactants (a strand and a gate) and two products (a strand and a gate). A gate consists of two strands bound together. (For simplicity the usual helical structure of DNA is not shown.) Labels show binding sites (domains); 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.

Seesaw networks. A simple yet powerful subclass of DNA strand displacement reactions is the "seesaw" model. Seesaw reactions have been used to create some of the largest synthetic biochemical reaction networks, including logic circuits and neural networks [13, 49]. The molecular structure schematic for a seesaw reaction is shown in Figure 2. Listing 6 models seesaw reactions by specializing the model of strands and gates (Listing 5), capturing the abstract molecular structure in an Alloy model. The signature `Domain` models the binding domains. The signature `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 signatures `RightGate` and `LeftGate` partition gates. The fact `UseAll` requires all species to be DNA species, and requires all domains to be a part of some species . The fact `UniqueSpecies` enforces that strands and gates are unique, i.e., there cannot be two or more strands (or left/right gates) with matching left and right domains. The fact `OneDomain` requires strands and gates to have exactly one left and exactly one right domain. The predicate `CanReactStrandAndLeftGate` is true if inputs (reactants) conform to the interaction rules of a strand and a left gate, similar holds for the predicate `CanReactStrandAndRightGate` on strands and right gates. The predicate `CanReact` is true if inputs (reactants) satisfy either `CanReactStrandAndLeftGate` or `CanReactStrandAndRightGate`. The predicate `ReactStrandAndLeftGate` is true if inputs (reactants and products) conform to the 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 functions `ReactantsSet` and `ProductsSet` returns a set of reactants (products) in a reaction. The predicate `Seesaw` specifies: (a) each reaction to be a seesaw reaction by enforcing the predicate `React` on every reaction; (b) that all possible reactions exist, i.e., if two species can interact based on seesaw interaction rules (predicate `CanReact`) than a reaction containing those species as reactants (or products) must exist; (c) that reactions only in one direction exist (to reduce number of solutions we enforce that only one direction of reaction exist in enumerated CRNs knowing that seesaw reactions are always reversible); (d) that reactions have a left gate as a reactant (this is to prevent multiple redundant solutions, since all reactions are reversible we can enforce that left gate is always on the left hand side).

An instance generated by Alloy running the predicate with command `GenSeesaw` is $S_{ab} + LG_{bc} \rightarrow S_{bc} + RG_{ab}$, where S_{ab} and S_{bc} are strands, LG_{bc} left gate, RG_{ab} right gate, while left and right domains $\{a, b, c\}$ are denoted in subscript. Note that this reaction is equivalent to the one shown in Figure 2.

Listing 6 Seesaw model.

```
open strandsandgates

sig Domain {}
sig DNASpecies in Species { left, right: lone Domain }
sig RightGate, LeftGate extends Gate {}

fact UseAll { DNASpecies = Species and DNASpecies.(left + right) = Domain }
fact UniqueSpecies {
  all s1, s2: Strand | s1.left = s2.left and s1.right = s2.right implies s1 = s2
  all s1, s2: RightGate | s1.left = s2.left and s1.right = s2.right implies s1 = s2
  all s1, s2: LeftGate | s1.left = s2.left and s1.right = s2.right implies s1 = s2 }
fact OneDomain { all s: Strand + LeftGate + RightGate | one s.left and one s.right }

pred CanReactStrandAndLeftGate[s: Strand, lg: LeftGate] {
  s in Strand and lg in LeftGate and s.right = lg.left }
pred CanReactStrandAndRightGate[s: Strand, rg: RightGate] {
  s in Strand and rg in RightGate and s.left = rg.right }
pred CanReact[r1: DNASpecies, r2: DNASpecies] {
  CanReactStrandAndLeftGate[r1, r2] or CanReactStrandAndRightGate[r1, r2] }

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 CanReactStrandAndLeftGate[s, lg]
   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 CanReactStrandAndRightGate[s, rg]
   and s'.left = rg.left and s'.right = rg.right and lg'.left = s.left and lg'.right = s
       .right) }
pred React[r1: Species, r2: Species, p1: Species, p2: Species] {
  ReactStrandAndLeftGate[r1, r2, p1, p2] or ReactStrandAndRightGate[r1, r2, p1, p2] }

fun ReactantsSet[r: Reaction]: set Species { Int.(r.reactants) }
fun ProductsSet[r: Reaction]: set Species { Int.(r.products) }

pred Seesaw {
  StrandsAndGates[]
  all r: Reaction { -- All reactions are seesaw reactions.
    let s = 0.(r.reactants), g = 1.(r.reactants), s' = 0.(r.products), g' = 1.(r.
        products) {
      React[s, g, s', g'] }}
  all s1, s2: Species { -- All possible reactions exist.
    CanReact[s1, s2] implies some r: Reaction {
      (s1 + s2) = ReactantsSet[r] or (s1 + s2) = ProductsSet[r] }}
  all s1, s2: Species | all rxn1, rxn2: Reaction { -- Prevent reverse direction.
    ((s1+s2) = ReactantsSet[rxn1]) implies ((s1+s2) != ProductsSet[rxn2]) }
  all r: Reaction { some LeftGate & ReactantsSet[r] }
}

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

To reduce the enumeration overhead for seesaw, we updated the `Reaction` signature by removing the representation of reactants and products as a sequence (sequence introduces integers as an overhead), and adding two relations for reactants and products (as seesaw reactions are restricted to two reactants and two products). The updated `Reaction` signature is: `abstract sig Reaction { r1, r2, p1, p2:  Species }`

Listing 7 Feed-forward, non-competitive CRNs in Alloy.

