# A General-Purpose CRN-to-DSD Compiler with Formal Verification, Optimization, and Simulation Capabilities

Stefan Badelt[1(✉)], Seung Woo Shin[1], Robert F. Johnson[1], Qing Dong[2], Chris Thachuk[1], and Erik Winfree[1(✉)]

[1] California Institute of Technology, Pasadena, USA
{badelt,winfree}@caltech.edu
[2] Stony Brook University, New York, USA

**Abstract.** The mathematical formalism of mass-action chemical reaction networks (CRNs) has been proposed as a mid-level programming language for dynamic molecular systems. Several systematic methods for translating CRNs into domain-level strand displacement (DSD) systems have been developed theoretically, and in some cases demonstrated experimentally. Software that facilitates the simulation of CRNs and DSDs, and that helps automate the construction of DSDs from CRNs, has been instrumental in advancing the field, but as yet has not incorporated the fundamental enabling concept for programming languages and compilers: a rigorous abstraction hierarchy with well-defined semantics at each level, and rigorous correctness proofs establishing the correctness of compilation from a higher level to a lower level. Here, we present a CRN-to-DSD compiler, `Nuskell`, that makes a first step in this direction. To support the wide range of translation schemes that have already been proposed in the literature, as well as potential new ones that are yet to be proposed, `Nuskell` provides a domain-specific programming language for translation schemes. A notion of correctness is established on a case-by-case basis using the rate-independent stochastic-level theories of pathway decomposition equivalence and/or CRN bisimulation. The "best" DSD implementation for a given CRN can be found by comparing the molecule size, network size, or simulation behavior for a variety of translation schemes. These features are illustrated with a 3-reaction oscillator CRN and a 32-reaction feedforward boolean circuit CRN.

## 1 Introduction

Toehold-mediated nucleic acid strand displacement has become a widely used technology to control and fine-tune the interactions of DNA and RNA molecules [8,22]. This contribution focuses on automated construction – compilation – of nucleic acid networks, to realize larger, dynamically more complex and precise algorithms using DSD. We use the abbreviation DSD for "domain-level strand displacement" as opposed to the more common notion of "DNA strand displacement", because all results presented in this work are using the *domain-level* abstraction and we do not analyse any *sequence-level* details.


R. Brijder and L. Qian (Eds.): DNA 23 2017, LNCS 10467, pp. 232–248, 2017.
DOI: 10.1007/978-3-319-66799-7_15

Domains are segments of a molecule with well defined properties. In the simplest case, we distinguish two types of domains: toehold domains and branch-migration domains. A DSD system contains many possible instances of those domains, where two domains can only bind if they are of the same type and complementary to each other. Toehold domains (short) bind reversibly, while branch-migration domains (long) bind irreversibly. Ensuring that these properties are fulfilled is something attributed to the *sequence-level*. This abstraction enables us to study nucleic acid reaction networks on a different level of detail, including rigorous proofs to guarantee the correctness of a domain-level compilation and simulations of DSD systems based on "typical" sequence-independent reaction rates. A correct domain-level network can then be compiled to either DNA or RNA sequences or combinations of different nucleic acids, as well as, hypothetically, other artificial polymers such as PNA sequences, or even proteins [1].

Formal CRNs are a natural language to formulate the intended dynamics of a nucleic acid network and therefore serve as the ideal input for a DSD compiler. We demonstrate automated translation of CRNs into DSD systems, as well as the formal verification and simulation using our compiler `Nuskell` (see Sect. 2). We show that there are many formally correct translations of particular CRNs, but that some types of CRNs are more efficiently implemented with different translation schemes [2–4,12,13,18,20]. Starting from CRNs highlights a fundamental difference from other existing compilers, e.g. `VisualDSD` [12], the most used software for DNA strand displacement design, that takes hand-crafted DSD modules as input in order to predict and verify their dynamics [10]. The DNA strand displacement compilers `Seesaw` [21] and `Piperine` [20] have each been developed for one experimentally tested/optimized translation scheme and translate digital circuits or bimolecular reactions respectively.

Formal CRNs themselves might be derived from higher-level languages such as digital-circuits, Turing machines, etc. [5,17,18]. A demonstration is shown in Sect. 3.3, where we present a translation from a logic circuit into a formal CRN, and then use `Nuskell` to compile this CRN into a DSD circuit. The DSD implementation is pathway decomposition equivalent [16] (see Sect. 2.3) with the input CRN.

## 2 Nuskell

The CRN-to-DSD compiler `Nuskell` is an open-source `Python` package[1] for the design, verification and analysis of DSD systems. Figure 1 provides an overview of the `Nuskell` compiler project. The translation from CRNs to DSD systems is described in Sect. 2.1, the domain-level reaction enumeration using the `peppercornenumerator`[2] library [7] in Sect. 2.2 and the two notions of stochastic trajectory-type CRN equivalence (pathway equivalence [16] and CRN bisimulation equivalence [9]) in Sect. 2.3.

[1] www.github.com/DNA-and-Natural-Algorithms-Group/nuskell.
[2] www.github.com/DNA-and-Natural-Algorithms-Group/peppercornenumerator.

