ACS SyntheticBiology

pubs.acs.org/synthbio Review

Domain-Specific Programming Languages for Computational Nucleic Acid Systems

Matthew R. Lakin* and Andrew Phillips

Cite This: *ACS Synth. Biol.* 2020, 9, 1499–1513 Read Online

ACCESS | Metrics & More | Article Recommendations

ABSTRACT: The construction of models of system behavior is of great importance throughout science and engineering. In bioengineering and bionanotechnology, these often take the form of dynamic models that specify the evolution of different species over time. To ensure that scientific observations and conclusions are consistent and that systems can be reliably engineered on the basis of model predictions, it is important that models of biomolecular systems can be constructed in a reliable, principled, and efficient manner. This review focuses on efforts to address this need by using domain-specific programming languages as the basis for custom design tools for researchers working on computational nucleic acid devices, where a domain-specific language is simply a programming language tailored to a particular application domain. The underlying thesis of our review is that there is a continuum of practical implementation strategies for computational nucleic acid systems, which can all benefit from appropriate domain-specific languages and software design tools. We emphasize the need for specialized yet flexible tools that can be realized using domain-specific languages that compile to more general-purpose representations.

KEYWORDS: *domain-specific languages, biomolecular modeling, nucleic acid computing, RNA synthetic biology, simulation, design*

In recent decades, many of the key processes of life have been found to involve the storage and processing of information, mediated by the sequence-specific chemistry of biopolymers such as DNA and RNA. Given this significant computational component, it is perhaps no surprise that the intersection of computer science and the biological sciences has become a fertile area of research and that computer science has contributed a broad range of tools to the biological and biomedical sciences. This review focuses on one area in which the methods of computer science find application in bioengineering: constructing and analyzing computational nucleic acid systems.

Models, in particular kinetic models, are of central importance throughout science and engineering. They provide a mathematical framework for expressing predictions about the behavior of a system, for testing those predictions against experimental data, and for building forward-engineered systems with specific behaviors. Given the ever increasing complexity of the computational nucleic acid systems that can now be built in the laboratory, it is vital that models are constructed and used appropriately and that they faithfully embody our assumptions about the system in question. Therefore, scientists and engineers must be able to create, test, and distribute models of biological systems that are robust and demonstrably correct. In this review, we argue that domain-specific languages, a well-known tool from computer science, offer an ideal solution for researchers that enables them to create models rapidly, with a high degree of assurance that they are accurate, and in a manner that makes them amenable to scrutiny by other members of the research community.

A domain-specific language (DSL) is a programming language that was designed specifically for use in a particular application area, or domain.[1] General-purpose programming languages, such as Java or Python, are intended for use by a broad range of programmers to express a wide range of different kinds of programs that will span many different application domains. DSLs, on the other hand, have a cut-down set of features that focus on the needs of a particular application domain. The benefit of a DSL is that, in its targeted application domain, the specialized features of the language allow programs to be written extremely concisely. This allows the core ideas to be expressed in

Received: January 28, 2020
Published: June 26, 2020

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499–1513

Table 1. A Continuum of Implementation Techniques for Computational Nucleic Acid Devices[a]

	DNA strand displacement circuits	*in vitro* enzyme-driven DNA circuits	RNA synthetic biology
mode of operation	DNA hybridization	DNA hybridization, enzyme catalysis	RNA interactions, enzyme catalysis
in vitro operation	+	+	+
in vivo operation	+	−	+
enzymatic components	−	+	+

[a]Moving from left to right, implementation techniques move further from pure DNA nanotechnology and toward the complexity of regulatory networks in living organisms. Relevant categories to differentiate between techniques include *in vitro* or *in vivo* operation, any required protein enzymes, and the kinds of reactions that actually implement the computation.

a high-level manner, making them easier to understand and less prone to errors.

In the context of biomolecular device design, DSLs can be used to specify a model at a high level of abstraction, e.g., by specifying that two nucleic acid components bind via certain interaction domains. One can then compile that high-level model description into a lower-level one, e.g., an ordinary differential equation (ODE) model of the kinetics associated with a particular high-level structural design. This is analogous to the way that a traditional compiler compiles a program written in a high-level language, such as Java, into an executable program in a lower-level language, such as machine code. The specialized nature of the DSL means that models can be specified more concisely and may also allow for models to be constructed in a modular fashion from reusable building blocks.

The key advantage of using the DSL approach in this context is that the compilation process that transforms the high-level model into the lower-level one is formally defined by a set of rules, known as the semantics of the language, that fully specify the results of applying this transformation to an arbitrary input model. Thus, the formalized semantics of the DSL encapsulates model assumptions and makes them explicit in a clear and mathematically precise way. This helps to reduce errors in the model construction process and can provide additional sanity checks on the correctness of the model. The DSL, with its precisely defined semantics, can also serve as a common interchange format between researchers working in a particular area. Finally, once a model has been formally defined, it can be converted into alternative formats for other purposes, such as computer-aided verification of certain model properties. This review will highlight these points in the context of the existing literature on the development and use of DSLs and associated software tools for programming nucleic acid systems.

The specific thesis of this review is that DSLs can aid researchers in navigating the continuum of experimental implementation techniques for computational nucleic acid devices that has emerged in the past several decades. These range from *in vitro* techniques powered solely by DNA interactions to synthetic gene circuits implemented using RNA regulators in cells or cell extracts and are summarized in Table 1. We focus on these fields because they all use nucleic interactions to compute and, yet, have seen varying levels of penetration of DSL-based design tools. By exploiting approaches from computer science of compiling high-level languages into lower-level ones, we show that these different approaches may be modeled individually using DSL techniques but can find a unified implementation by compiling them into lower-level modeling languages. Figure 1 lists the languages discussed in this review, organized by the style of the language, the abstraction level(s) at which each operates, and the target application domain(s). These domains include DNA strand displacement, *in vitro* enzyme-driven DNA circuits, and general-purpose

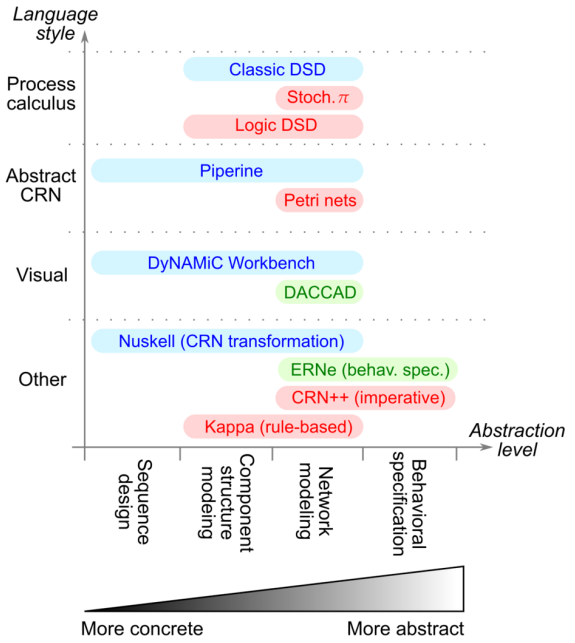


Figure 1. Summary of domain specific programming languages for computational nucleic acid devices mentioned in this review. Blue means the system is primarily applicable to toehold-mediated strand displacement; green means the system is primarily applicable to *in vitro* enzyme-driven DNA circuits, and red means a general-purpose modeling tool. On the *x*-axis, languages are organized by the level(s) of abstraction in the design process that they target, arranged from more concrete on the left to more abstract on the right. On the *y*-axis, languages are arranged by type, including process calculi (languages developed in the study of concurrent programming), abstract CRNs, visual programming languages, and systems that accept other types of input. Abbreviations: stoch π. = stochastic π-calculus; behav. spec. = behavioral specification.

modeling languages. We also discuss the potential for future application of DSLs in RNA synthetic biology.

LANGUAGES AND TOOLS FOR DNA STRAND DISPLACEMENT

Broadly speaking, the field of DNA nanotechnology uses DNA as an engineering material to construct nanoscale structures. The sequence-specific nature of nucleic acid chemistry makes nucleic acids ideal engineering materials because their nucleotide sequences determine their structures and functions. Thus, modifying nucleotide sequences allows molecular interactions to be rationally designed. In this section, we survey previous work on languages and design tools for dynamic DNA nanotechnology, focusing on one of the most successful mechanisms for implementing dynamic DNA computational

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

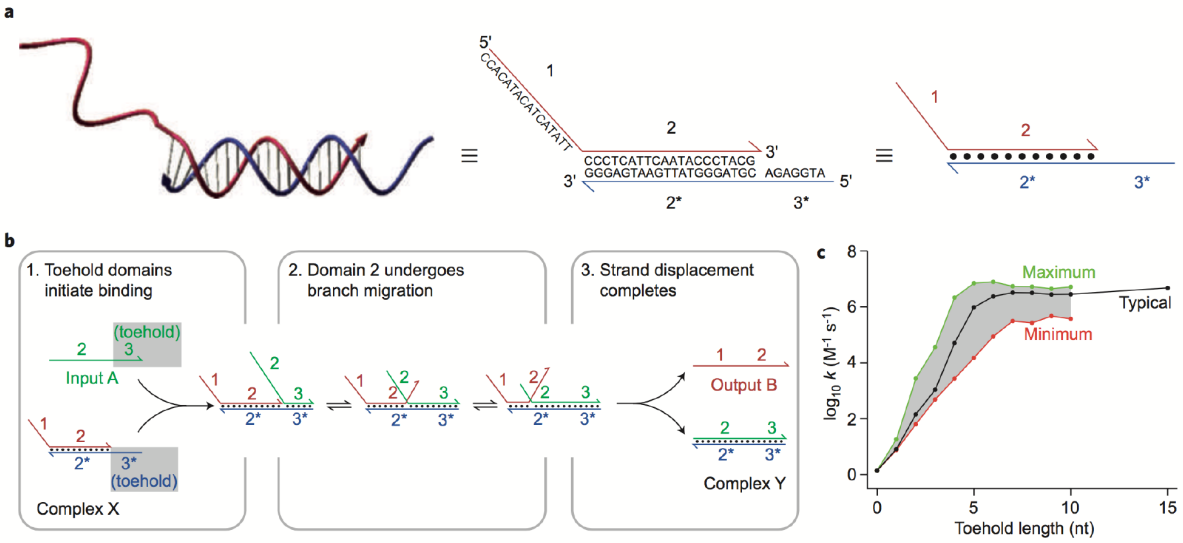


Figure 2. Abstractions for DSL-based modeling of toehold-mediated strand displacement. (a) Abstractions of DNA molecule structures (left) into secondary structures (center) with domain-based sequence motifs (right). Such abstractions simplify high-level modeling. (b) Three-step domain-based model of strand displacement reactions. (c) Strand displacement kinetics as a function of toehold length. Reprinted with permission from ref 3, Copyright 2011, Springer.