```
open elementary

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

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

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

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

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

pred MustConsume[] {
  all r: Reaction | some s: Species | NetConsumes[r, s] }

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

Feed-forward, non-competitive CRNs. Listing 7 models feed-forward, non-competitive CRNs. Recall, we define feed-forward as: there exists a total ordering on the reactions such that no reaction consumes a species produced by a reaction later in the ordering. 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 reactions with respect to the feed-forward property. Observe that there can be multiple valid total orderings of reactions 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, our goal is to search for CRNs exhibiting desired functionality, and thus we aim to enumerate each CRN once, and as quickly as possible. To tackle this problem we achieve the total ordering by creating a graph of reaction dependencies, and enforce it to be *directed-acyclic*.

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 signature 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 `NonCompetitive` enforces that if a species is used as a reactant in a reaction then it cannot be consumed by any other reaction. The predicate `MustConsume` enforces that every reaction consumes some species (boundedness condition). The predicate `Feedforward` defines elementary, feed-forward, and non-competitive reactions where each reaction must consume some species.

Algorithm 1 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 CRN Enumeration and Search

In this section we describe our algorithm (shown in Algorithm 1) that performs a bounded exhaustive search enumerating all CRNs in a given class and within a given bounds respecting properties defined by an Alloy model, to find the CRN implementing desired function.

Inputs to the algorithm are the Alloy model, the size of CRNs (e.g., number of reactions and species) defined by the *scope*, desired target function f, and the number of inputs to the function N. Function *findAllInstances* accepts the 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 2 (Section 4) to check if CRN computes f. If CRN implementing given function is found then it is returned (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* [26, 28, 30] 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 1.

4 CRN Analysis

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 the each reaction, where $flux_i$ represents the flux of the reaction i. Then the concentration of a species S can be expressed in terms of its initial concentration S_0 and reaction fluxes:

$$s = s_0 + \sum_{i=1}^{N} netGain(rxn_i, S) \cdot flux_i \tag{1}$$

where $netGain(rxn_i, S)$ 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 Figure 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 $flux_i$ variables represent fluxes of reactions.

$$\begin{array}{ll} a = a_0 - flux_1 & b = b_0 - flux_2 \\ z_1 = z_{10} + flux_1 - flux_3 & z_2 = z_{20} + flux_2 - flux_3 \\ k = k_0 + flux_3 - flux_4 & y = y_0 + flux_1 + flux_2 - flux_4 \end{array} \tag{2}$$

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).[5] 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 concentrations of a set of species to zero, where 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 Figure 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 its species concentrations to zero and solve the system 2. This results in 4 solutions shown in 3 (we do not show solutions for flux variables due to the space limits).

$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{k}$	$\boldsymbol{y}$	$\boldsymbol{z_1}$	$\boldsymbol{z_2}$
0	0	$-b_0 + k_0 - y_0 + z_{10}$	0	0	$-a_0 + b_0 - z_{10} + z_{20}$
0	0	$-a_0 + k_0 - y_0 + z_{20}$	0	$a_0 - b_0 + z_{10} - z_{20}$	0
0	0	0	$b_0 - k_0 + y_0 - z_{10}$	0	$-a_0 + b_0 - z_{10} + z_{20}$
0	0	0	$a_0 - k_0 + y_0 - z_{20}$	$a_0 - b_0 + z_{10} - z_{20}$	0

(3)

Although there are 4 solutions, for any particular initial concentrations of the species only one of the solutions is non-negative (concentrations of species must be 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)[6]. 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.

[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.

[6] An input tuple (a,b) will be separately considered from (b,a). However, if the sought function is known to be commutative than the order of species can be ignored.