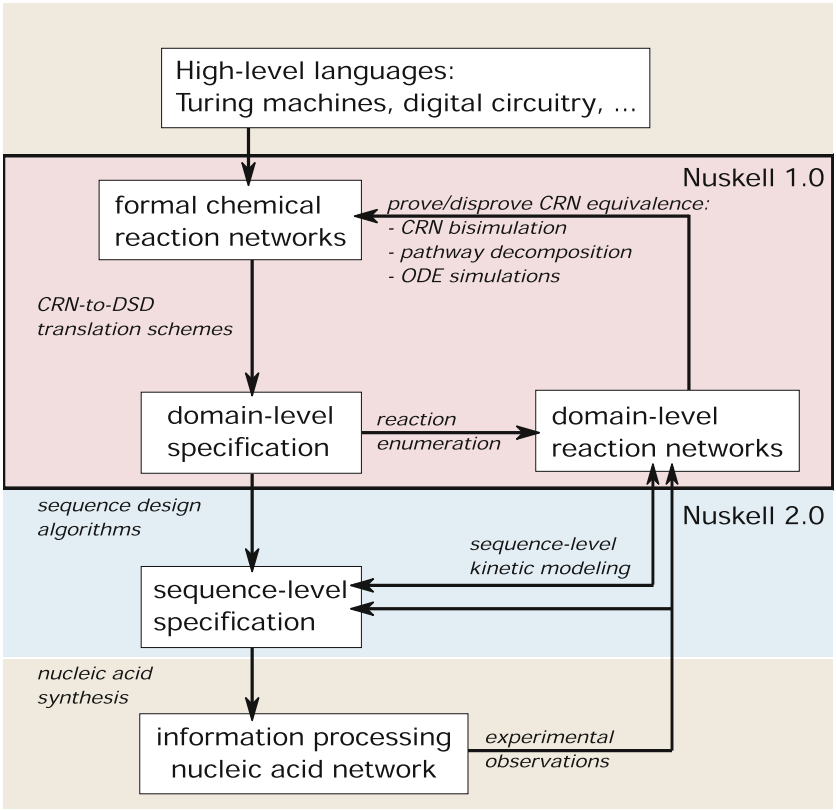



**Fig. 1.** The `Nuskell` compiler. **The current version** "`Nuskell 1.0`" translates formal CRNs into a set of domain-level nucleic acid complexes. The algorithm for translation can be chosen from multiple different CRN-to-DSD translation schemes (see Sect. 2.1). Domain-level complexes are input for a DSD reaction network enumerator (see Sect. 2.2). Two CRN equivalence notions (see Sect. 2.3) can be used to formally verify the equivalence between the domain-level reaction network and the formal CRN. Alternatively, initial complex concentrations can be specified to simulate the formal and/or enumerated CRN using ODEs. Domain-level specifications may be imported or exported to a plain-text format (*.pil) for manual adjustments or as an alternative to translation schemes. **The next version** "`Nuskell 2.0`" will translate *correct* domain-level specifications into sequence-level specifications and use sequence-level kinetic models to verify the correct implementation of domain-level reaction networks. **Eventually**, the `Nuskell` project may incorporate experimental feedback to train domain-level and sequence-level biophysics.

### 2.1 Translation: From a CRN to DSD Species

A multitude of translations from formal reactions into DSD systems have been shown previously [2–4,11–13,18,20], and there are many more possible variations. `Nuskell`'s CRN-to-DSD translation is a *top-down* approach. First, one has

to conceptualize a domain-level design in terms of an algorithm and then apply it to a formal input CRN. From this point on, we call a *translation scheme* a script written in the `Nuskell` programming language, which takes an input CRN and produces a DSD system. The language provides an "easy" formulation of translation schemes, and, more importantly, a standardized format for comparison, evaluation and debugging. This approach is in contrast to the *bottom-up* language of `VisualDSD`, where the user prototypes domain-level complexes as individual modules and combines them into a DSD system. In the bottom-up approach, it is not obvious whether a particular DSD implementation or its components can be generalized to implement different algorithms or whether conceptually new modules are required.

```
# Translate formal reactions with two reactants and two products.
# Lakin et al. (2012) "Abstractions for DNA circuit design." [Fig. 5]

# Define a global short toehold domain
global toehold = short();

# Define domains and structure of signal species
class formal(s) = "? t f" | ". . ."
  where { t = toehold; f = long() };

# Define fuel complexes for bimolecular reactions
class bimol_fuels(r, p) =
  [ "a t i + b t k + ch t c + dh t d + t* dh* t* ch* t* b* t* a* t*"
  | "( ( . + ( ( . + (  ( . + (  ( . + )   )  )   )  )  )  )  .",
    "a t i" | ". . .", "t ch t dh t" | ". . . . ." ]
  where {
    a = r[0].f;
    b = r[1].f;
    c = p[0].f; ch = long();
    d = p[1].f; dh = long();
    i = long(); k = long();
    t = toehold };

# Module *rxn* applies the fuel production to every bimolecular reaction
module rxn(r) = sum(map(infty, fuels))
  where fuels =
    if len(r.reactants) != 2 or len(r.products) != 2 then
      abort('Reaction type not implemented')
    else
      bimol_fuels(r.reactants, r.products);

# Module *main* applies *rxn* to the crn
module main(crn) = sum(map(rxn, crn))
  where crn = irrev_reactions(crn);
```

**Listing 1.1.** An example of a translation scheme. The **formal** class defines a signal species for every formal species, here, consisting of three unpaired domains: a history domain, a global short domain and a unique long domain. The **main** module translates a CRN into a set of fuel complexes: the CRN is converted to irreversible reactions, every reaction translated into a set of fuel complexes and the sum over all sets returned by the main function.

A drawback of CRN-to-DSD translation schemes is that they require a particular DSD architecture. There are always two types of species involved: *signal species* and *fuel species*. Signal species are at low concentrations and they

represent the dynamical information, e.g. input and output species. Fuel species are at high (ideally constant) concentrations and they mediate the information transfer by consuming and/or releasing signal species. All signal species must have the same domain-level constitution and structure, and they must be independent of each other. After compilation, every signal species corresponds to one species in the formal CRN.

We provide a continuously growing translation scheme library online[3], and a basic example in Listing 1.1 (translating Fig. 5 of [12]). The `Nuskell` programming language is inspired by the functional programming language `Haskell` and provides DSD specific classes, functions and macros to generalize translations for arbitrary CRNs [16]. There are two required parts: (i) the *formal* class defines sequence and structure of signal complexes, (ii) the *main* module produces a set of fuel species from the input CRN.