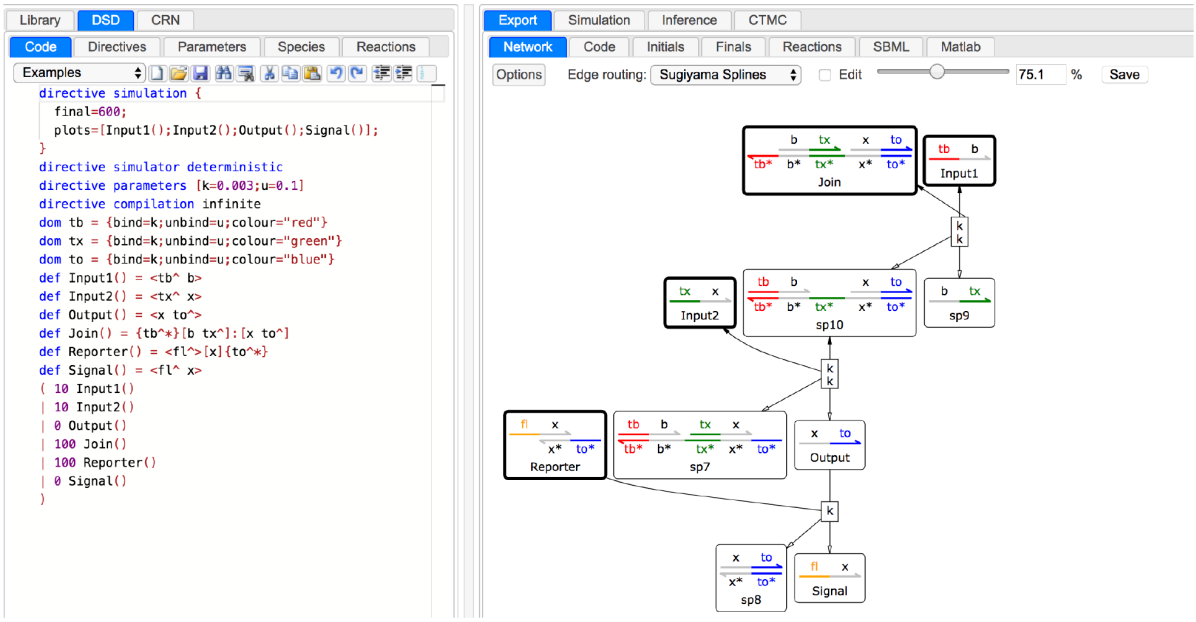


Figure 3. Visual DSD online tool example.[2] Left-hand side shows the DSD code required to define a two-input AND logic gate using DNA strand displacement. Right-hand side shows a graphical visualization of the resulting CRN. Screenshot from https://classicdsd.azurewebsites.net/.

devices: DNA strand displacement.[3] We note that other approaches to DNA-based molecular computation exist, such as DNAzymes[4] which, while they can be used in conjunction with strand displacement circuits,[5] have not yet been the target of DSL development efforts.

DNA strand displacement is a form of competitive hybridization in which an invader strand displaces an incumbent strand that was initially bound to its Watson−Crick complement, by first binding to a short overhanging single-stranded region known as a toehold,[6] and then initiating a branch migration in which the invader and incumbent strands compete to bind to the complementary strand, as outlined in Figure 2. Strand displacement reactions have been demonstrated within,[7] and on the surface of,[8] cells, suggesting potential future applications in biomedical diagnostics and therapeutics.

It has been shown[9] that any abstract chemical reaction network (CRN) can be implemented using a cascade of DNA strand displacement reactions, and a number of different encoding schemes have been proposed to achieve this.[10−12] Abstract CRNs are known to be computationally universal (Turing-complete) up to an arbitrarily small error bound[13] and can thus specify a rich class of behaviors. They thus offer a high-level programming language for DNA strand displacement systems, as outlined below.

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

A

```
directive rules {
bind(P1,P2,Q,D!i) :-
  P1 = C1 [D], P2 = C2 [D'], compl(D, D'),
  Q = C1 [D!i] | C2 [D'!i], freshBond(D!i, P1|P2).

displace(P,Q,E!j,D!i) :-
  P = C [E!j D] [D!i] [D'!i E'!j],
  Q = C [E!j D!i] [D] [D'!i E'!j].

displaceL(P,Q,E!j,D!i) :-
  P = C [D!i] [D E!j] [E'!j D'!i],
  Q = C [D] [D!i E!j] [E'!j D'!i].

unbind(P,Q,D!i) :-
  P = C [D!i] [D'!i], toehold(D),
  Q = C [D] [D'], not adjacent(D!i,_,P).

adjacent(D!i,E!j,P) :- P = C [D!i E!j] [E'!j D'!i].
adjacent(D!i,E!j,P) :- P = C [E!j D!i] [D'!i E'!j].

reaction([P1;P2], "bind",Q) :- bind(P1,P2,Q,_).
reaction([P],"displace",Q) :- displace(P,Q,_,_).
reaction([P],"displace",Q) :- displaceL(P,Q,_,_).
reaction([P],"unbind",Q) :- unbind(P,Q,_).
}
directive parameters [
  bind = 0.003; displace = 1; unbind = 0.1
]
( 10 [<tb^ b>]
| 10 [<tx^ x>]
| 100 [<to^*!1 x*!2 tx^*!3 b*!4 tb^*>
      | <x!2 to^!1> | <b!4 tx^!3>] )
```

B

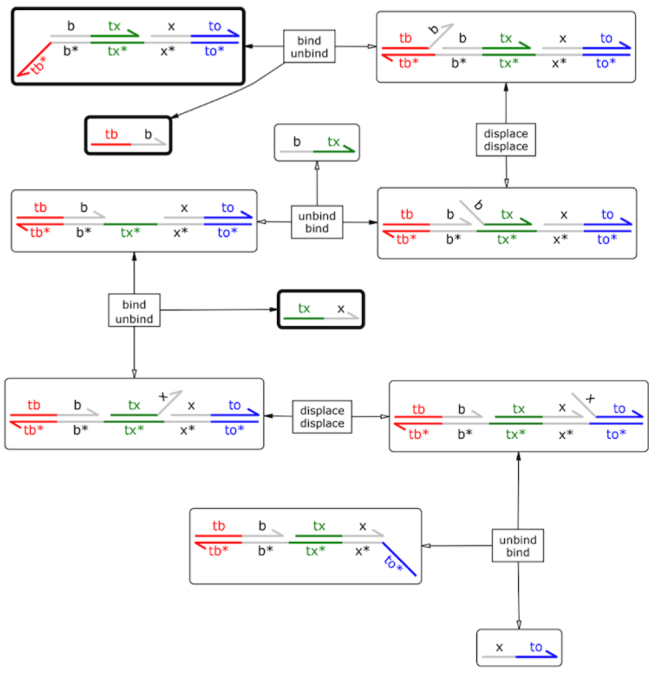


Figure 4. Logic programming for specification and modeling of toehold-mediated strand displacement reactions. (a) Logic program containing predicates that encode binding, unbinding, and displacement reactions as well as the initial conditions of a "Join" gate that functions as an AND logic gate on two input signals. (b) Graphical representation of the resulting chemical reaction containing all species reachable from the initial conditions via reactions specified by the user-supplied predicates. Bold outlines indicate species present in the initial conditions. Reprinted with permission from ref 28, Copyright 2019, American Chemical Society.

Strand displacement is a tempting target for DSLs because the kinds of structures involved significantly constrain the possible interactions. As such, a significant number of DSL-based design tools have been developed. If we assume that nucleotide sequences are assigned such that interaction between non-complementary sequence "domains" is minimal, we can model at the more abstract domain level (Figure 2(a)). In particular, a strand displacement process can be modeled with reasonable accuracy as a two-step process in which the binding of the invader to the complex is nucleated in the toehold region, leading to displacement across the long domain (Figure 2(b)).[14] The derived reversible "toehold exchange" process can be modeled by a similar three-step process of binding, migration, and subsequent unbinding of a secondary toehold that is not displaced by the invader. The domain abstraction for modeling of nucleic acid nanodevices stems from early work by Hagiya and coworkers, where they used the same idea but referred to domains as "abstract bases". That work produced a simulator called VNA (Virtual Nucleic Acid)[15] and applied it to a range of systems, including whiplash PCR[16] and DNA tile assembly.[17]

The first DSL specifically for modeling DNA strand displacement reaction systems was the DSD language, published by Phillips and Cardelli in 2009.[18] An implementation of this system was made available online as the Visual DSD software package.[2] The DSD language is based on a process calculus and consists of a collection of species whose structures are expressed using an ASCII syntax (Figure 3). The set of structures that could be modeled in the early DSD releases was restricted to a limited class of multistranded DNA heteropolymer structures that nevertheless exhibits rich behavior. The core of the DSD compiler is a collection of semantic rules that embodies fundamental assumptions about the domain abstraction and about the dynamic behavior of DNA strand displacement systems. This enables the input structural model to be automatically converted into a kinetic model in the form of a CRN, using rules encoded formally within the compiler itself. This early work was extended with multiple different semantic interpretations that enable a given system to be modeled at different levels of abstraction simply by changing a setting in the DSD compiler, in addition to the first formal modeling of unintended "leak" reactions.[19] Subsequent extensions provided additional simulation and analysis capabilities.[20−23]

The DSD input language itself has been generalized to allow structures expressed as strand graphs to be used as the basis of the reaction enumeration algorithm,[24] building on related work on graph-based formalisms for DNA structures and interactions.[25,26] This approach enables arbitrary structures to be represented, but also includes some that may not be physically plausible for geometric reasons. Thus, there is a need for geometric constraints being included in the semantics of the language, and work is underway in this direction in the context of localized reaction systems.[27] This is a reminder that abstractions must be re-evaluated where appropriate, for example, if the underlying assumptions of the modeler change. More recently, we have refounded the entire DSD language and semantics on a logic programming system (Logic DSD) so that users can define the rules used to compile structural models into kinetic models as part of the program itself,[28] as shown in Figure 4. This enables

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

```
# 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);
```

Figure 5. An example of a translation scheme definition in Nuskell. 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 is translated into a set of fuel complexes, and the sum over all sets is returned by the main function. Reprinted with permission from ref 29, Copyright 2017, Springer.

a range of devices from across the continuum of implementation strategies to be implemented within a single modeling language, as we discuss below.