Algorithm 2 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 ≠ f(x_1, x_2, ..., x_N))
14:        if counterExample = null then return true
15:    end for
16:    return false
17: end procedure
```

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

a	b	k	y	z_1	z_2
0	0	$-b_0$	0	0	$-a_0+b_0$
0	0	$-a_0$	0	a_0-b_0	0
0	0	0	b_0	0	$-a_0+b_0$
0	0	0	a_0	a_0-b_0	0

(4)

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 $a_0 = 0 \wedge b_0 = 0$. 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 ($a_0 \geq 0 \wedge b_0 \geq 0$). 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 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 2. In step 2, conservation equations are constructed, while in step 3 we initialize a set of equilibrium solutions *equilibriumSolutions* to an 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,

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

	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	22	00:00:00	0	00:00:00	0	00:00:00
3 *Species*	6	00:00:00	199	00:00:00	287	00:00:00	0	00:00:00
4 *Species*	1	00:00:00	391	00:00:00	4,666	00:00:05	5,643	00:00:07
5 *Species*	0	00:00:00	291	00:00:00	17,509	00:00:19	140,064	00:03:57
6 *Species*	0	00:00:00	100	00:00:00	27,257	00:00:32	817,742	00:30:35

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*.

5 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.

Smallest *max* CRN. We perform bounded exhaustive search for 1 to 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 1 shows the number of enumerated CRNs and Alloy enumeration time for different scope sizes. We perform (not perfect) isomorphic breaking in Alloy by requiring lexicographic ordering on reactions among other things (details of symmetry breaking are shown in Appendix F). Note that while we perform some isomorphic breaking[7], not all isomorphic cases are pruned, and thus number of non-isomorphic instances may be less then numbers reported in Table 1. In spite of this, our approach is still exhaustive, meaning that all possible CRNs will be enumerated, but some may be enumerated multiple times. The first occurrence of *max* is found in the scope of 4 reactions and 6 species, and it was the $124,118^{th}$ instance Alloy enumerated in that scope. The CRN discovered is equivalent to the one shown in Figure 1, modulo reaction and species ordering.

[7] Alloy 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

$$X^+ \longrightarrow M + Y^+$$
$$M + X^- \longrightarrow Y^-$$

$$X^+ \longrightarrow Y^+ + C$$
$$X^- \longrightarrow Y^+ + E$$
$$C + E \longrightarrow 2\,Y^-$$

$$X_1^+ \longrightarrow M_1 + Y_{max}^+$$
$$X_1^- \longrightarrow M_2 + Y_{min}^-$$
$$X_2^+ \longrightarrow M_2 + Y_{max}^+$$
$$X_2^- \longrightarrow M_1 + Y_{min}^-$$
$$M_1 + M_2 \longrightarrow Y_{max}^- + Y_{min}^+$$

Figure 3 Minimal *ReLU* (left), *abs* (middle) and *minmax* (right) CRNs. (left) The *ReLU* CRN produces $x^+(0)$ amount of M and Y^+ by the first reaction. The second reaction produces $min(x^+(0), x^-(0))$ amount of Y^-. Thus, the amount of output produced is: $y = y^+ - y^- = x^+(0) - min(x^+(0), x^-(0))$ which can be shown to be equal to $ReLU(x^+(0) - x^+(0)) = ReLU(x)$. (middle) The *abs* CRN produces $x^+(0)$ amount of C and E by the first and second reactions, respectively, $x^+(0) + x^-(0)$ amount of Y^+, and $2min(x^+(0), x^-(0))$ amount of Y^-. Thus, $y = x^+(0) + x^-(0) - 2min(x^+(0), x^-(0)) = abs(x^+(0), x^-(0)) = abs(x)$.

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 a two species (e.g., the output value is equal to the concentration of species Y^+ minus that of Y^-).

An additional requirement for CRN modules is to be composable, in the sense that the output of one can be input to another. Note, for example, that the *max* system (Figure 1) is not composable because the downstream module might consume some amount of Y before it is consumed in its interaction with K (last reaction). Composability can be ensured if the output species are never consumed [9, 14, 52]. Note that consuming Y^+ is logically equivalent to producing Y^- (and vise versa for Y^-), and thus we restrict dual-rail computation in this way without losing expressibility.

Smallest *ReLU* CRN. Using the above described procedure we run experiments for finding the smallest CRN computing *ReLU* (rectified linear unit) function. We confirm that the CRN introduced in [58], which is shown in Figure 3, is indeed the smallest. Note that CRNs were already enumerated when searching for *max*, and that was no need to re-enumerate them as they were saved on disk.

Our analysis shows that the *ReLU* CRN is the smallest in the sense that there is no other CRN computing this function with fewer than 2 reactions or 5 species. In Appendix D we argue that our enumeration in Table 1 is sufficient to ensure that 5 species are necessary no matter how many reactions are allowed.

Smallest *abs* CRN. We conducted a similar experiment for finding the smallest CRN computing the absolute value function, finding CRN shown in Figure 3.

Smallest *minmax* CRN. *Minmax* CRN accepts two inputs and has two outputs, where one output computes *max*, and other output computes *min* of the inputs. Since species are in dual-rail form, there is 4 input and 4 output species. Thus, for *minmax* search we enumerated CRNs that have at least 8 species, where at least 4 species only appear as products (output species candidates), and at least 4 species which do not appear only as products (input species candidates). We have further restricted the CRNs to have a total of at most 16 reactants and products over all reactions. Enumeration results with those constraints are shown in Table 2 (isomorphic breaking is imperfect in this case as well). We discovered the

Table 2 Number of enumerated feed-forward, non-competitive CRNs with at least two dual-rail inputs (4 actual species) and two outputs (4 actual species). Star (*) denotes that the scope has been partially enumerated.

	2 *Reactions*		3 *Reactions*		4 *Reactions*		5 *Reactions*	
8 *Species*	1	00:00:00	1,176	00:00:03	67,323	00:03:09	0	00:00:00
9 *Species*	0	00:00:00	1,073	00:00:03	223,775	00:12:48	2,439,310	13:31:19
10 *Species*	0	00:00:00	385	00:00:02	328,397	00:19:30	4,669,000*	47:39:39

Table 3 Number of enumerated seesaw reactions with different number of domains and reactions, and up to 20 distinct species.

	1 *Reaction*		2 *Reactions*		3 *Reactions*		4 *Reactions*		5 *Reactions*	
1 *Domain*	1	00:00:00	0	00:00:00	0	00:00:00	0	00:00:00	0	00:00:00
2 *Domains*	1	00:00:00	4	00:00:00	0	00:00:00	2	00:00:01	1	00:00:03
3 *Domains*	1	00:00:00	5	00:00:00	15	00:00:01	13	00:00:05	14	00:00:17
4 *Domains*	0	00:00:00	9	00:00:01	33	00:00:02	92	00:00:18	121	00:01:58
5 *Domains*	0	00:00:00	4	00:00:00	55	00:00:04	243	00:00:48	705	00:10:16
6 *Domains*	0	00:00:00	1	00:00:00	43	00:00:10	436	00:06:40	2027	03:01:06