In some translation schemes, multiple signal species can correspond to the same formal species [2,12,18,20]. These schemes make use of so-called *history domains*. A history domain is considered to be an inert branch-migration domain of a signal species, but it is unique to the reaction that has produced the signal species. Hence, multiple species that differ only by their history domains map to the same formal species. When writing a translation scheme, a history domain is a wildcard: '?'. Together with the remainder of the molecule, a species with a wildcard forms a regular-expression, matching every other species in the system that differs only by a single domain in place of '?'. `Nuskell` automatically replaces history domains *after* domain-level enumeration, when it is known which species actually got produced. If there exists a species matching the regular expression, then the species with the wildcard domain and every enumerated reaction involving that species is removed from the system, otherwise, the wildcard domain is replaced by a new branch-migration domain.

A given translation schemes may be particularly efficient for certain types of formal reactions but inefficient or incorrect for other types, or it can be correct for every possible formal CRN, typically at the cost of being less efficient. For example, some translation schemes are particularly efficient for reversible reactions, while others implement reversible reactions using two irreversible reactions. To bypass the strict DSD system architectures that are imposed by translation schemes, `Nuskell` also provides an import/export file format (*.pil) to modify DSD systems, add extra modules or analyse bottom-up, hand-crafted, or alternatively compiled domain-level designs. In order to verify CRN equivalence, users have to ensure that names of signal species in the DSD implementation match the formal species in the input CRN, potentially using wildcards to indicate history domains. Also, `Nuskell` can export DSD systems to the `VisualDSD` file format (*.dna), which enables convenient access to the functionality of `VisualDSD`, including visualization, alternative reaction enumeration semantics and verification using probabilistic model checking [10].

[3] http://www.github.com/DNA-and-Natural-Algorithms-Group/nuskell/schemes.

### 2.2 Reaction Enumeration and Reaction Rate Calculation

The domain-level representation provides a more coarse-grained perspective on nucleic acid folding than the single-nucleotide level. At the nucleotide level every step is a base pair opening or closing reaction and the corresponding rate can be calculated from the free energy change of a reaction combined with inherent kinetic rate constants [6,15]. On the domain level, we consider a more diverse set of reactions in order to account for the fine-grained details that can happen on the sequence level. `Nuskell` uses the domain-level reaction enumerator `Peppercorn` [7] to predict desired and, potentially, undesired reactions emerging from interactions between previously compiled signal and fuel species.

*Detailed Enumeration.* The general types of reactions are summarized in Fig. 2. In particular: spontaneous binding and unbinding of domains, 3-way branch migration, 4-way branch migration and remote toehold branch migration. These reactions have been identified as most relevant in DSD systems. A typical DSD reaction pathway (also shown in Fig. 2) is a sequence of these detailed reaction steps. `Peppercorn`'s enumeration semantics are justified based on the assumption that

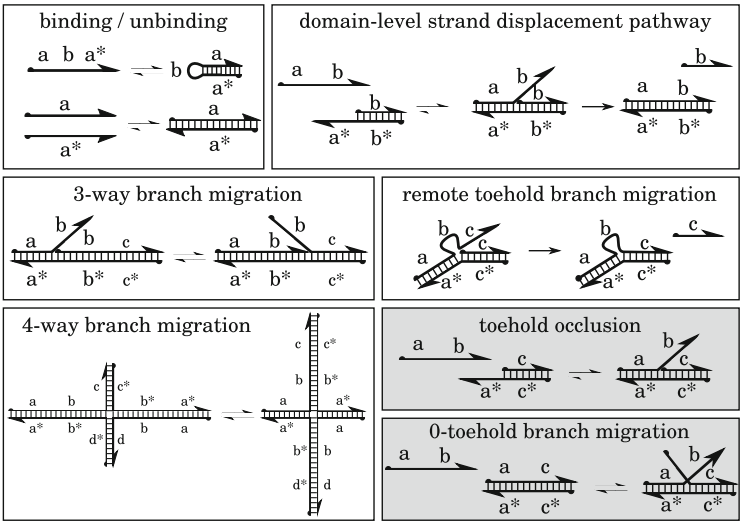



**Fig. 2.** Reaction semantics of the DSD enumerator `Peppercorn` [7]. Four generally supported detailed forms of reactions: intermolecular and intramolecular **binding/unbinding** of domains, **3-way branch migration**, **4-way branch migration** and **remote toehold branch migration**. A typical detailed **domain-level strand displacement pathway**. The condensed reaction network notion removes the intermediate (transient) complex and calculates one irreversible rate (see main text). **Toehold occlusion** describes the effect of toehold binding to a complementary domain that does not have the correct adjacent branch-migration domain. **0-toehold branch migration** is an invalid reaction in the `Peppercorn` semantics, but it is a well-known unintended *leak* reaction.

the DSD system is operated at sufficiently low concentrations, such that unimolecular reactions always go to completion before the next bimolecular interaction takes place. A number of enumeration constraints are implemented to avoid combinatoric explosions [7].

*Condensed Enumeration.* Under the assumptions of low concentrations, a *condensed* CRN can be calculated, with reactions that indicate just the eventual results after all unimolecular reactions complete, and with rate constants systematically derived from the detailed reaction network rate constants. Reaction condensation can drastically reduce the size of an enumerated network by removing reactions that do not result in stable resting states. A particular example, *toehold occlusion* is shown in Fig. 2. The reversible binding of a single toehold domain without the prospect of initiating branch migration is captured in the detailed reaction network, but not in its condensed form. For more details and subtleties, see [7].