One of the most complete and integrated systems for DNA strand displacement circuit design is the Nuskell compiler and related toolkit, developed by the Winfree group.[29] This brings together a number of tools to provide an integrated system for specification, verification, and compilation of molecular programs. The desired behavior is specified as an abstract CRN which can then be translated into a DNA strand displacement implementation using a (user-definable) translation scheme. Figure 5 gives example Nuskell code for one such encoding, specifically the "three-domain" encoding.[10,19] Nuskell makes use of the Peppercorn reaction enumerator[30] to compute the sets of possible reactions in the associated dynamic model. Importantly, the correctness of the encodings can be formally verified using pathway decomposition[31] and/or bisimulation.[32] Thus, the Nuskell system provides one of the most integrated systems for the creation of DNA-based molecular programs that are correct by construction. The Peppercorn reaction enumerator has also been used in KinDA,[33] a tool for sequence-level analysis of the thermodynamics and kinetics of DNA strand displacement systems that can automatically detect deviations from the domain-level abstraction. This is an important direction for future research, as understanding the limits of the domain-level abstraction is a pressing concern for designers of nucleic acid circuits.

Another highly integrated toolkit is the DyNAMiC Workbench,[34] which provides a wide range of design tools in a web-based format. The DyNAMiC Workbench uses the port-based, "nodal" design abstraction, as shown in Figure 6, to provide a high-level specification framework for intended circuit behavior. This abstraction was developed by Yin et al. to specify programmed hairpin assembly pathways.[35] The nodal abstraction is used more generally within the DyNAMiC Workbench system, which provides a number of premade nodal species definitions for different experimental frameworks and interfaces to external reaction enumeration and thermodynamic modeling systems. Once a high-level design has been specified using drag-and-drop visual interface, the corresponding components can be specified at the "segment" level (similar to the "domain" abstraction) and designed in terms of specific sequences and base-pairs. Thermodynamic modeling and reaction enumeration are available via integration with external tools.[30,36] This produces a powerful graphical interface for system specification and design.

The Piperine system was developed to compile abstract CRNs into nucleotide sequences suitable for experimental implementation, using the "four-domain" encoding of CRNs into DNA strand displacement reaction networks.[9] In particular, this system was used to design and build an enzyme-free "rock-paper-scissors" chemical oscillator using just DNA.[37] The pipeline of the Piperine system is summarized in Figure 7: the input CRN (specified via a simple textual notation) is fed through various translation passes to produce a specification for the strand displacement signal strands and gate complexes. Nucleotide sequences for the toeholds and signals are then designed using pre-existing sequence design algorithms, to ensure toehold binding energies are balanced and signal strands have minimal secondary structure when free in solution. The

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

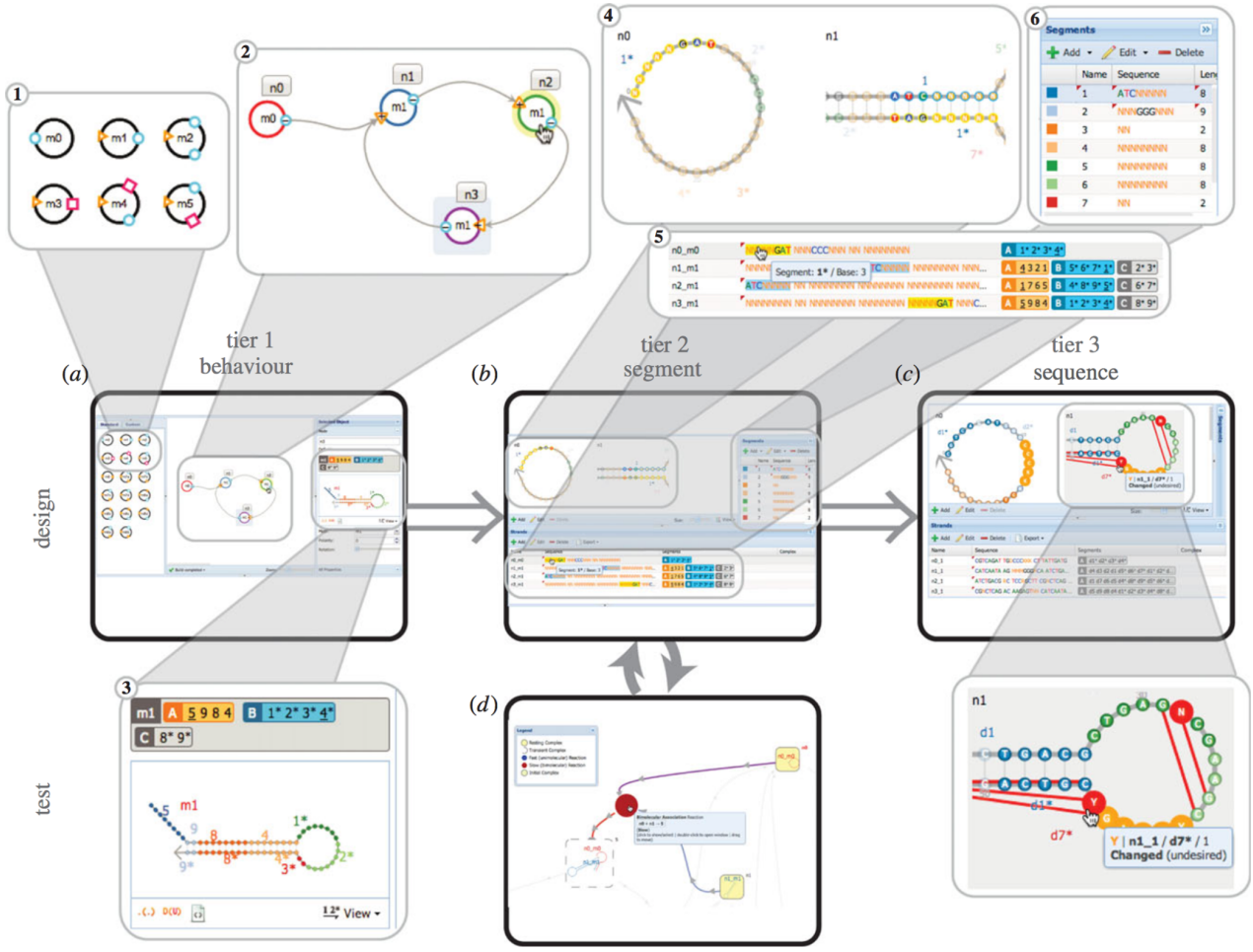


Figure 6. Visual language and interface of the DyNAMiC Workbench system,[34] illustrating the design of hairpin assembly reactions.[35] (a) Nodal design interface. Systems are composed by dragging and dropping nodes from the palette on the left (inset 1) and then connecting nodes in the center panel (inset 2). The right panel shows a preview of the molecular implementation (inset 3). (b) Segment-level design interface. The center panel shows secondary structure view of each complex in the system (inset 4). The lower panel lists sequences and composition of strands in the system (inset 5). The right panel shows the name and sequence of each distinct segment in the system (inset 6). (c) Multisubjective sequence design interface. Similar to the segment-level design interface, different panels show complexes, strands, and segments. The secondary structure view also highlights unintended interactions and shows bases flagged for modification by the analysis. (d) Reaction enumerator interface. Rectangular nodes represent complexes, joined by circular nodes representing reactions between intermediates. Reprinted with permission from ref 34, Copyright 2015, The Royal Society.

output from the tool is a ranked list of candidate sequence designs. Piperine is less flexible than other tools such as Nuskell, as it is specialized to the four-domain translation, but the advantage is that it provides a simple interface to a tightly integrated design system. Other highly specialized compilers have enabled similarly impressive experimental results in DNA strand displacement design. One example is the "seesaw gate" motif,[38] which has been used to implement large-scale digital logic circuits[39] and neural networks.[40,41] The development of a seesaw gate compiler enabled the development of large-scale DNA logic circuits by compiling a high-level logic circuit representation into executable code in a number of output languages for simulations.[39] It also enabled model-guided design of seesaw circuits with unpurified components.[42]

DNA strand displacement systems are typically designed in a modular fashion, which tends to be reflected in the structure of design systems such as Nuskell and DSD. Individual modules can be defined to represent different kinds of strand displacement gate, which can be used to implement even higher-level abstractions that compile to the DSD language. Examples include the seesaw gate motif discussed above, as well as systems designed using techniques from control theory[43,44] and the implementation of high-level CRNs to obtain behaviors such as simple distributed algorithms.[23] Modularity is important because it enables models to be defined in a scalable manner that encourages code reuse. It also reduces the opportunities to introduce errors into models.

Furthermore, once a strand displacement system has been formalized in a design language, computer-aided verification can be applied to check system correctness. In previous work, such analyses have included state-space analysis via probabilistic model checking[45] and via satisfiability modulo theories (SMT) solving,[46] two powerful verification techniques from computer science. There also exist powerful techniques for proving correctness or equivalence of CRN models, including those of TMSD circuits, based on techniques including network morphisms,[47] pathway decomposition,[31] bisimulation,[32] and other approaches that exploit modularity.[48] While the details of