minimal *minmax* CRN, which is shown in Figure 3. We performed several optimizations to speed up the analysis phase which are described in Appendix E.

Seesaw enumeration. We enumerated all nonisomorphic seesaw CRNs up to specified bounds on the number of domains and reactions. Table 3 shows the number of enumerated CRNs restricted to 1-5 reactions, 1-6 domains, and up to 20 species. Since 5 seesaw reactions can have at most 20 distinct species this includes all possible seesaw CRNs in the scope of 1-5 reactions. For seesaw networks, we define isomorphic CRNs as those that can be obtained by: (a) swapping domain names, (b) changing order of reactants or products, (c) changing order of reactions, (d) swapping reactants with products (follows from the reversibility of seesaw reactions).

In order to check for isomorphisms while enumerating seesaw CRNs, we maintain a set of previously enumerated CRNs and all their isomorphisms. If a newly enumerated CRN is not found in the current set, we create the isomorphic class of the CRN by making all permutations of the CRN, and adding them to the set. Permutations are done only with respect to domains. Permuting the order of reactants and products, as well as swapping reactants and products, is not needed as we follow the convention of enumerating CRNs in a form $S_{??} + LG_{??} \leftrightarrow S_{??} + RG_{??}$. Permuting the order of reactions is not needed, as the set of CRNs is preserved as a hash table where a custom-made hash function is used for CRNs (a same hash value is returned for a CRN irrespective of the order of reactions). The isomorphic breaking is implemented as a post-processing step in Java. The run-times reported in Table 3 include both generation and isomorphic breaking times.

Note that we require that the CRN corresponding to a seesaw system contain all reactions that can occur. For illustration, we analyze seesaw CRNs with 2 domains and 1 reaction. Due to the reversibility of seesaw reactions we can limit our analysis to CRNs that have a left gate on the left hand side; thus our CRN will be of the form $S_{??} + LG_{??} \leftrightarrow S_{??} + RG_{??}$, where ? represent domains to be filled in. We denote two available domains with a and b,

and we enforce that both domains are used in a CRN. The possible combinations for the domains of the first strand are $\{aa, ab, ba, bb\}$, where we can remove cases starting with b as they are symmetrical. Choosing S_{aa} as a first strand, the only option for left gate is LG_{ab} as we have to use two domains and left domain of LG must match right domain of S. This leads to a CRN: $S_{aa} + LG_{ab} \leftrightarrow S_{ab} + RG_{aa}$. Note that this CRN is not a valid one, as in this case S_{aa} and RG_{aa} can also interact creating additional reaction. Another option for the strand is S_{ab}, in which case there are two options for left gate LG_{bb} and LG_{ba}. In a case of LG_{bb} reaction is following: $S_{ab} + LG_{bb} \leftrightarrow S_{bb} + RG_{ab}$. This is also not a valid CRN since S_{bb} and LG_{bb} can interact creating additional reaction. The final option is $S_{ab} + LG_{ba} \leftrightarrow S_{ba} + RG_{ab}$, which is only valid seesaw CRN in a case of 2 domains and 1 reaction; thus Table 3 shows count 1 for seesaw CRNs with 2 domains and 1 reaction.

Similarly, note that there are 0 CRNs with 2 domains and 3 reactions, but there are 2 with 2 domains and 4 reactions. This is due to the fact that all 3 reaction CRNs with 2 domains have some other species that can also interact producing additional (spurious) reaction. A curious reader can check that removing any reaction from 4 reaction 2 domain seesaw CRNs (Table 4) will leave some species that can interact creating the fourth reaction.

Table 4 Seesaw CRNs with 2 domains and 4 reactions.

$$S_{aa} + LG_{aa} \longleftrightarrow S_{aa} + RG_{aa}$$
$$S_{ba} + LG_{aa} \longleftrightarrow S_{aa} + RG_{ba}$$
$$S_{bb} + LG_{ba} \longleftrightarrow S_{ba} + RG_{bb}$$
$$S_{bb} + LG_{bb} \longleftrightarrow S_{bb} + RG_{bb}$$

$$S_{ab} + LG_{ba} \longleftrightarrow S_{ba} + RG_{ab}$$
$$S_{bb} + LG_{ba} \longleftrightarrow S_{ba} + RG_{bb}$$
$$S_{ab} + LG_{bb} \longleftrightarrow S_{bb} + RG_{ab}$$
$$S_{bb} + LG_{bb} \longleftrightarrow S_{bb} + RG_{bb}$$

6 Related Work

CRN Enumeration. Deckard et al. [18] 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 [45]. 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 [56]. 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 [62] discovers the smallest elementary CRN with bistability. Wilhelm and Heinrich [63] 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 [47] is close to ours in spirit, but focuses on discrete-state stochastic systems (integer molecular counts of the species), rate-dependent reactions, and does not guarantee that discovered CRNs are minimal. Cardelli et al [8] 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 [4, 17].

Computational power of CRNs. Much ongoing work has explored computational power of CRNs [31,43,51,59]. 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.

7 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 seesaw 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*, *minmax*, *abs* and *ReLU* functions in a natural subclass of rate-independent CRNs where rate-independence follows from structural network properties.

There remain a number of open questions that motivate future research directions. An important area of optimization is improving the run-time of the Alloy enumeration. Can we optimize the isomorphic breaking process to eliminate all isomorphisms? For improved efficiency and ease of use, do we need to rely on a separate tool like Mathematica to determine whether a given CRN computes the desired function, or can the necessary functionality be performed in Alloy alone? Finally, it remains to be seen how easily the techniques developed in this paper could be applied to rate-dependent computation.

References

1 Dana Angluin, James Aspnes, and David Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

2 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

3 Murad Banaji. Counting chemical reaction networks with NAUTY. *arXiv preprint arXiv:1705.10820*, 2017.

4 Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV*, 2011.

5 Gilles Bernot, Jean-Paul Comet, Adrien Richard, and Janine Guespin. Application of formal methods to biological regulatory networks: extending thomas' asynchronous logical approach with temporal logic. *Journal of theoretical biology*, 2004.

6 Luca Cardelli. Strand algebras for DNA computing. *Natural Computing*, 10(1):407–428, 2011.

7 Luca Cardelli. Morphisms of reaction networks that couple structure to function. *BMC systems biology*, 8(1):84, 2014.