*Limitations.* There are other forms of interactions which cannot be modelled using the presented set of reactions. Most prominently, every conformation in the DSD system has to be free of pseudoknots. That means every bound domain dissects the structure into an independent left and a right part, such that there are no base pairs connecting them. Also, initiation of 3-way branch-migration reactions requires at least one already bound domain. So-called 0-toehold branch-migration reactions (see Fig. 2) have been observed in practice due to partial unbinding at helix ends, but cannot be enumerated. They belong to the broader category of *leak* reactions which we faithfully ignore in the current version of the compiler.

*Reaction rates.* `Peppercorn` uses empirical domain-level reaction rates derived from DNA strand displacement and general DNA biophysics experiments. The domain-level reaction rate constants assume perfect Watson-Crick complementary of domains and "typical" sequences, as they only depend on the length and the type of a reaction. Detailed explanation on rates, as well as their justification compared with thermodynamic models can be found in [7], but it is important to emphasize that domain-level designs may choose from a range of realistic rates, which are here presented in a discrete form as typical for certain toehold and branch-migration domain lengths. Finding sequences that confirm these chosen rates constants and verifying them using stochastic sequence-level simulations is the responsibility of a sequence-level compiler. There are many mechanisms, such as small variations in toehold sequence composition, single-nucleotide mismatches, wobble base pairs, and non-canonical base pairs that can be exploited to fine-tune reaction dynamics.

### 2.3 Verification of DSD Reaction Networks

The most fundamental requirement towards compilation of large scale DSD systems is verification. Every *formal reaction* is translated into multiple *implementation reactions*. Thus, there are many possibilities for introducing "bugs",

i.e. unwanted side reactions that alter the implemented algorithm. We present two case-by-case verification strategies that compare formal CRNs with their implementations. As intended, our approach does not verify the general correctness of a particular scheme, but the correctness of a particular implementation.

*Pathway Decomposition Equivalence.* This notion was introduced in [16] together with an early version of the `Nuskell` compiler. The core idea is to represent each implementation trajectory as a combination of independent pathways of reactions between formal species. Pathway decomposition yields a set of pathways which are indivisible (or *prime*) and are called the *formal basis* of a CRN. The formal basis is unique for any valid implementation. Any two CRNs are said to be equivalent if they have the same formal basis. Conveniently, a CRN without intermediate species has itself as the formal basis, and it is worth pointing out that this equivalence relation allows for the comparison of one implementation with another implementation.

A common artifact of incorrect CRN-to-DSD translations is that intermediate species accumulate. That means the implementation network produces intermediate species, but they do not get cleaned up after a formal reaction goes to completion. In the notion of pathway equivalence, a given implementation is *tidy* if all intermediate species are cleaned up after a formal reaction goes to completion, and not tidy otherwise. The pathway decomposition verification method removes fuel species and inert waste products before equivalence testing, the compiler distinguishes formal from intermediate species.

*CRN Bisimulation Verification.* A CRN bisimulation [9] is an *interpretation* of the implementation CRN, where every implementation species is mapped to a multiset of formal species. This often yields so-called *trivial* reactions, where reactants and products do not change according to the interpretation. An interpretation is only a bisimulation if three conditions are fulfilled: (i) *atomic condition* – for every formal species there exists an implementation species that interprets to it, (ii) *delimiting condition* – any reaction in the implementation is either trivial or a valid formal reaction, and (iii) *permissive condition* – for any initial condition in the implementation CRN, the set of possible next non-trivial reactions is exactly the same as it would be in the formal CRN. CRNs are said to be bisimulation equivalent, if the translation can be interpreted as an implementation of that formal CRN.

Bisimulation does not require any upfront information of which signal species correspond to formal species. In fact an implementation can be bisimulation equivalent without the intended correspondence between signal and formal species. For this reason, `Nuskell` provides a mapping from signal to formal species as a *partial interpretation* upfront, guaranteeing that the species are interpreted as intended, and also guaranteeing that the atomic condition is fulfilled.

*Differences of Equivalence Notions.* In most cases of practical interest, pathway decomposition and CRN bisimulation agree. However, it is worth pointing out examples where pathway decomposition and CRN bisimulation disagree.

First, note that pathway decomposition theory was intended to be applied to translation schemes that implement reversible reactions as two independent irreversible reaction pathways; it generally does not handle schemes that provide a single reversible implementation of each reversible reaction. For example, consider the following implementations using the scheme presented in [13]:

| $A + B \rightleftharpoons C + D$ | $A + B \rightleftharpoons B + C$ | $A + B \rightleftharpoons C + B$ |
|---|---|---|
| $A \rightleftharpoons i1$ | $A \rightleftharpoons i1$ | $A \rightleftharpoons i1$ |
| $B + i1 \rightleftharpoons i2$ | $B + i1 \rightleftharpoons i2$ | $B + i1 \rightleftharpoons i2$ |
| $i2 \rightleftharpoons C + i3$ | $i2 \rightleftharpoons B + i3$ | $i2 \rightleftharpoons C + i3$ |
| $i3 \rightleftharpoons D$ | $i3 \rightleftharpoons C$ | $i3 \rightleftharpoons B$ |
| not pathway equivalent<br>bisimulation equivalent | pathway equivalent<br>bisimulation equivalent | not pathway equivalent<br>bisimulation equivalent |

It is easy to see that all CRNs are bisimulation equivalent, e.g. the interpretation $\{i1 = \{A\}; i2 = \{C, D\}; i3 = \{D\}\}$ is a valid bisimulation of $A + B \rightleftharpoons C + D$. However, the first example is not pathway equivalent, because the species $C$ can be produced and then reverse without producing $D$. This form of *prematurely generated or consumed species* is forbidden in pathway equivalence, because it is problematic for implementations of irreversible reactions. In the second example of a catalytic reaction, this effect is not present, because the catalyst last and producing it first. Changing this order of reactants makes the two CRNs pathway inequivalent. On the other hand, bisimulation demands an interpretation of every species in terms of a formal species. A particularly relevant example is the *delayed choice* phenomenon [16]. Consider the formal CRN $\{A \to B; A \to C\}$ and its implementation $\{A \to i; i \to B; i \to C\}$. The two CRNs are clearly pathway equivalent, but bisimulation cannot interpret $i$ such that both formal reactions are possible. Taken together, although both pathway decomposition and bisimulation capture the majority of intuitive equivalence relations, particular forms of very efficient implementations, or shortcuts might result in differences between the notions.