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

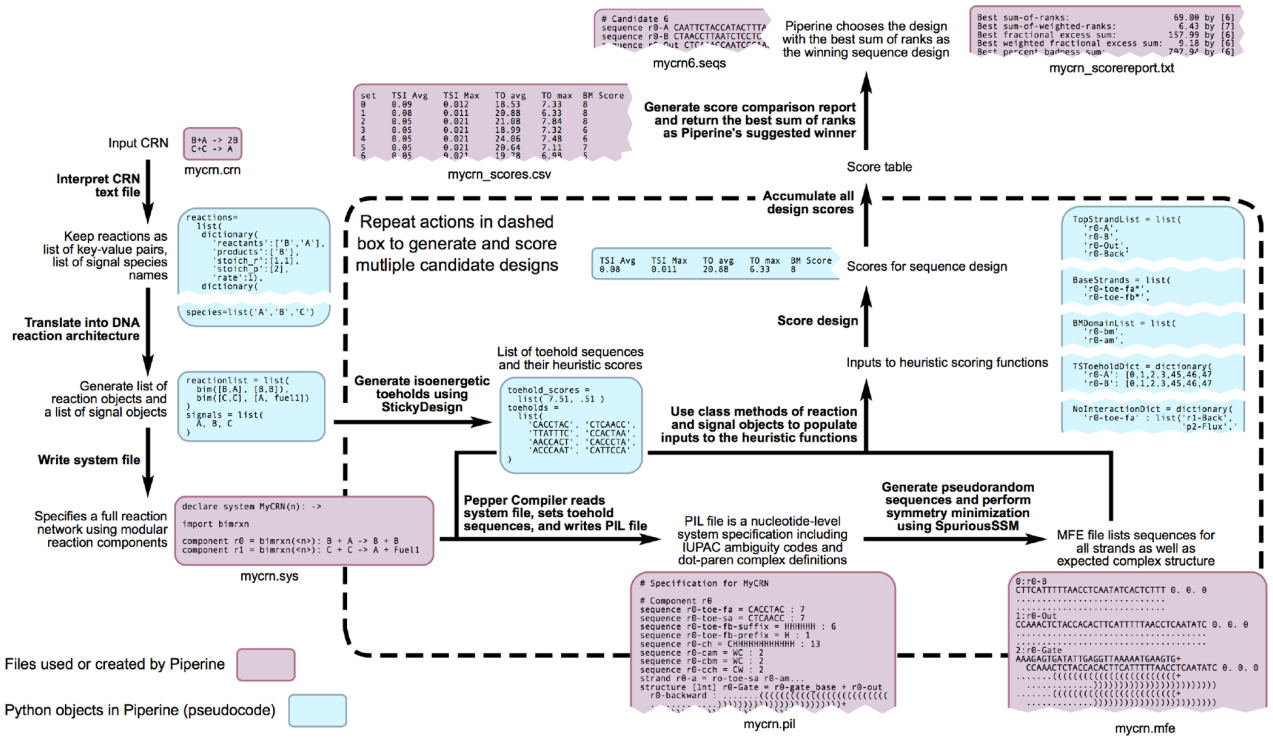


Figure 7. Overview of the Piperine sequence design pipeline. Mauve boxes show example input and output text files for each stage of the compilation pipeline in an example run of the software, and blue boxes illustrate internal data structures. Bold statements describe operations performed on data; standard-case statements explain the contents of that data, and sawtooth breaks in text bubbles indicate that a portion of that data or text is hidden for display purposes. Reprinted with permission from ref 37, Copyright 2017, AAAS.

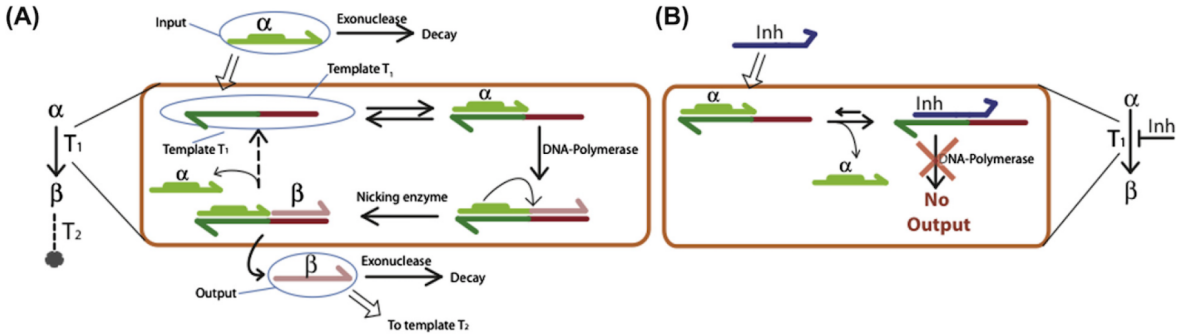


Figure 8. Domain-level abstraction of PEN (polymerase, exonuclease, nickase) toolbox reactions. (a) Activation of node β by node α is implemented via the template strand T_1. Binding of α to T_1 causes extension of α by the polymerase and subsequent nicking, releasing α and β strands into solution. Both α and β are degraded by the exonuclease. (b) Inhibition of the reaction from part (a) by Inh is implemented by Inh displacing α from the template. The Inh strand itself has mismatches at the 3′ end and thus cannot be extended by the polymerase, thereby preventing the activation reactions from occurring. Reprinted with permission from ref 57, Copyright 2014, Elsevier.

these approaches are beyond the scope of this review, we note that creating a model of the system under study in a formal language is a necessary precursor to all of these analyses.

While we have focused in particular on tools specifically for designing DNA strand displacement systems, we note that a range of general-purpose tools exists for prediction and design of nucleic acid nanostructures for DNA strand displacement. A prime example is NUPACK,[36] which actually includes its own DSL for specifying nucleic acid design tasks.[49] Tools for base-level or coarse-grained simulation of nucleic acids, such as Multistrand[50] and oxDNA,[51,52] have already been used to gain insight into strand displacement reaction kinetics.[14] A fruitful future research direction will be to combine these tools with the higher-level modeling approaches outlined above to enable more realistic multiscale modeling and design, as exemplified by the Nuskell[29] and Piperine[37] systems.

LANGUAGES AND TOOLS FOR *IN VITRO* ENZYME-DRIVEN DNA CIRCUITS

Another prominent area of current research in nucleic acid computing is *in vitro* enzyme-driven DNA circuits. These are circuits in which DNA strands provide the template for a circuit that is executed *in vitro* by protein enzymes, including, but not limited to, DNA or RNA polymerases. These enzymes drive circuits via the synthesis and degradation of nucleic acids,

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

drawing on ubiquitous and versatile fuel sources, such as dNTPs, that can be present in relative abundance. Furthermore, the fact that circuit components are also being degraded greatly simplifies the implementation of dynamical systems such as oscillators. These systems require more distinct classes of biomolecular components than DNA strand displacement systems but fewer than the RNA synthetic gene circuits discussed below.

In the genelet approach to *in vitro* enzyme-driven DNA circuits developed in the Winfree lab,[53,54] regulation takes the form of RNA-triggered strand displacement reactions that either activate or deactivate promoter sites on double-stranded DNA templates. The genelet approach relies on RNA synthesis and RNA hybridization interactions, which are less predictable than the corresponding DNA processes. However, the basic principles of execution of these circuits can be well described using the same DNA domain abstractions as are used in DNA strand displacement design, possibly with additional modeling required to capture the enzymatic processes. As such, *in vitro* enzyme-driven DNA circuits have seen a moderate uptake of DSL-based modeling tools. The recently reported primer exchange reaction framework[55] and other similar approaches[56] are also highly amenable to modeling using DSLs. In this review, however, we focus on DSLs applied to another framework: the PEN toolbox developed in the Rondelez lab.[57]

Briefly, the PEN toolbox (also known as "DNA toolbox") approach to *in vitro* enyzme-driven circuits gets its name because it uses three different protein enzymes: a polymerase, an exonuclease and a nickase. The basic abstraction and reactions are shown in Figure 8. Single-stranded DNA templates are activated by the binding of a partially complementary input strand. This triggers extension of that strand and then the nickase nicks the extended strand, producing two shorter strands that can then unbind from the template. The exonuclease degrades (nontemplate) single strands that are free in solution, thereby pulling strand concentrations down. Templates can also be inhibited by the binding of a strand midway along the template strand, which inhibits binding of inputs but does not trigger extension of the inhibiting strand. Experimentally, the PEN toolbox approach has been used to implement robust molecular oscillators,[58] bistable switches,[59] and predator–prey systems.[60] Simulation results have also shown that the PEN toolbox can implement trainable neural networks based on a winner-take-all approach.[61]

Importantly for modeling purposes, the fundamental building blocks of PEN toolbox systems (activation and inhibition of templates that catalytically generate an output, with ongoing degradation of signals) are almost identical to the models of gene regulatory networks studied in systems and synthetic biology. This abstraction provides the basis for the ecosystem of computational design tools that has grown up around the PEN toolbox system. In this case, the design language is the language of regulatory networks expressed as graphs, with nodes representing signals, and with edges representing activation or inhibition of signals by other signals. The Rondelez group has reported computer-assisted tools for the rational design of PEN toolbox circuits, in particular, the DACCAD visual programming tool for the design and simulation of PEN toolbox networks,[62] shown in Figure 9. The DACCAD system enables models of networks to be specified very concisely as the underlying dynamics of the PEN toolbox system is abstracted out. In addition, in one of the few attempts to build behavioral specification into a design tool, the ERNe system uses

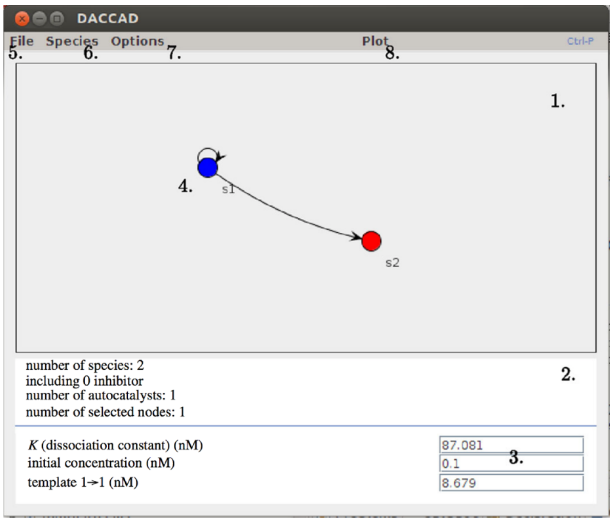


Figure 9. DACCAD visual input interface, including the display panel (1), data panel (2), parameters area (3), graph of the system with selected nodes displayed in blue (4), and menus (5–8). Reprinted with permission from ref 62, Copyright 2014, The Royal Society.

evolutionary algorithms to search for PEN toolbox networks with specified dynamic behaviors.[63] This work highlights the utility of a powerful and well-defined abstraction as the basis of a practical and useful DSL and modeling system.