8 Luca Cardelli, Milan Češka, Martin Fränzle, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, and Max Whitby. Syntax-guided optimal synthesis for chemical reaction networks. In *CAV*, 2017.

9 Cameron Chalk, Niels Kornerup, Wyatt Reeves, and David Soloveichik. Composable rate-independent computation in continuous chemical reaction networks. In *CMSB*, pages 256–273. Springer, 2018.

10 Ho-Lin Chen, David Doty, and David Soloveichik. Deterministic function computation with chemical reaction networks. *Natural computing*, 13(4):517–534, 2014.

11 Ho-Lin Chen, David Doty, and David Soloveichik. Rate-independent computation in continuous chemical reaction networks. In *Proceedings of the 5th conference on Innovations in theoretical computer science*, pages 313–326. ACM, 2014.

12 Yuan-Jyue Chen, Neil Dalchau, Niranjan Srinivas, Andrew Phillips, Luca Cardelli, David Soloveichik, and Georg Seelig. Programmable chemical controllers made from DNA. *Nature nanotechnology*, 8(10):755, 2013.

13 Kevin M Cherry and Lulu Qian. Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature*, 559(7714):370, 2018.

14 Ben Chugg, Anne Condon, and Hooman Hashemi. Output-oblivious stochastic chemical reaction networks. *arXiv preprint arXiv:1812.04401*, 2018.

15 Edmund M. Clarke, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model Checking*. MIT Press, 2018.

16 CRNs Exposed Github Page. URL: `https://github.com/marko-vasic/crnsExposed`.

17 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

18 Anastasia C Deckard, Frank T Bergmann, and Herbert M Sauro. Enumeration and online library of mass-action reaction networks. *arXiv preprint arXiv:0901.3067*, 2009.

19 Greg Dennis, Felix Sheng-Ho Chang, and Daniel Jackson. Modular verification of code with SAT. In *ISSTA*, 2006.

20 Niklas Een and Niklas Sorensson. An extensible SAT-solver. In *SAT03*, Santa Margherita Ligure, Italy, 2003.

21 Marcelo F. Frias, Juan P. Galeotti, Carlos G. López Pombo, and Nazareno M. Aguirre. DynAlloy: Upgrading Alloy with actions. In *ICSE*, 2005.

22 Juan P. Galeotti, Nicolás Rosner, Carlos G. López Pombo, and Marcelo F. Frias. TACO: efficient SAT-based bounded verification using symmetry breaking and tight bounds. *TSE*, 2013.

23 Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

24 Mirco Giacobbe, Călin C Guet, Ashutosh Gupta, Thomas A Henzinger, Tiago Paixão, and Tatjana Petrov. Model checking gene regulatory networks. In *TACAS*, 2015.

25 Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011.

26 Patrice Godefroid. VeriSoft: A tool for the automatic analysis of concurrent reactive software. In *CAV*, pages 476–479. Springer, 1997.

27 Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 2000.

28 Klaus Havelund and Thomas Pressburger. Model checking Java programs using Java pathfinder. *International Journal on Software Tools for Technology Transfer*, 2(4):366–381, 2000.

29 John Heath, Marta Kwiatkowska, Gethin Norman, David Parker, and Oksana Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 2008.

30 Gerard J Holzmann. *The SPIN model checker: Primer and reference manual*, volume 1003. Addison-Wesley Reading, 2004.

31 De-An Huang, Jie-Hong R. Jiang, Ruei-Yang Huang, and Chi-Yun Cheng. Compiling program control flows into biochemical reactions. In *Proceedings of the International Conference on Computer-Aided Design*, pages 361–368, 2012.

32 Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV*, 2017.

33 Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(2):256–290, 2002.

34 Daniel Jackson and Alan Fekete. Lightweight analysis of object interactions. In *TACS*, 2001.

35 Daniel Jackson, Ian Schechter, and Ilya Shlyakhter. ALCOA: The Alloy constraint analyzer. In *International Conference on Software Engineering*, Limerick, Ireland, June 2000.

36 Daniel Jackson and Mandana Vaziri. Finding bugs with a constraint solver. In *ISSTA*, 2000.

37 Eunsuk Kang, Aleksandar Milicevic, and Daniel Jackson. Multi-representational security analysis. In *FSE*, 2016.

38 Sarfraz Khurshid, Darko Marinov, and Daniel Jackson. An analyzable annotation language. In *ACM SIGPLAN Notices*, volume 37, pages 231–245. ACM, 2002.

39 Sarfraz Khurshid, Darko Marinov, Ilya Shlyakhter, and Daniel Jackson. A case for efficient solution enumeration. In *Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT)*, Santa Margherita Ligure, Italy, May 2003.

40 Matthew R Lakin, David Parker, Luca Cardelli, Marta Kwiatkowska, and Andrew Phillips. Design and analysis of DNA strand displacement devices using probabilistic model checking. *Journal of the Royal Society Interface*, 2012.

41 Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

42 Tong Ihn Lee, Nicola J Rinaldi, François Robert, Duncan T Odom, Ziv Bar-Joseph, Georg K Gerber, Nancy M Hannett, Christopher T Harbison, Craig M Thompson, Itamar Simon, et al. Transcriptional regulatory networks in saccharomyces cerevisiae. *Science*, 298(5594):799–804, 2002.

43 Marcelo O Magnasco. Chemical kinetics is Turing universal. *Physical Review Letters*, 78(6):1190, 1997.

44 Darko Marinov and Sarfraz Khurshid. TestEra: A novel framework for automated testing of Java programs. In *ASE*, pages 22–31, 2001.

45 Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, {II}. *Journal of Symbolic Computation*, 2014.

46 Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference (DAC)*, 2001.

47 Niall Murphy, Rasmus Petersen, Andrew Phillips, Boyan Yordanov, and Neil Dalchau. Synthesizing and tuning stochastic chemical reaction networks with specified behaviours. *Journal of The Royal Society Interface*, 15(145):20180283, 2018.

48 Jason Ptacek, Geeta Devgan, Gregory Michaud, Heng Zhu, Xiaowei Zhu, Joseph Fasolo, Hong Guo, Ghil Jona, Ashton Breitkreutz, Richelle Sopko, et al. Global analysis of protein phosphorylation in yeast. *Nature*, 438(7068):679, 2005.

49 Lulu Qian and Erik Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

50 Lulu Qian and Erik Winfree. A simple DNA gate motif for synthesizing large-scale circuits. *Journal of the Royal Society Interface*, 8(62):1281–1297, 2011.

51 Sayed Ahmad Salehi, Keshab K. Parhi, and Marc D. Riedel. Chemical reaction networks for computing polynomials. *ACS Synthetic Biology*, 6(1):76–83, 2017.

52 Eric E Severson, David Haley, and David Doty. Composable computation in discrete chemical reaction networks. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 14–23, 2019.

53 Shalin Shah, Jasmine Wee, Tianqi Song, Luis Ceze, Karin Strauss, Yuan-Jyue Chen, and John Reif. Using strand displacing polymerase to program chemical reaction networks. *Journal of the American Chemical Society*, 2020.

54 Ilya Shlyakhter. Generating effective symmetry-breaking predicates for search problems. In *Proc. Workshop on Theory and Applications of Satisfiability Testing*, June 2001.

55 David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences*, 107(12):5393–5398, 2010.