## 3 Case Studies

This section provides a glimpse into the future of automated DSD circuit design. We discuss potential problems of translation schemes, optimization strategies, and compare different schemes for a small oscillating CRN. Last but not least, we demonstrate the correct compilation of a large CRN implementing a digital circuit.

### 3.1 The Effects of Network Condensation (and Toehold Occlusion)

The intention behind network condensation is primarily to reduce the size of enumerated reaction networks. This makes verification methods, which often scale

poorly with CRN size, more likely to be computationally tractable. However, here we use network condensation to study the effects of *toehold occlusion*, i.e. an effect where complementary toeholds bind "unintentionally" without actually triggering strand displacement reactions (see Fig. 2). Toehold occlusion is believed to influence the dynamics of a DSD system [14,20], especially in schemes where the consumption of fuels results in accumulation of waste species with accessible toeholds.

We start with compiling an oscillator CRN with a translation scheme that has recently been able to confirm DNA oscillations experimentally [20]. The CRN is composed of three autocatalytic reactions:

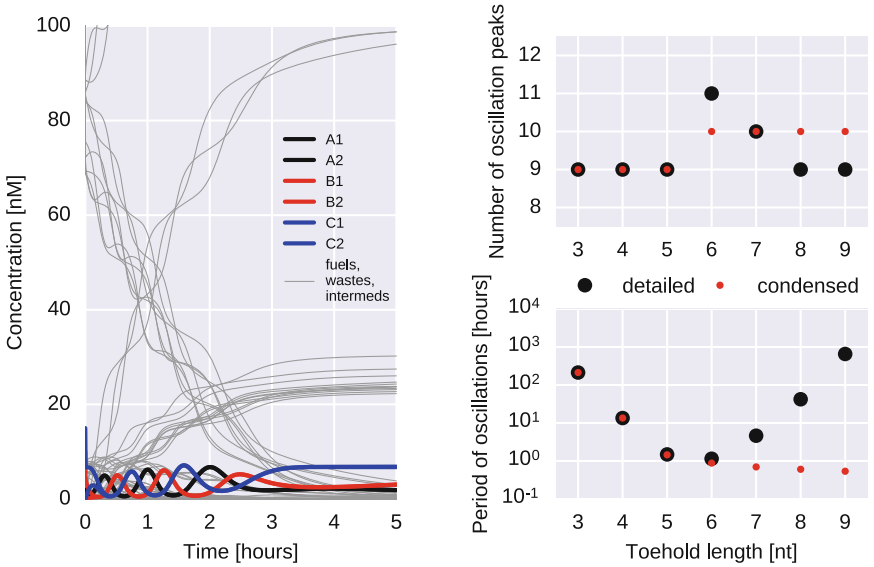
$$\begin{aligned} A + B &\rightarrow 2B \\ B + C &\rightarrow 2C \\ C + A &\rightarrow 2A \end{aligned} \tag{1}$$

As the scheme uses history domains that are unique to each reaction output and every formal species ($A$, $B$, $C$) is produced twice in the formal CRN, signal species exist in two versions: $A_1, A_2, B_1, B_2, C_1, C_2$. We simulate the system at the domain level using fuel concentrations at initially 100 nM and signal concentrations at $[A_1] = 2$ nM, $[B_1] = 10$ nM, $[C_1] = 15$ nM. Keeping the species at very low concentrations extends the number of oscillations, as fuel species get depleted more slowly.

Figure 3 shows data from multiple simulations, analysing the influence of reaction network condensation as a function of toehold length. We observed a roughly constant number of oscillations ranging from 9 to 11 peaks in total, across toehold lengths between 3–9 nt. The differences come from minor fluctuations when fuel species get depleted, i.e. the first 9 oscillation peaks are present across all examples. Hence, neither toehold length, nor reaction network condensation has a strong effect on the number of oscillation peaks.

The period of oscillations, however, changes drastically for chosen toehold lengths. At the typical lengths of 5–7 nt we observe the fastest oscillations according to the detailed reaction network. For this range of toehold lengths and concentations, binding and unbinding of toeholds occurs at a similar rate, which means (a) toeholds frequently bind to complementary sites and have enough time to initiate-branch migration and (b) toeholds bound to sites where branch migration cannot be initiated dissociate quickly. For shorter toeholds, both detailed and condensed enumerations agree, because toeholds dissociate at a high rate and the effects of toehold occlusion are insignificant. However, the fraction of toeholds completing branch migration is low, slowing down the oscillation period. For longer toeholds, detailed and condensed enumeration disagree. In the detailed network, toehold occlusion slows down the system, such that species cannot bind to their intended complementary regions. The condensed network does not simulate toehold-occlusion effects and therefore these networks oscillate faster with increased toehold length.

This result is particularly interesting because some translation schemes use a mechanism called *garbage collection* [2,3]. The intention is to collect waste




(a) ODE simulation: detailed network, 6 nt toeholds

(b) Simulations as a function of toehold length.