Extensions to the DSD system for strand displacement modeling have also been used to model PEN toolbox circuits. Initially this work was based on *ad hoc* manual addition of chemical reactions that do not involve strand displacement to the compiled model,[44] which demonstrated some of the difficulty involved in trying to push a DSL beyond the limits of its internalized abstractions. More recent work on logic programming-based modeling in DSD has enabled enzyme-driven circuits such as the PEN toolbox to be modeled directly by programming predicates that formally specify the semantics of those enzyme reactions.[28]

LANGUAGES AND TOOLS FOR RNA SYNTHETIC BIOLOGY

Synthetic biology, which includes the design and construction of synthetic gene regulatory networks,[65] is a field that offers considerable scope for the application of DSLs. Again, this is because the operating mechanisms of synthetic gene circuits enable meaningful and useful abstractions to be made. In synthetic biology, the programmed network structure can be broken down into discrete biological parts (such as promoters, terminators, ribosome binding sites, protein coding regions, and transcription factors) whose interactions with effectors can be abstracted and captured in a formal semantics. Here, however, we focus on RNA synthetic biology,[66] a fast-growing subfield of synthetic biology in which the regulatory function is implemented via RNA interactions. For a treatment of design tools for synthetic biology more broadly construed, we refer the reader to a thorough review by Appleton et al.[67]

Our primary interest in this review is on rational design of RNA regulatory motifs in synthetic biology. Examples include the development of toehold switches, which activate translation via strand displacement reactions initiated by a trans-acting RNA,[64,68] and small transcription-activating RNAs (STARs), which activate transcription via a similar mechanism.[69] Other

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499–1513

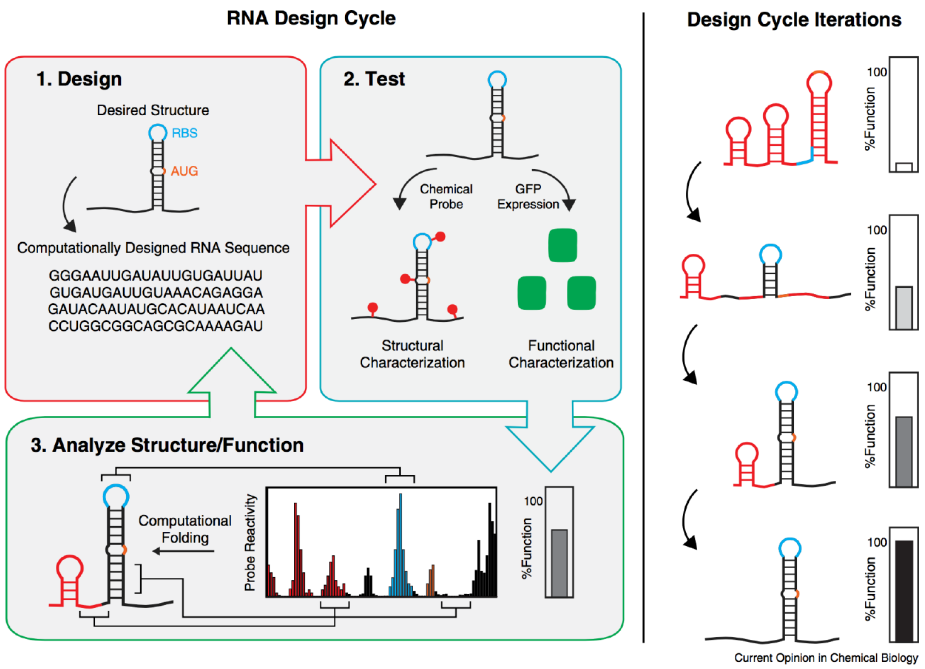


Figure 10. Computational design−test−build cycle in the context of RNA synthetic biology. The relatively predictable nature of RNA folding simplifies the design process and enables rapid design iterations. Reprinted with permission from ref 66, Copyright 2015, Elsevier.

approaches to implementing RNA regulation include repurposing naturally occurring regulatory motifs such as antisense transcriptional attenuators.[70] Some of the earliest work in computer-aided design of RNA-based computing devices, albeit carried out *in vitro*, was Penchovsky's design and implementation of ribozyme-based logic gates.[71]

Early work in RNA engineering used thermodynamic models to predict the conformations, and hence switching behavior, of RNA-based "riboregulator" switches, and to select the best sequences to build multi-input RNA logic gates, including YES, NOT, and AND gates.[72,73] This work was encapsulated into RiboMaker, a web-based tool for riboregulator design.[74] A similar approach had previously enabled the computational design of synthetic ribosome binding sites.[75] Another design tool has been created for designing toehold switches, which uses machine learning to estimate the relative importance of various design criteria.[76] Rational design principles have also been applied to STARs to enhance their fold activation,[77] and NUPACK modeling has been used to design new STAR activation sequences.[78] RNA feedback loops have also been constructed using model-based design.[79] However, these approaches did not use DSLs specifically and, to date, the use of DSLs with formal semantics to construct system models in RNA synthetic biology has been limited.

Nevertheless, RNA synthetic biology remains a promising application domain for biodesign automation techniques developed for other computational nucleic acid devices (Figure 10). Despite the fact that RNA hybridization is more promsicuous than DNA hybridization, and that RNA molecules also undergo additional forms of binding such as kissing loop interactions, the interactions between RNA molecules are still arguably more "designable" than interactions involving protein effectors. However, it must be noted that recent work on *de novo* protein design has produced impressive results.[80] RNA interactions may still be represented, and reasoned about, via a domain-based abstraction, even if RNA hybridization is less specific than DNA hybridization. It is also possible to model the other forms of RNA interaction, such as kissing loop interactions. Principles of modularity are also still of great importance in RNA design and engineering, in part due to the cotranscriptional nature of RNA folding which may often favor the formation of localized structural motifs over distal ones. Computational modeling tools such as CoFold[81] have been developed to address this issue, and RNA structural motifs have been used in the design of modular RNA nanostructures.[82] Furthermore, the engineering of the transcriptional effectors discussed above relied on a modularity principle.[70]

To demonstrate the potential for DSL-based models in RNA synthetic biology, we recently used the Logic DSD system[28] to model toehold switch-based "ribocomputing" circuits,[64] as shown in Figure 11. In that work, the same logic programming rules originally developed to implement toehold-mediated strand displacement reactions in DNA strand displacement circuits were used to encode strand displacement-based control systems in RNA toehold switches. With the inclusion of additional rules to encode extension and cotranscriptional folding of RNAs, similar approaches could be used to model STARs,[69] thereby broadening the applicability of DSL-based modeling in RNA synthetic biology. Yet more semantic rules could be used to accommodate other forms of RNA binding exploited in RNA synthetic biology, such as kissing loop interactions and aptamer-driven riboswitches and ribozymes,[83] as discussed above. The relative promiscuity of RNA hybridization could also be modeled by specifying a nonzero "degree of complementarity" between nominally noninteracting pairs of domains. Therefore, existing principles of domain-based design may be readily extended to RNA structures in RNA synthetic biology, which highlights the potential for further application of DSLs in this field.

Finally, a notable recent development in RNA synthetic biology is the creation of strand-displacement switches for controlling the DNA-binding activity of CRISPR/Cas systems,

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

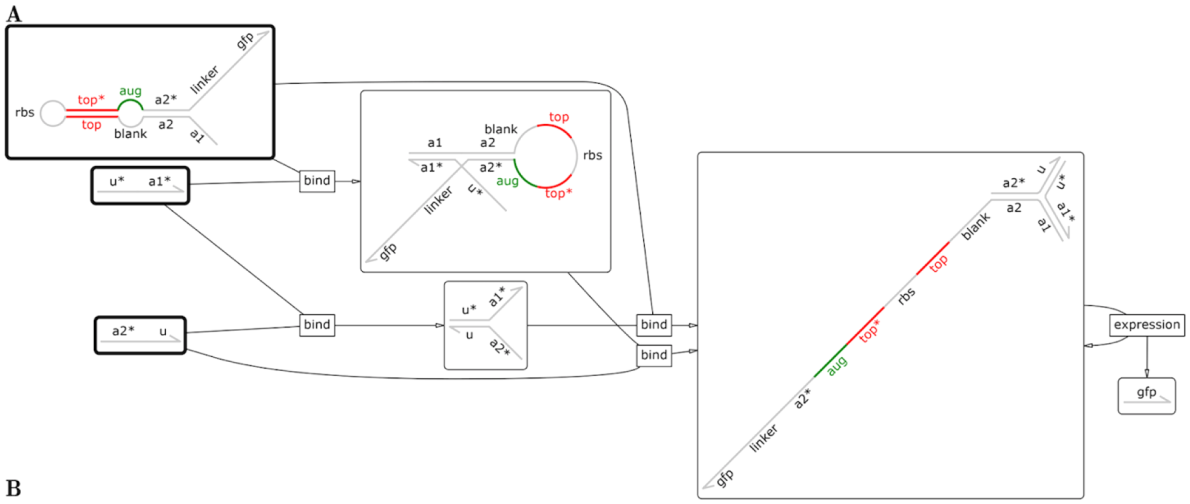


Figure 11. Logic programming for specification and modeling of RNA-based "ribocomputing" reactions, as reported in ref 64. This model uses the same underlying DSL and semantics as in Figure 4. (a) Inferred chemical reaction network and species for the ribocomputing AND logic gate. (b) Logic program fragment encoding reporter expression and initial conditions for the ribocomputing AND logic gate. Reprinted with permission from ref 28, Copyright 2019, American Chemical Society.

based on programmable control of guide RNA structure.[84−87] We anticipate that design tools focused on CRISPR engineering will be a particularly promising area for future application of DSLs in RNA synthetic biology, given the wide potential applicability of CRISPR-based frameworks.

DSLS FOR GENERAL-PURPOSE MODELING

In addition to application-specific DSLs, a range of DSLs exist for general-purpose modeling of (bio)chemical systems. While these languages are not explicitly focused on nucleic acid-based systems, their generality means that they may be used either (i) as low-level compilation targets for more specialized DSLs or (ii) as high-level specification languages to express the desire behavior of an engineered nucleic acid system. Thus, we briefly mention several here.

Petri nets[88] offer a general-purpose modeling framework equivalent to abstract CRNs, and have been used for several decades as a formalism for distributed computing research. For example, the abstract CRN shown below defines a consensus algorithm that converts all individuals of the abstract species X and Y into whichever species was initially present in the majority:[23,89]

$$X + Y \xrightarrow{k} B + B$$

$$B + X \xrightarrow{k} X + X$$

$$B + Y \xrightarrow{k} Y + Y$$

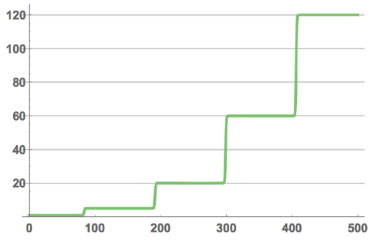
Many DSLs for computational nucleic acid devices accept an abstract CRN as their input language, e.g., the Piperine system.[37] Recent work has developed convenient DSLs for programming the high-level behavior of abstract CRNs themselves, such as the CRN++ language.[90] CRN++ is an imperative programming language, complete with arithmetic operations and control flow constructs such as loops and conditionals. CRN++ programs are compiled directly into an abstract CRN, and an example of a CRN++ program and corresponding simulation output is presented in Figure 12. Another recently produced tool, Kaemika,[91] enables the modular specification of abstract CRNs, with a rigorously defined formal semantics. Kaemika also includes support for programming of microfluidic protocols so that entire reaction protocols can be simulated, and is available as a mobile app for multiple platforms. Other recent approaches to program abstract CRNs have included exhaustive search of limited subsets of the CRN space using SMT solving[92] and targeted search using a user-supplied "sketch", which is simply a CRN containing "gaps" that the solver tries to fill in.[93] As our ability to experimentally realize these devices improves, such programming languages and tools that enable us to map high-level behaviors directly onto implementable biomolecular primitives will become indispensible. Other general-purpose languages include the stochastic π-calculus,[94] a process calculus developed by theoretical computer scientists to model concurrent computation.