56 Carlo Spaccasassi, Boyan Yordanov, Andrew Phillips, and Neil Dalchau. Fast enumeration of non-isomorphic chemical reaction networks. In *CMSB*, pages 224–247. Springer, 2019.

57 Niranjan Srinivas, James Parkin, Georg Seelig, Erik Winfree, and David Soloveichik. Enzyme-free nucleic acid dynamical systems. *Science*, 358(6369):eaal2052, 2017.

58 Marko Vasic, Cameron Chalk, Sarfraz Khurshid, and David Soloveichik. Deep Molecular Programming: A Natural Implementation of Binary-Weight ReLU Neural Networks. In *International Conference on Machine Learning*, 2020.

59 Marko Vasic, David Soloveichik, and Sarfraz Khurshid. CRN++: molecular programming language. In *International Conference on DNA Computing and Molecular Programming*, pages 1–18. Springer, 2018.

60 Vito Volterra. *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. C. Ferrari, 1927.

61 Qinsi Wang, Paolo Zuliani, Soonho Kong, Sicun Gao, and Edmund M Clarke. Sreach: A probabilistic bounded delta-reachability analyzer for stochastic hybrid systems. In *CMSB*, 2015.

62 Thomas Wilhelm. The smallest chemical reaction system with bistability. *BMC systems biology*, 3(1):90, 2009.

63 Thomas Wilhelm and Reinhart Heinrich. Smallest chemical reaction system with hopf bifurcation. *Journal of mathematical chemistry*, 17(1):1–14, 1995.

64 David Yu Zhang and Georg Seelig. Dynamic DNA nanotechnology using strand-displacement reactions. *Nature chemistry*, 3(2):103, 2011.

A Proof of Rate Independence

In this section we develop an argument that the class of feed-forward, non-competitive CRNs as defined in the main text is rate-independent. For simplicity, we base our argument on the discrete CRN model, in which concentrations are integer molecular counts, reactions are discrete events (firings), and rate-independence corresponds to behaving correctly no matter what order the reactions occur in [10]. The continuous model is usually taken as an approximation of the discrete model.

Note that when we say that a species S is consumed by a reaction, we mean that it appears with negative net stoichiometry in the reaction. So we would not say that a catalyst is consumed. We define produced similarly. We say configuration d is reachable from c if there is a sequence of reactions that can fire to get from c to d.

In the main text, we define non-competitive as follows: if a species is consumed in a reaction then it cannot appear as a reactant somewhere else. Feed-forward is defined as follows: there exists a total ordering on the reactions such that no reaction consumes a species produced by a reaction later in the ordering. We also require that all reactions consume some species (boundedness condition).

Here we show that the feed-forward condition combined with boundedness implies that the CRN will always reach a static equilibrium. (A static equilibrium is one where no reaction can fire.) We then show that adding the non-competitive condition implies that the CRN always reaches the same static equilibrium independent of the order in which the reactions happen to occur.

The CRN always reaches some static equilibrium: If not then there is a set of reactions that can fire infinitely often. Choose the earliest (according to the ordering) reaction in this set. It must consume some S by boundedness. But by feed-forwardness, S can only be produced earlier in the ordering. Which means that the reactions that net produce S can only fire finite many times (they are not in this set). This is a contradiction.

The CRN always reaches the same static equilibrium: Toward a contradiction, suppose two different static equilibria c and d are reachable. Let p be the path to c and q be the path to d. Without loss of generality there are reactions that fire fewer times in p than in q. Let R be the reaction among these that comes earliest in the ordering. So compared to q, p has at least as many firings of reactions earlier in the ordering than R. By non-competitiveness, no other reaction consumes the reactants of R. Let S be a reactant of R. Consider two cases: (1) S is consumed in R. By feed-forwardness, S must be produced in a reaction earlier in the ordering than R. This means that the reactions producing S fire at least as much in p as in q. Since R fired fewer times in p than in q, there are some of S left in c. (2) S is not consumed in R (it acts as a catalyst). By the argument below, since R fires in q at least once, R fires in p at least once. Thus S is present in c. Combining (1) and (2), we have that R can fire in c, which contradicts the assumption that c is a static equilibrium.

There are no reactions that can fire on the path toward one static equilibrium but not fire on the path to another: Toward a contradiction, suppose two different static equilibria c and d are reachable. Let p be the path to c and q be the path to d. Let Ω be the set of reactions that fire in q but not in p. Let R be the reaction in Ω that occurs first (in time) in q. Its reactants must be either inputs or produced outside of Ω since R is the first reaction in Ω that fired in q. By non-competitiveness, the reactants of R cannot be consumed in any reaction other than R. So it must be possible to fire R at the end of p, which contradicts the assumption that p is a static equilibrium.

B Background: Alloy

The Alloy modeling language is a first-order logic with transitive closure [33]. The Alloy analyzer is a fully automatic tool for *scope-bounded* analysis of properties of Alloy models [35]. 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 [20] to analyze the model.

An Alloy model consists of a set of paragraphs where each paragraph declares some typed sets or relations, defines some logical constraints, or defines a command that informs the analyzer of the analysis to perform. Each command defines a constraint solving problem. and each solution to the problem defines an Alloy *instance*, i.e., a valuation of the sets and relations declared in the model such that the constraints with respect to the command are satisfied. The analyzer supports instance enumeration using incremental SAT solvers [20, 46]. In addition, the analyzer supports *symmetry breaking* and adds symmetry breaking predicates [54] to the original formula, which allows the backend SAT solvers to more effectively prune their search, and when enumerating solutions, create fewer solutions [39]. The analyzer's default symmetry breaking does not guarantee removal of all isomorphisms but is quite effective in practice.

C Autocatalytic Reactions

Similarly to catalytic reactions we model autocatalytic (Listing 8). Autocatalytic reactions add a requirement that in addition to existence of a catalyst species, the catalyst converts the other species into itself, for example: $X + Y \to Y + Y$.

Listing 8 Autocatalytic reactions.