**Fig. 3.** Controlling the dynamics of a DSD oscillator via toehold length. Data compares an oscillator implemented using Srinivas' translation scheme [19], with initial conditions: $A_1 = 2$ nM, $B_1 = 10$ nM, $C_1 = 15$ nM. **(a)** Simulation of a detailed enumerated DSD reaction network. Black, red and blue lines correspond to the formal species A, B and C respectively. Note that there are two molecules with distinct history domains for each formal species. 9 oscillation peaks are clearly visible, starting with C (blue) and ending with B (red). These peaks are also present in all other simulations, independent of toehold length. However, there are actually two more hardly visible peaks for species $A$ and $C$ just before they reach equilibrium. **(b)** Number of oscillation peaks (top) and the period of oscillations (bottom) as a function of toehold length. Oscillation peaks are counted after the species with distinct history domains have been added (e.g. $A = A_1 + A_2$). The oscillation period for $n$ oscillation peaks is calculated as $3(t_n - t_1)/(n-1)$, where $t_n$ is the time point of the last oscillation peak. (Color figure online)

species with available toehold domains into inert complexes. Therefore, garbage collection introduces additional complexes and reactions to keep the computation speed of a DSD system constant. However, in practice, this makes systems larger and harder to verify. In a physical realization, it increases the synthesis cost as well as the possibilities for *leak* reactions, such that experimental realizations have so far refrained from these additional complexes [4]. Studying the differences of detailed vs. condensed reaction networks shows that, if one chooses the rates for toehold binding appropriately and with respect to intended concentrations, toehold occlusion is not a limiting issue for the presented system.

### 3.2 Comparing DSD Oscillator Translations

In Fig. 4 we compare implementations of the above oscillator CRN using 13 different translation schemes. All schemes compared here verified correct for a single autocatalytic reaction $A + B \rightarrow 2B$, according to at least one of the two equivalence notions. Figure 4b shows verification results of the full system, including potential cross reactions between the three autocatalytic reactions. The schemes are *generalized* versions to support $n$-arity of reactions, but use exclusively reaction mechanisms shown (or described) in the original publication. A *variant* differs from the originally published version, either to correct the original version, to generalize it in a form that was not obvious from the publication, or to make a modification that enhances the performance of the scheme.

Figure 4 compares the size of the condensed enumerated network and the number of nucleotides in a system. The number of nucleotides is an indicator for the synthesis cost of nucleic acid sequences, calculated as the combined length of all distinct strands. The size of the implementation network is an indicator of computation efficiency, calculated as number of irreversible implementation reactions (i.e. reversible reactions are two irreversible reactions) in the condensed reaction network.

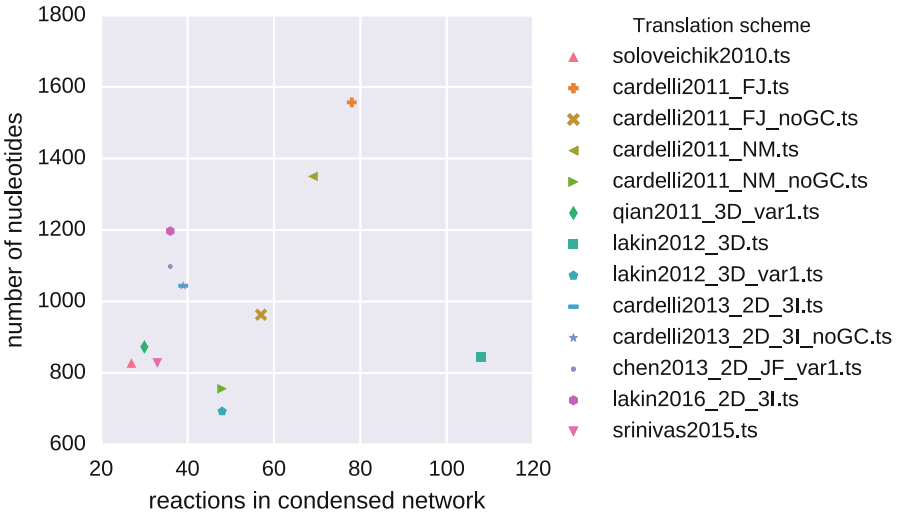
The implementations range from 27 to 108 reactions in the condensed enumerated CRN and from 693 to 1557 nucleotides. Obviously, removing garbage collection complexes reduces the total number of nucleotides as well as the number of reactions. The largest system in terms of reactions is `lakin2012_3D.ts`. A simple modification in `lakin2012_3D_var1.ts`: removing inert domains of strands that reverse the consumption of input strands, makes them independent of the implemented formal reaction and reduces the numbers to 48 reactions and 693 nucleotides. The 2-domain scheme `cardelli2013_2D_3I.ts` is implemented with the 3-domain irreversible step as suggested in the publication [3]. This scheme is particularly optimized for autocatalytic reactions such that they do not require extra garbage collection complexes and reactions. Hence, `cardelli2013_2D_3I.ts` and `cardelli2013_2D_3I_noGC.ts` both return the same set of fuel species for this CRN.

### 3.3 Towards Compilation of Large CRNs

We now demonstrate the domain-level implementation of larger systems. Our test case, adapted from [14], is a dual-rail implementation of a logic circuit computing the floor of the square root of a 4-bit binary number:

$$y_2 y_1 = \lfloor \sqrt{x_4 x_3 x_2 x_1} \rfloor \tag{2}$$

First, the logic circuit was translated into a CRN that consists of 32 uni- and bimolecular reactions (see Fig. 5a), second, the CRN was compiled using `Nuskell` with the scheme `soloveichik2010.ts` [18]. The condensed enumerated reaction network has 316 species (52 signal species, 92 fuel species, 172