While not explicitly focused on nucleic acids, the tools described in this section nevertheless play a critical role in the landscape of DSLs and modeling tools for computational nucleic acid devices. Interestingly, general-purpose languages find use both at the high level, as abstract specification languages,[37] and at the low level, as general-purpose compilation targets.[20]

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

```
1 crn={
2   conc[f,1],  conc[one,1],  conc[i,f0],
3   step[{
4     cmp[i,one],
5     mul[f,i,fnext],
6     sub[i,one,inext]
7   }],
8   step[{
9     ifGT[{
10      ld[inext,i],
11      ld[fnext,f]
12    }]
13  }]
14 }
```

(a) *CRN++* code.



(b) Simulation results for $f0 = 5$; value of f is shown (green line).

Figure 12. Example CRN++ program and corresponding simulation for calculating the factorial of 5. Reprinted with permission from ref 90, Copyright 2020, Springer.

Table 2. Summary of Pros and Cons of DSLs for Computational Nucleic Acid Devices Mentioned in This Review[a]

DSL	pros	cons
Classic DSD[18]	visualization tools, analysis options, good fit for many TMSD frameworks	no integration with sequence design tools, limited set of structures by default
Logic DSD[28]	highly flexible, can handle arbitrary structures	must implement new rules manually, performance hit due to flexibility
Nuskell[29]	special-purpose language for CRN translations, correctness verification tools	nonpseudoknotted structures only
DyNAMiC Workbench[34]	visual input system, high-level "nodal" abstraction	must interface to external modeling and simulation tools
Piperine[37]	integrated pipeline from CRN to sequences, thermodynamic modeling	only implements "4-domain" CRN translation scheme, no interaction with external tools
DACCAD[62]	visual input language, abstracts details of PEN toolbox networks	no integration with sequence design tools
ERNe[63]	high-level design via evolutionary search, input is a behavioral specification	no integration with lower-level design tools

[a]TMSD = toehold-mediated strand displacement.

Table 3. Summary of Pros and Cons of General-Purpose DSLs Mentioned in This Review

DSL	pros	cons
abstract CRNs/Petri nets[88]	simple graphical syntax, studied extensively, general-purpose	limited scalability and modularity, no structural information about species
stochastic π-calculus[94]	scalability, studied extensively, verification techniques	no structural information about species
Kappa[95]	scalability, strong theoretical underpinnings	specialized for combinatorial protein interactions
CRN++[90]	imperative programming interface, familiar control flow primitives	resulting CRNs may be hard to implement in practice
Kaemika[91]	modular CRN composition, formal semantics, protocol simulation	no links to chemical implementation frameworks

However, modeling nucleic acid reaction systems using such tools is cumbersome because the languages do not exploit the underlying structure of nucleic acid strands from which many of the reaction rules inevitably follow. This highlights a niche for DSLs at medium levels of abstraction that are geared specifically for modeling computational nucleic acid devices such as those reviewed above.

CONCLUSIONS

We have reviewed research on the application of DSLs to the challenge of modeling and designing biological and biomolecular systems. We have covered a broad range of application domains, from toehold-mediated DNA strand displacement circuits and enzyme-driven *in vitro* DNA circuits to RNA-mediated synthetic gene circuits in synthetic biology and general-purpose modeling tools. Pros and cons for the field-specific DSLs discussed in this review are summarized in Table 2; those for the general-purpose languages are summarized in Table 3.

Beyond the areas covered explicitly by this review, we note that related fields of research are also beginning to benefit from the use of DSLs and associated tools for modeling and design. One example is systems biology, where the Kappa language was specifically designed to concisely capture "the combinatorics of the interaction between proteins"[95] in systems biology models. Kappa models can concisely represent and model changes in state (e.g., phosphorylation) at particular sites on a protein via formally specified, user-provided rewrite rules.[96] Another example is the design of DNA nanostructures, where widely used software such as caDNAno[97] has revolutionized the design of DNA origami nanostructures.[98] A third example is microfluidics, which has benefited from language-based approaches to device specification and automated layout design.[99] Thus, there are broad possibilities for applying DSL techniques in a wide range of application areas, typically wherever the task at hand is to design elements from a class of target objects whose structures and/or behaviors can be described formally. As DSLs proliferate into new areas of research, there will be additional opportunities for standardizing DSLs for new subareas, following the example of standardized representation languages in synthetic biology such as the Synthetic Biology Open Language (SBOL).[100] These standardized languages can then be used as model interchange formats as well as compilation targets for higher-level design tools. Formalizing languages as standards is not an easy task, however, as the standard will only be a success with broad support from the research community.

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

A key challenge to drive broader adoption of DSLs by researchers will be to identify and define abstractions that are specific enough to provide practical productivity gains without being so restrictive that they stifle creativity or are rapidly rendered obsolete by experimental advances. While computational tools have enabled recent impressive experimental work, e.g., the Piperine compiler discussed above[37] and compilers built specifically for seesaw gate circuits,[42] a major challenge remains to understand how computational tools can gain broader adoption within the scientific community. In addition, much experimental work seems to rely on ad-hoc use of individual tools, such as NUPACK[36] for sequence design. It would be an interesting sociological study to determine the user needs from DSL-based tools and the attitude to such systems, particularly among experimental researchers in dynamic DNA nanotechnology. One possible explanation for the limited penetration of high-level tools into standard experimental practice is their limited integration with low-level sequence design tools. This is one feature shared by the custom-built tools used by experimentalists outlined above.[37,42] In addition, the rapid pace of advances in experimental nucleic acid computing means that a tool will be of limited use if its underlying abstraction is too restrictive. Recent work on reaction enumerators using an underlying graph-based representation of nucleic acid structures[24,26,30] provides an abstraction that should be flexible enough to handle most future needs. Another possible issue is the fragmentation of the tool landscape, with interoperability between tools a major barrier to integrating the tools that already exist to create flexible yet powerful pipelines. The adoption of a common host or scripting language for a larger number of tools could ease this burden, as could the wider adoption of standardized data interchange formats. Finally, we anticipate that the demand for tool support in DNA computing in particular would pick up if the field were to find industrial applications to match those already found by synthetic biology. Recent use of DNA circuits as a mechanism to orchestrate nanoscale processes, including in responsive materials design[101] and patterning,[102] may shed light on the possible path of future developments that could lead to new avenues of practical application and thus drive demand for specialized design tools and DSLs.

As discussed above, lowering the barrier of entry to use and also create DSLs and interface them with other existing tools is likely to greatly expand the uptake of DSLs for the design of computational nucleic acid devices. As such, an important research goal will be to reduce the amount of programming effort and specialized knowledge required to implement such languages. In the field of programming language semantics, substantial effort has been put into developing tools for rapid specification and prototyping of DSLs. A number of powerful language prototyping tools have been developed that could simplify the practical implementation of DSL language definitions, such as Ott.[103] Other potential approaches include hosting interpreters for DSLs of interest within general-purpose programming languages. For example, the Racket system for language-based programming[104] provides advanced facilities for defining new languages within a functional host language, allowing DSL authors to take advantage of the host language's powerful compiler-writing facilities. This approach can also enable deep integration between the DSL and library routines available in the host language for common operations such as data analysis and plotting. This could provide a solution to current fragmentation of the tooling landscape for dynamic DNA nanotechnology, in particular.

Taking a broader view, where before a modeler might write a program to model a particular instance of an experimental system, the author of a DSL instead defines, via the formal semantics of their language, an entire class of programs that model an entire class of experimental systems. The ability to make this leap depends primarily on whether a suitable formal representation of the behavior of the underlying systems can be made, but when it can, the benefits of working at the language level rather than the individual program level are substantial. In particular, it becomes possible to reason not just about the behavior of a particular system but rather about all systems that fall under the abstraction. It also enables the development of general-purpose model generation and compilation tools for automation of the design process. Thus, the application of DSLs for biomolecular systems can be seen as the logical next step in the computerization of biological modeling. In the future, instead of writing new programs to aid our research, we may increasingly write new languages.

■ ASSOCIATED CONTENT

Special Issue Paper

Invited contribution from the 11th International Workshop on Bio-Design Automation.

■ AUTHOR INFORMATION

Corresponding Author

Matthew R. Lakin − *Department of Computer Science, Department of Chemical & Biological Engineering, and Center for Biomedical Engineering, University of New Mexico, Albuquerque, New Mexico 87131, United States;* orcid.org/0000-0002-8516-4789; Phone: +1 505 2773351; Email: mlakin@cs.unm.edu; Fax: +1 505 2775433

Author

Andrew Phillips − *Microsoft Research, Cambridge CB1 2FB, U.K.;* orcid.org/0000-0001-9725-1073

Complete contact information is available at: https://pubs.acs.org/10.1021/acssynbio.0c00050

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grants 1814906, 1763718, 1525553, and 1518861.

■ REFERENCES

(1) Fowler, M. (2010) *Domain-specific languages*, Addison-Wesley.

(2) Lakin, M. R., Youssef, S., Polo, F., Emmott, S., and Phillips, A. (2011) Visual DSD: a design and analysis tool for DNA strand displacement systems. *Bioinformatics 27*, 3211−3213.

(3) Zhang, D. Y., and Seelig, G. (2011) Dynamic DNA nanotechnology using strand-displacement reactions. *Nat. Chem. 3*, 103−113.

(4) Stojanovic, M. N., Stefanovic, D., and Rudchenko, S. (2014) Exercises in Molecular Computing. *Acc. Chem. Res. 47*, 1845−1852.

(5) Brown, C. W., III, Lakin, M. R., Horwitz, E. K., Fanning, M. L., West, H. E., Stefanovic, D., and Graves, S. W. (2014) Signal propagation in multi-layer DNAzyme cascades using structured chimeric substrates. *Angew. Chem., Int. Ed. 53*, 7183−7187.