```
module autocatalytic
open elementary
pred Autocatalytic[] { Elementary[] and all r: Reaction | AutocatalyticReaction[r] }
pred AutocatalyticReaction[r: Reaction] {
    some elems[r.reactants] & elems[r.products]
    eq[#r.products, 2] and eq[#elems[r.products], 1] }
```

D ReLU Minimality

In this section we argue that our enumeration in Table 1 is sufficient to ensure that 5 species are necessary for computing $ReLU$ no matter how many reactions are allowed.

Because with 4 species there are at most 2 different reactions possible (which we enumerate). Consider the $ReLU$ CRN with 4 species. This CRN must consist of 2 input species (X^+ and X^-) and 2 output species (Y^+ and Y^-), which we require to be distinct. Further, the output species have to appear only as products. Thus, only species X^+ and X^- can appear as reactants. Due to the requirement that every reaction has to net consume some species (Listing 7), and that different reactions have to consume different species (non-competitiveness), it follows that the CRN can have at maximum 2 reactions, one net consuming X^+, and other X^+ species. Considering that our technique did not discover any $ReLU$ CRN with 2 reactions and 4 species, we conclude that there is no $ReLU$ computing CRN with 4 species.

E Optimizing Analysis

In this section we explain how we optimize the analysis phase of search for *minmax* CRN.

The optimization is done by including tests. Instead of invoking *FindInstance* SMT solver for every combination of inputs and outputs, we construct a set of concrete test cases. If a test case fails we immediately discard that combination and move to the next one. This optimization improved analysis from 75s to 7.3s measured on the discovered *minmax* CRN. Furthermore from equality $|max(a,b)| + |min(a,b)| = min(|a|,|b|) + max(|a|,|b|)$, we first checked for CRNs that sattisfy this condition (using tests and FindInstance), and only run the check whether output species compute min and max on those. Checking for the above equality speeded up analysis becase the equality does not depend on the order of output species $y1$ and $y2$, thus reducing number of input output combinations that need to be tried. After implementing this additional optimization step analysis time went down to 0.75s measured on the discovered *minmax* CRN. The optimizations made it feasible to discover the *minmax* CRN.

F Symmetry breaking

This section shows our Alloy model for symmetry breaking of CRNs (Listing 9).

The Alloy analyzer during its translation from Alloy to propositional formulas automatically adds to the propositional formulas *symmetry breaking* predicates, which reduce the number of isomorphic solutions [54]. However, this automatic support is not practical for breaking all isomorphisms since there is a delicate trade-off between the complexity of the predicates that are added and the time it takes for the back-end solvers to handle them.

We follow a more effective approach where additional constraints *in Alloy* are mechanically added directly to the Alloy model [39]. The key idea is to define a linear order on the atoms and require that any solution when scanned in a pre-defined manner contains the atoms in

conformance with the linear order. The approach breaks all symmetries for rooted, edge-labeled graphs. However, CRNs represent a more complex structure and the approach does not guarantee breaking all symmetries. Nonetheless, it removes many isomorphic solutions and provides us a practical tool for exploring CRNs.

Note that the symmetry breaking is focused on a case of elementary CRNs as those CRNs are our focus group (all of our inherited CRN models are subclass of elementary).

Listing 9 Alloy modeling of CRN symmetry breaking.