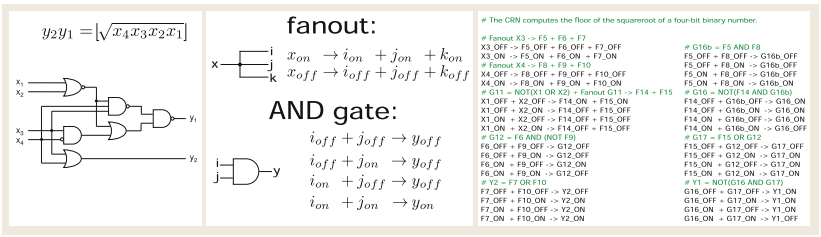



(a) Efficiency of translations schemes

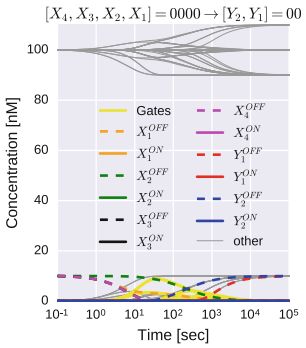
| Translation scheme | pathway equivalent | bisimulation equivalent |
|---|---|---|
| `soloveichik2010.ts` | True | True |
| `cardelli2011_FJ.ts` | False | - |
| `cardelli2011_FJ_noGC.ts` | False | - |
| `cardelli2011_NM.ts` | - | - |
| `cardelli2011_NM_noGC.ts` | False | True |
| `qian2011_3D_var1.ts` | True | True |
| `lakin2012_3D.ts` | False | - |
| `lakin2012_3D_var1.ts` | False | True |
| `cardelli2013_2D_3I.ts` | False | - |
| `cardelli2013_2D_3I_noGC.ts` | - | - |
| `chen2013_2D_JF_var1.ts` | - | - |
| `lakin2016_2D_3I.ts` | - | - |
| `srinivas2015.ts` | True | True |

(b) Verification of translation schemes

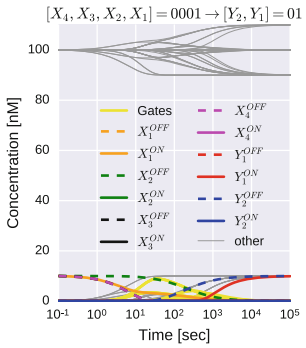
**Fig. 4.** DSD oscillator implemented using 13 different translation schemes. A "`_noGC`" indicates that the scheme differs from the published version in that it does not implement garbage collection reactions. Other variants are indicated by "`_var`". `lakin2012_3D.ts` produces identical complexes with the scheme presented in Listing 1.1 for this input CRN. **(a)** The plot shows the total length of all distinct strands in the circuit as an indicator of synthesis cost, and the size of the enumerated reaction network as indication of computation speed. The number of nucleotides is calculated assuming 6 nt toeholds and 15 nt branch-migration domains. **(b)** A table summarizing the results of verification. None of the schemes was shown incorrect by both equivalence notions, but many reaction networks are too complicated, such that equivalence testing did not terminate within 1 hour.



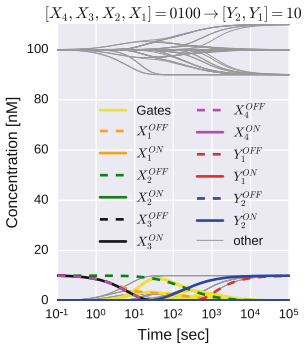

(a) from digital circuit to a feedforward CRN



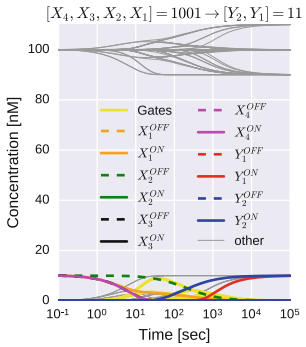

(b) $\sqrt{0} = 0$




(c) $\sqrt{1} = 1$




(d) $\sqrt{4} = 2$




(e) $\sqrt{9} = 3$

**Fig. 5.** A CRN calculating the square root of a 4-bit binary number compiled to a DSD system using the translation scheme presented in [18]. **(a)** A digital circuit taken from [14] is translated following the rules shown for fanouts and AND gates. NOT and OR gates follow the same principle, and the three-input AND gate is translated using two two-input AND gates. **(b-e)** The four simulations show the results for inputs $0, 1, 4, 9$. Input signal species are called $X$, output signal species called $Y$, all other signal species are "Gates", which are only transiently produced. All signal species exist in an ON and OFF version which is either initially present (10 nM) or absent (0 nM). Fuel species are initially at 100 nM.

intermediate species), 180 reactions, and it verifies as correct according to the pathway equivalence notion.

Figure 5 shows the simulations for four calculations: $0000, 0001, 0100, 1001$. Every input and output digit is represented by one ON and one OFF species, which are either initially present (10 nM) or absent (0 nM). The remaining 40 signal species (with unique history domains) represent the 12 transient formal species in the formal CRN, also called "Gates" in Fig. 5. The fuel species are initially at 100 nM; some of them are consumed during the DSD calculations, while others become more abundant. The reaction rates as calculated by the `peppercornenumerator` library suggest the completion of the DSD circuit after approximately 27 h, which is comparable to the computation time using the Seesaw architecture [14], with respect to the lower concentrations of initial species in this example.

Both enumeration and verification can be bottlenecks to compile large systems. For example, we have tested other techniques to translate digital circuits into CRNs with trimolecular reactions, where the enumerator had difficulties to deal with the combinatorial explosion of intermediate species due to history domains.

## 4 Conclusions

The strength of `Nuskell` comes from three features: First, formulating DSD design principles as translation schemes makes the design and optimization of complex networks easily accessible to a broad scientific community. Second, rigorous proofs of correctness guarantee a successful domain-level compilation, and are applied on a flexible case-by-case basis. Third, multiple translation schemes can be compared for a given CRN, exploiting the diversity of DSD circuits implementing the same CRN and allowing for optimization of circuits at the domain level, before proceeding to the computationally more expensive DNA sequence-level design and verification.