(6) Yurke, B., and Mills, A. P., Jr. (2003) Using DNA to Power Nanostructures. *Genet. Prog. Evol. Mach. 4*, 111−122.
(7) Groves, B., Chen, Y.-J., Zurla, C., Pochekailov, S., Kirschman, J. L., Santangelo, P. J., and Seelig, G. (2016) Computing in mammalian cells with nucleic acid strand exchange. *Nat. Nanotechnol. 11*, 287−294.
(8) Rudchenko, M., Taylor, S., Pallavi, P., Dechkovskaia, A., Khan, S., Butler, V. P., Jr., Rudchenko, S., and Stojanovic, M. N. (2013) Autonomous molecular cascades for evaluation of cell surfaces. *Nat. Nanotechnol. 8*, 580−586.
(9) Soloveichik, D., Seelig, G., and Winfree, E. (2010) DNA as a universal substrate for chemical kinetics. *Proc. Natl. Acad. Sci. U. S. A. 107*, 5393−5398.
(10) Cardelli, L. (2011) Strand Algebras for DNA Computing. *Nat. Comput. 10*, 407−428.
(11) Cardelli, L. (2013) Two-domain DNA strand displacement. *Math. Struct. Comput. Sci. 23*, 247−271.
(12) Thachuk, C., Winfree, E., and Soloveichik, D. (2015) Leakless DNA Strand Displacement Systems. *Proceedings of the 21st International Conference on DNA Computing and Molecular Programming 9211*, 133−153.
(13) Cook, M., Soloveichik, D., Winfree, E., and Bruck, J. (2009) In *Algorithmic Bioprocesses*, Condon, A., Harel, D., Kok, J. N., Salomaa, A., and Winfree, E., Eds.; Springer-Verlag, pp 543−584.
(14) Srinivas, N., Ouldridge, T. E., Šulc, P., Schaeffer, J. M., Yurke, B., Louis, A. A., Doye, J. P. K., and Winfree, E. (2013) On the biophysics and kinetics of toehold-mediated DNA strand displacement. *Nucleic Acids Res. 41*, 10641−10658.
(15) Nishikawa, A., Yamamura, M., and Hagiya, M. (2001) DNA computation simulator based on abstract bases. *Soft Comput. 5*, 25−38.
(16) Nishikawa, A., and Hagiya, M. (1999) Towards a system for simulating DNA computing with whiplash PCR. *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)*, 960−966.
(17) Rothemund, P. W. K., Papadakis, N., and Winfree, E. (2004) Algorithmic Self-Assembly of DNA Sierpinski Triangles. *PLoS Biol. 2*, e424.
(18) Phillips, A., and Cardelli, L. (2009) A programming language for composable DNA circuits. *J. R. Soc., Interface 6*, S419−S436.
(19) Lakin, M. R., Youssef, S., Cardelli, L., and Phillips, A. (2012) Abstractions for DNA circuit design. *J. R. Soc., Interface 9*, 470−486.
(20) Lakin, M. R., Paulevé, L., and Phillips, A. (2012) Stochastic simulation of multiple process calculi for biology. *Theor. Comput. Sci. 431*, 181−206.
(21) Lakin, M. R., and Phillips, A. (2011) Modelling, simulating and verifying Turing-powerful strand displacement systems. *Proceedings of the 17th International Conference on DNA Computing and Molecular Programming 6937*, 130−144.
(22) Dalchau, N., Chandran, H., Gopalkrishnan, N., Phillips, A., and Reif, J. (2015) Probabilistic Analysis of Localized DNA Hybridization Circuits. *ACS Synth. Biol. 4*, 898−913.
(23) Chen, Y.-J., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., and Seelig, G. (2013) Programmable chemical controllers made from DNA. *Nat. Nanotechnol. 8*, 755−762.
(24) Petersen, R. L., Lakin, M. R., and Phillips, A. (2016) A strand graph semantics for DNA-based computation. *Theor. Comput. Sci. 632*, 43−73.
(25) Kawamata, I., Aubert, N., Hamano, M., and Hagiya, M. (2012) Abstraction of Graph-Based Models of Bio-molecular Reaction Systems for Efficient Simulation. *Computational Methods in Systems Biology: CMSB 7605*, 187−206.
(26) Mokhtar, R., Garg, S., Chandran, H., Bui, H., Song, T., and Reif, J. (2017) In *Advances in Unconventional Computing Vol. 2: Prototypes, Models and Algorithms*; Adamatzky, A., Ed.; Springer International Publishing, Vol. 23; Chapter 15, pp 347−395.
(27) Lakin, M. R., and Phillips, A. (2018) Automated analysis of tethered DNA nanostructures using constraint solving. *Nat. Comput. 17*, 709−722.
(28) Spaccasassi, C., Lakin, M. R., and Phillips, A. (2019) A logic programming language for computational nucleic acid devices. *ACS Synth. Biol. 8*, 1530−1547.
(29) Badelt, S., Shin, S. W., Johnson, R. F., Dong, Q., Thachuk, C., and Winfree, E. (2017) A General-Purpose CRN-to-DSD Compiler with Formal Verification, Optimization, and Simulation Capabilities. *Proceedings of the 23rd International Conference on DNA Computing and Molecular Programming 10467*, 232−248.
(30) Badelt, S., Grun, C., Sarma, K. V., Wolfe, B., Shin, S. W., and Winfree, E. (2020) A domain-level DNA strand displacement reaction enumerator allowing arbitrary non-pseudoknotted secondary structures. *J. R. Soc., Interface 17*, 20190866.
(31) Shin, S. W., Thachuk, C., and Winfree, E. (2019) Verifying Chemical Reaction Network Implementations: A Pathway Decomposition Approach. *Theor. Comput. Sci. 765*, 67−96.
(32) Johnson, R. F., Dong, Q., and Winfree, E. (2019) Verifying Chemical Reaction Network Implementations: A Bisimulation Approach. *Theor. Comput. Sci. 765*, 3−46.
(33) Berleant, J., Berlind, C., Badelt, S., Dannenberg, F., Schaeffer, J., and Winfree, E. (2018) Automated sequence-level analysis of kinetics and thermodynamics for domain-level DNA strand-displacement systems. *J. R. Soc., Interface 15*, 20180107.
(34) Grun, C., Werfel, J., Zhang, D. Y., and Yin, P. (2015) DyNAMiC Workbench: an integrated development environment for dynamic DNA nanotechnology. *J. R. Soc., Interface 12*, 20150580.
(35) Yin, P., Choi, H. M. T., Calvert, C. R., and Pierce, N. A. (2008) Programming biomolecular self-assembly pathways. *Nature 451*, 318−322.
(36) Zadeh, J. N., Steenberg, C. D., Bois, J. S., Wolfe, B. R., Pierce, M. B., Khan, A. R., Dirks, R. M., and Pierce, N. A. (2011) NUPACK: analysis and design of nucleic acid systems. *J. Comput. Chem. 32*, 170−173.
(37) Srinivas, N., Parkin, J., Seelig, G., Winfree, E., and Soloveichik, D. (2017) Enzyme-free nucleic acid dynamical systems. *Science 358*, eaal2052.
(38) Qian, L., and Winfree, E. (2011) A simple DNA gate motif for synthesizing large-scale circuits. *J. R. Soc., Interface 8*, 1281−1297.
(39) Qian, L., and Winfree, E. (2011) Scaling up digital circuit computation with DNA strand displacement cascades. *Science 332*, 1196−1201.
(40) Qian, L., Winfree, E., and Bruck, J. (2011) Neural network computation with DNA strand displacement cascades. *Nature 475*, 368−372.
(41) Cherry, K. M., and Qian, L. (2018) Scaling up molecular pattern recognition with DNA-based winner-take-all neural networks. *Nature 559*, 370−376.
(42) Thubagere, A. J., Thachuk, C., Berleant, J., Johnson, R. F., Ardelean, D. A., Cherry, K. M., and Qian, L. (2017) Compiler-aided systematic construction of large-scale DNA strand displacement circuits using unpurified components. *Nat. Commun. 8*, 14373.
(43) Oishi, K., and Klavins, E. (2011) Biomolecular implementation of linear I/O systems. *IET Syst. Biol. 5*, 252−260.
(44) Yordanov, B., Kim, J., Petersen, R. L., Shudy, A., Kulkarni, V. V., and Phillips, A. (2014) Computational Design of Nucleic Acid Feedback Control Circuits. *ACS Synth. Biol. 3*, 600−616.
(45) Lakin, M. R., Parker, D., Cardelli, L., Kwiatkowska, M., and Phillips, A. (2012) Design and analysis of DNA strand displacement devices using probabilistic model checking. *J. R. Soc., Interface 9*, 1470−1485.
(46) Yordanov, B., Wintersteiger, C. M., Hamadi, Y., Phillips, A., and Kugler, H. (2015) Functional Analysis of Large-Scale DNA Strand Displacement Circuits. *Proceedings of the 21st International Conference on DNA Computing and Molecular Programming 8141*, 189−203.
(47) Cardelli, L. (2014) Morphisms of reaction networks that couple structure to function. *BMC Syst. Biol. 8*, 84.
(48) Lakin, M. R., Stefanovic, D., and Phillips, A. (2016) Modular verification of chemical reaction network encodings via serializability analysis. *Theor. Comput. Sci. 632*, 21−42.
(49) Wolfe, B. R., Porubsky, N. J., Zadeh, J. N., Dirks, R. M., and Pierce, N. A. (2017) Constrained Multistate Sequence Design for Nucleic Acid Reaction Pathway Engineering. *J. Am. Chem. Soc. 139*, 3134−3144.