```
module symmetry

open elementary

open util/ordering[Species] as Sordering
open util/ordering[Reaction] as Rordering

pred CheckFirstReaction {
  let first = Rordering/first,
      r1 = 0.(first.reactants), r2 = 1.(first.reactants),
      p1 = 0.(first.products), p2 = 1.(first.products)
  {
    r1 = Sordering/first
    r2 in r1 + r1.next
    p1 in r1 + r2 + (r1 + r2).next
    p2 in r1 + r2 + p1 + (r1 + r2 + p1).next
  }
}

pred CheckNonFirstReaction() {
  all r: Reaction - Rordering/first {
    let prevRxns = Rordering/prevs[r],
        prevSpecies = Int.(prevRxns.reactants + prevRxns.products),
        r1 = 0.(r.reactants), r2 = 1.(r.reactants),
        p1 = 0.(r.products), p2 = 1.(r.products)
    {
      r1 in prevSpecies + prevSpecies.next
      r2 in prevSpecies + r1 + (prevSpecies + r1).next
      p1 in prevSpecies + r1 + r2 + (prevSpecies + r1 + r2).next
      p2 in prevSpecies + r1 + r2 + p1 + (prevSpecies + r1 + r2 + p1).next
    }
  }
}

pred OrderReactionsBySize() {
  all disj r1, r2 : Reaction {
    Rordering/lt[r1, r2] implies {
      lt[#r1.reactants, #r2.reactants]
      or (eq[#r1.reactants, #r2.reactants]
           and lte[#r1.products, #r2.products])
    }
  }
}

pred ReactionsSameSize[r1, r2: Reaction] {
  eq[#r1.reactants, #r2.reactants]
    and eq[#r1.products, #r2.products]
}

pred CheckLexicographic() {
  all r: Reaction - Rordering/first {
    let p = r.prev,
    rr1 = 0.(r.reactants), rr2 = 1.(r.reactants), rp1 = 0.(r.products), rp2 = 1.(r.products),
    pr1 = 0.(p.reactants), pr2 = 1.(p.reactants), pp1 = 0.(p.products), pp2 = 1.(p.products)
    {
       ReactionsSameSize[r, p] implies {
        // DO only if sizes are the same assuming the size constraing.
        rr1 in pr1.*next
        rr1 = pr1 implies (no pr2 or rr2 in pr2.*next)
        (rr1 = pr1 and rr2 = pr2) implies (rp1 in pp1.*next)
        (rr1 = pr1 and rr2 = pr2 and rp1 = pp1) implies (no pp2 or rp2 in pp2.*next)
       }
    }
  }
```

```
  all r: Reaction {
    let r1 = 0.(r.reactants), r2 = 1.(r.reactants), p1 = 0.(r.products), p2 = 1.(r.products)
    {
      some r1 and some r2 implies Sordering/lte[r1, r2]
      some p1 and some p2 implies Sordering/lte[p1, p2]
    }
  }
}

pred SymmetryBreaking {
  Elementary
  CheckFirstReaction
  CheckNonFirstReaction
  OrderReactionsBySize
  CheckLexicographic
}
```