The discussed verification methods ensure that – given a particular reaction semantics – a CRN is correctly translated. Variations of these enumeration semantics are sometimes necessary and can help to identify problems. For example, some schemes are only correct if remote toehold branch migration or 4-way branch-migration reactions are disabled, which reveals clues for identifying unintended side reactions and making schemes more robust. On the other hand, ODE simulations of domain-level systems can be used to compare the performance of schemes, e.g. in terms of oscillation periods or fuel consumption; some schemes can be technically incorrect but with a probability of error that decreases with molecular counts in the stochastic regime and disappears entirely in the large-volume deterministic regime. Future versions of the compiler might calculate leak reaction rates to fine-tune the length of particular domains and to more efficiently combat leak during nucleic acid sequence design.

There are many open questions about the limitations of the *algorithmic behavior* that can be programmed into nucleic acid systems. How complex can

DSD systems get? Does efficiency decrease with a larger number of reactions? Can particularly efficient translation schemes be combined? Compilers can be used to study and optimize DSD systems in order to reveal their full potential.

**Acknowledgements.** We thank the U.S. National Science Foundation for support: NSF Grant CCF-1213127 and NSF Grant CCF-1317694 ("The Molecular Programming Project"). The Gordon and Betty Moore Foundation's Programmable Molecular Technology Initiative (PMTI). SB is funded by the Caltech Biology and Biological Engineering Division Fellowship. SWS's current address is Google, Mountain View, California. QD's current address is Epic Systems, Madison, Wisconsin.

## References

1. Boyken, S.E., Chen, Z., Groves, B., Langan, R.A., Oberdorfer, G., Ford, A., Gilmore, J.M., Xu, C., DiMaio, F., Pereira, J.H., et al.: De novo design of protein homo-oligomers with modular hydrogen-bond network-mediated specificity. Science **352**(6286), 680–687 (2016)
2. Cardelli, L.: Strand algebras for DNA computing. Nat. Comput. **10**(1), 407–428 (2011)
3. Cardelli, L.: Two-domain DNA strand displacement. Math. Struct. Comput. Sci. **23**(02), 247–271 (2013)
4. Chen, Y.J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from DNA. Nat. Nanotechnol. **8**(10), 755–762 (2013)
5. Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Condon, A., Harel, D., Kok, J., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses. Natural Computing Series, pp. 543–584. Springer, Heidelberg (2009)
6. Flamm, C., Fontana, W., Hofacker, I.L., Schuster, P.: RNA folding at elementary step resolution. RNA **6**, 325–338 (2000)
7. Grun, C., Sarma, K., Wolfe, B., Shin, S.W., Winfree, E.: A domain-level DNA strand displacement reaction enumerator allowing arbitrary non-pseudoknotted secondary structures. arXiv:1505.03738 (2014)
8. Hochrein, L.M., Schwarzkopf, M., Shahgholi, M., Yin, P., Pierce, N.A.: Conditional dicer substrate formation via shape and sequence transduction with small conditional RNAs. J. Am. Chem. Soc. **135**(46), 17322–17330 (2013)
9. Johnson, R.F., Dong, Q., Winfree, E.: Verifying chemical reaction network implementations: a bisimulation approach. In: Rondelez, Y., Woods, D. (eds.) DNA 2016. LNCS, vol. 9818, pp. 114–134. Springer, Cham (2016). doi:10.1007/978-3-319-43994-5_8
10. Lakin, M.R., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of DNA strand displacement devices using probabilistic model checking. J. Roy. Soc. Interface **9**, 1470–1485 (2012)
11. Lakin, M.R., Stefanovic, D., Phillips, A.: Modular verification of chemical reaction network encodings via serializability analysis. Theoret. Comput. Sci. **632**, 21–42 (2016)
12. Lakin, M.R., Youssef, S., Cardelli, L., Phillips, A.: Abstractions for DNA circuit design. J. Roy. Soc. Interface **9**(68), 470–486 (2012)

13. Qian, L., Soloveichik, D., Winfree, E.: Efficient Turing-universal computation with DNA polymers. In: Sakakibara, Y., Mi, Y. (eds.) DNA 2010. LNCS, vol. 6518, pp. 123–140. Springer, Heidelberg (2011). doi:10.1007/978-3-642-18305-8_12
14. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science **332**(6034), 1196–1201 (2011)
15. Schaeffer, J.M., Thachuk, C., Winfree, E.: Stochastic simulation of the kinetics of multiple interacting nucleic acid strands. In: Phillips, A., Yin, P. (eds.) DNA 2015. LNCS, vol. 9211, pp. 194–211. Springer, Cham (2015). doi:10.1007/978-3-319-21999-8_13
16. Shin, S.W.: Compiling and verifying DNA-based chemical reaction network implementations. Master's thesis, Caltech (2011)
17. Soloveichik, D., Cook, M., Winfree, E., Bruck, J.: Computation with finite stochastic chemical reaction networks. Nat. Comput. **7**(4), 615–633 (2008)
18. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. **107**(12), 5393–5398 (2010)
19. Srinivas, N.: Programming chemical kinetics: engineering dynamic reaction networks with DNA strand displacement. Ph.D. thesis, Caltech (2015)
20. Srinivas, N., Parkin, J., Seelig, G., Winfree, E., Soloveichik, D.: Enzyme-free nucleic acid dynamical systems. bioRxiv (2017). http://biorxiv.org/content/early/2017/05/16/138420
21. Thubagere, A.J., Thachuk, C., Berleant, J., Johnson, R.F., Ardelean, D.A., Cherry, K.M., Qian, L.: Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. Nat. Commun. **8**, 14373 (2017)
22. Zhang, D.Y., Seelig, G.: Dynamic DNA nanotechnology using strand-displacement reactions. Nat. Chem. **3**(2), 103–113 (2011)