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

(50) Schaeffer, J. M., Thachuk, C., and Winfree, E. (2015) Stochastic Simulation of the Kinetics of Multiple Interacting Nucleic Acid Strands. *Proceedings of the 21st International Conference on DNA Computing and Molecular Programming 9211*, 194−211.
(51) Doye, J. P. K., Ouldridge, T. E., Louis, A. A., Romano, F., Šulc, P., Matek, C., Snodin, B. E. K., Rovigatti, L., Schreck, J. S., Harrison, R. M., and Smith, W. P. J. (2013) Coarse-graining DNA for simulations of DNA nanotechnology. *Phys. Chem. Chem. Phys. 15*, 20395−20414.
(52) Šulc, P., Romano, F., Ouldridge, T. E., Rovigatti, L., Doye, J. P. K., and Louis, A. A. (2012) Sequence-dependent thermodynamics of a coarse-grained DNA model. *J. Chem. Phys. 137*, 135101−135114.
(53) Kim, J., White, K. S., and Winfree, E. (2006) Construction of an in vitro bistable circuit from synthetic transcriptional switches. *Mol. Syst. Biol. 2*, 68.
(54) Kim, J., and Winfree, E. (2011) Synthetic *in vitro* transcriptional oscillators. *Mol. Syst. Biol. 7*, 465.
(55) Kishi, J. Y., Schaus, T. E., Gopalkrishnan, N., Xuan, F., and Yin, P. (2018) Programmable autonomous synthesis of single-stranded DNA. *Nat. Chem. 10*, 155−164.
(56) Song, T., Eshra, A., Shah, S., Bui, H., Fu, D., Yang, M., Mokhtar, R., and Reif, J. (2019) Fast and compact DNA logic circuits based on single-stranded gates using strand-displacing polymerase. *Nat. Nanotechnol. 14*, 1075−1081.
(57) Baccouche, A., Montagne, K., Padirac, A., Fujii, T., and Rondelez, Y. (2014) Dynamic DNA-toolbox reaction circuits: A walkthrough. *Methods 67*, 234−249.
(58) Hasatani, K., Leocmach, M., Genot, A. J., Estévez-Torres, A., Fujii, T., and Rondelez, Y. (2013) High-throughput and long-term observation of compartmentalized biochemical oscillators. *Chem. Commun. 49*, 8090−8092.
(59) Padirac, A., Fujii, T., and Rondelez, Y. (2012) Bottom-up construction of in vitro switchable memories. *Proc. Natl. Acad. Sci. U. S. A. 109*, E3212−E3220.
(60) Fujii, T., and Rondelez, Y. (2013) Predator-Prey Molecular Ecosystems. *ACS Nano 7*, 27−34.
(61) Genot, A. J., Fujii, T., and Rondelez, Y. (2013) Scaling down DNA circuits with competitive neural networks. *J. R. Soc., Interface 10*, 20130212.
(62) Aubert, N., Mosca, C., Fujii, T., Hagiya, M., and Rondelez, Y. (2014) Computer-assisted design for scaling up systems based on DNA reaction networks. *J. R. Soc., Interface 11*, 20131167.
(63) Dinh, H. Q., Aubert, N., Noman, N., Fujii, T., Rondelez, Y., and Iba, H. (2015) An Effective Method for Evolving Reaction Networks in Synthetic Biochemical Systems. *IEEE Trans. Evol. Comput. 19*, 374−386.
(64) Green, A. A., Kim, J., Ma, D., Silver, P. A., Collins, J. J., and Yin, P. (2017) Complex cellular logic computation using ribocomputing devices. *Nature 548*, 117−121.
(65) Andrianantoandro, E., Basu, S., Karig, D. K., and Weiss, R. (2006) Synthetic biology: new engineering rules for an emerging discipline. *Mol. Syst. Biol. 2*, 2006.0028.
(66) Chappell, J., Watters, K. E., Takahashi, M. K., and Lucks, J. B. (2015) A renaissance in RNA synthetic biology: new mechanisms, applications and tools for the future. *Curr. Opin. Chem. Biol. 28*, 47−56.
(67) Appleton, E., Madsen, C., Roehner, N., and Densmore, D. (2017) Design Automation in Synthetic Biology. *Cold Spring Harbor Perspect. Biol. 9*, a023978.
(68) Green, A. A., Silver, P. A., Collins, J. J., and Yin, P. (2014) Toehold Switches: De-Novo-Designed Regulators of Gene Expression. *Cell 159*, 925−939.
(69) Chappell, J., Takahashi, M. K., and Lucks, J. B. (2015) Creating small transcription activating RNAs. *Nat. Chem. Biol. 11*, 214−220.
(70) Takahashi, M. K., and Lucks, J. B. (2013) A modular strategy for engineering orthogonal chimeric RNA transcription regulators. *Nucleic Acids Res. 41*, 7577−7588.
(71) Penchovsky, R., and Breaker, R. R. (2005) Computational design and experimental validation of oligonucleotide-sensing allosteric ribozymes. *Nat. Biotechnol. 23*, 1424−1433.
(72) Rodrigo, G., Landrain, T. E., and Jaramillo, A. (2012) De novo automated design of small RNA circuits for engineering synthetic riboregulation in living cells. *Proc. Natl. Acad. Sci. U. S. A. 109*, 15271−15276.
(73) Rodrigo, G., Landrain, T. E., Majer, E., Daros, J.-A., and Jaramillo, A. (2013) Full Design Automation of Multi-State RNA Devices to Program Gene Expression Using Energy-Based Optimization. *PLoS Comput. Biol. 9*, e1003172.
(74) Rodrigo, G., and Jaramillo, A. (2014) RiboMaker: computational design of conformation-based riboregulation. *Bioinformatics 30*, 2508−2510.
(75) Salis, H. M., Mirsky, E. A., and Voigt, C. A. (2009) Automated design of synthetic ribosome binding sites to control protein expression. *Nat. Biotechnol. 27*, 946−950.
(76) To, A. C.-Y., Chu, D. H.-T., Wang, A. R., Li, F. C.-Y., Chiu, A. W.-O., Gao, D. Y., Choi, C. H. J., Kong, S.-K., Chan, T.-F., Chan, K.-M., and Yip, K. Y. (2018) A comprehensive web tool for toehold switch design. *Bioinformatics 34*, 2862−2864.
(77) Meyer, S., Chappell, J., Sankar, S., Chew, R., and Lucks, J. B. (2016) Improving Fold Activation of Small Transcription Activating RNAs (STARs) with Rational RNA Engineering Strategies. *Biotechnol. Bioeng. 113*, 216−225.
(78) Chappell, J., Westbrook, A., Verosloff, M., and Lucks, J. B. (2017) Computational design of small transcription activating RNAs for versatile and dynamic gene regulation. *Nat. Commun. 8*, 1051.
(79) Hu, C. Y., Takahashi, M. K., Zhang, Y., and Lucks, J. B. (2018) Engineering a Functional Small RNA Negative Autoregulation Network with Model-Guided Design. *ACS Synth. Biol. 7*, 1507−1518.
(80) Fallas, J. A., Ueda, G., Sheffler, W., Nguyen, V., McNamara, D. E., Sankaran, B., Pereira, J. H., Parmeggiani, F., Brunette, T. J., Cascio, D., Yeates, T. R., Zwart, P., and Baker, D. (2017) Computational design of self-assembling cyclic protein homo-oligomers. *Nat. Chem. 9*, 353−360.
(81) Proctor, J. R., and Meyer, I. M. (2013) CoFold: an RNA secondary structure prediction method that takes co-transcriptional folding into account. *Nucleic Acids Res. 41*, e102.
(82) Geary, C., Chworos, A., Verzemnieks, E., Voss, N. R., and Jaeger, L. (2017) Composing RNA Nanostructures from a Syntax of RNA Structural Modules. *Nano Lett. 17*, 7095−7101.
(83) Lou, C., Stanton, B., Chen, Y.-J., Munsky, B., and Voigt, C. A. (2012) Ribozyme-based insulator parts buffer synthetic circuits from genetic context. *Nat. Biotechnol. 30*, 1137−1142.
(84) Siu, K.-H., and Chen, W. (2019) Riboregulated toehold-gated gRNA for programmable CRISPR-Cas9 function. *Nat. Chem. Biol. 15*, 217−220.
(85) Hanewich-Hollatz, M. H., Chen, Z., Hochrein, L. M., Huang, J., and Pierce, N. A. (2019) Conditional Guide RNAs: Programmable Conditional Regulation of CRISPR/Cas Function in Bacterial and Mammalian Cells via Dynamic RNA Nanotechnology. *ACS Cent. Sci. 5*, 1241−1249.
(86) Oesinghaus, L., and Simmel, F. C. (2019) Switching the activity of Cas12a using guide RNA strand displacement circuits. *Nat. Commun. 10*, 2092.
(87) Jin, M., de Loubresse, N. G., Kim, Y., Kim, J., and Yin, P. (2019) Programmable CRISPR-Cas Repression, Activation, and Computation with Sequence-Independent Targets and Triggers. *ACS Synth. Biol. 8*, 1583−1589.
(88) Goss, P. J. E., and Peccoud, J. (1998) Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. *Proc. Natl. Acad. Sci. U. S. A. 95*, 6750−6755.
(89) Angluin, D., Aspnes, J., and Eisenstat, D. (2008) A Simple Population Protocol for Fast Robust Approximate Majority. *Distrib. Comput. 21*, 87−102.
(90) Vasić, M., Soloveichik, D., and Khurshid, S. (2020) CRN++: Molecular programming language. *Nat. Comput. 19*, 391−407.
(91) Cardelli, L. (2020) Kaemika app: Integrating protocols and chemical simulation. https://arxiv.org/abs/2005.08097, accessed June 26, 2020.

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513

(92) Murphy, N., Petersen, R., Phillips, A., Yordanov, B., and Dalchau, N. (2018) Synthesizing and tuning stochastic chemical reaction networks with specified behaviours. *J. R. Soc., Interface 15*, 20180283.
(93) Cardelli, L., Češka, M., Fränzle, M., Kwiatkowska, M., Laurenti, L., Paoletti, N., and Whitby, M. (2017) Syntax-Guided Optimal Synthesis for Chemical Reaction Networks. *Proceedings of the 29th International Conference on Computer Aided Verification 10427*, 375−395.
(94) Priami, C., Regev, A., Shapiro, E., and Silverman, W. (2001) Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Inform. Process. Lett. 80*, 25−31.
(95) Danos, V., and Laneve, C. (2004) Formal molecular biology. *Theor. Comput. Sci. 325*, 69−110.
(96) Boutillier, P., Maasha, M., Li, X., Medina-Abarca, H. F., Krivine, J., Feret, J., Cristescu, I., Forbes, A. G., and Fontana, W. (2018) The Kappa platform for rule-based modeling. *Bioinformatics 34*, i583−i592.
(97) Douglas, S. M., Marblestone, A. H., Teerapittayanon, S., Vazquez, A., Church, G. M., and Shih, W. M. (2009) Rapid prototyping of 3D DNA-origami shapes with caDNAno. *Nucleic Acids Res. 37*, 5001−5006.
(98) Rothemund, P. W. K. (2006) Folding DNA to create nanoscale shapes and patterns. *Nature 440*, 297−302.
(99) Huang, H., and Densmore, D. (2014) Fluigi: Microfluidic Device Synthesis for Synthetic Biology. *ACM J. Emerg. Technol. Comput. Syst. 11*, 26.
(100) Roehner, N., et al. (2016) Sharing Structure and Function in Biological Design with SBOL 2.0. *ACS Synth. Biol. 5*, 498−506.
(101) Scalise, D., and Schulman, R. (2019) Controlling Matter at the Molecular Scale with DNA Circuits. *Annu. Rev. Biomed. Eng. 21*, 469−493.
(102) Wang, S. S., and Ellington, A. D. (2019) Pattern Generation with Nucleic Acid Chemical Reaction Networks. *Chem. Rev. 119*, 6370−6383.
(103) Sewell, P., Zappa Nardelli, F., Owens, S., Peskine, G., Ridge, T., Sarkar, S., and Strniša, R. (2010) Ott: Effective tool support for the working semanticist. *J. Funct. Program. 20*, 71−122.
(104) Felleisen, M., Findler, R. B., Flatt, M., Krishnamurthi, S., Barzilay, E., McCarthy, J., and Tobin-Hochstadt, S. (2018) A Programmable Programming Language. *Commun. ACM 61*, 62−71.

https://dx.doi.org/10.1021/acssynbio.0c00050
ACS Synth. Biol. 2020, 9, 1499−1513