A Rule-Based Design Specification Language for Synthetic Biology

ERNST OBERORTNER, SWAPNIL BHATIA, ERIK LINDGREN,
and DOUGLAS DENSMORE, Boston University

Synthetic Biology is an engineering discipline where parts of DNA sequences are composed into novel, complex systems that execute a desired biological function. Functioning and well-behaving biological systems adhere to a certain set of biological "rules". Data exchange standards and Bio-Design Automation (BDA) tools support the organization of part libraries and the exploration of rule-compliant compositions. In this work, we formally define a design specification language, enabling the integration of biological rules into the Synthetic Biology engineering process. The supported rules are divided into five categories: *Counting*, *Pairing*, *Positioning*, *Orientation*, and *Interactions*. We formally define the semantics of each rule, characterize the language's expressive power, and perform a case study in that we iteratively design a genetic Priority Encoder circuit following two alternative paradigms—rule-based and template-driven. Ultimately, we touch a method to approximate the complexity and time to computationally enumerate all rule-compliant designs. Our specification language may or may not be expressive enough to capture all designs that a Synthetic Biologist might want to describe, or the complexity one might find through experiments. However, computational support for the acquisition, specification, management, and application of biological rules is inevitable to understand the functioning of biology.

Categories and Subject Descriptors: J.3 [**Life and Medical Sciences**]: *Biology and genetics*

General Terms: Design, Languages, Algorithms, Verification

Additional Key Words and Phrases: Synthetic biology, rule, design, specification, language, template

ACM Reference Format:
Ernst Oberortner, Swapnil Bhatia, Erik Lindgren, and Douglas Densmore. 2014. A rule-based design specification language for synthetic biology. ACM J. Emerg. Technol. Comput. Syst. 11, 3, Article 25 (December 2014), 19 pages.
DOI: http://dx.doi.org/10.1145/2641571

1. INTRODUCTION

The practice of principled engineering of biological systems with new desired behaviors is called Synthetic Biology [Andrianantoandro et al. 2006]. Engineers use sequences of DNA and RNA nucleotides as primary building blocks for synthetic bioengineering. Therefore, engineers adopt a programming strategy that is different from that used in programming computers, where the effect of each instruction type on the state of the computation is well defined, context-independent, and known. If the organism-wide effect of every nucleotide was completely predictable, then engineers would be able to write synthetic genetic programs with a rich well defined and well-understood instruction set, which would lead to more robust genetic programs. At the current time, however, we do not have such a complete mapping of sequence to function [Lucks et al. 2008]. As a result, synthetic programs whose behavior cannot be completely

E. Oberortner was funded under NSF grant 1147158 and by the Agilent Technologies Applications and Core Technology University Research (ACT-UR) program.
Authors' address: Department of Electrical and Computer Engineering, Boston University, 8 Saint Mary's Street, Boston, MA, 02461; email: {ernstl, swapnilb, dougd}@bu.edu; erikmlindgren@gmail.com.

DOI: http://dx.doi.org/10.1145/2641571

and reliably predicted are executed in an uncertain operating context that often causes interference, disruption, and in many cases, malfunction or termination of the synthetic program [Cardinale and Arkin 2012].

Engineers compose new genetic programs from naturally occurring program fragments that are known to work in the operating context, and are tweaked or tuned to work in a new context. These fragments are known as genetic "parts" analogous to the mechanical and electronic parts used to assemble widgets and devices. Analogous to mechanical and electronic engineering, and in a more compelling way due to the combinatorial complexity inherent in nucleotide sequences, these genetic parts are known to function as expected only in a specific operating context captured by a set of conditions or "rules" [Endy 2005; Purnick and Weiss 2009].

Because of the gap in the knowledge of how DNA describes functionality, rules must be reverse engineered via a large number of characterization experiments and may only become known over a long period of time. It becomes important to represent parts of DNA sequences accompanied with rules of use and composition. Apart from reverse engineering, rules can also complement forward engineering. When composing and reusing parts into complex systems, the rules associated with the parts can be composed into the rules of the new system's design. The combined set of rules capture the degrees of freedom of a part's use in a composition, and are important when a system's design is being composed into an even larger system. Then, the degrees of freedom may either grow or shrink depending on the rules of the composed systems.

When recursively composing parts into complex systems, the design space of possible compositions grows rapidly with the size of the desired design. Exploring a combinatorial design space manually is at best tedious and error-prone and at worst impossible, resulting mostly in nonfunctional, noisy, and unreliable systems and a slow rate of understanding the function of the biological system under design. Bio-Design Automation (BDA) [Densmore and Hassoun 2012] tools are starting to address this problem and data exchange standards, such as the Synthetic Biology Open Language (SBOL) [Galdzicki et al. 2014], enable to share and communicate the design of synthetic biological systems, their composed parts, and performance characteristics. BDA can lead to further advancements by having large design spaces systematically explored by sophisticated algorithms that integrate "rules" [Densmore et al. 2010].

The ability to engineer BDA tools that enable design space explorations depends on the design specification language. Simple specification languages may have tractable algorithms but may be insufficiently expressive to capture biological design rules whereas powerful languages may lack the algorithms needed for design space exploration. The development of a rule-based design specification language is guided by three requirements: the expressivity required by the inherent complexity of biological rules, the expressivity required by the specification needs of the end-user, and the availability of efficient algorithms for the operations of interest.

A specification language must at minimum enable the representation of parts with their attributes as the basic objects of interest, and the relationships among parts based on their attributes, along with the ability to combine specifications using standard logical operators, such as $\wedge$ (and), $\vee$ (or), and $\neg$ (not). While individual genetic designs are composed specifications of fully defined parts, the rules of part and design compositions may be conditional, relative, or incomplete of is restricted, allowed, or ensured compositions. Thus, a rule-based design specification language must allow expressing unknown parts, attribute values, and indices. These requirements can be satisfied by a specification language with part and attribute predicates or functions, logical operators, and the $\forall$ (universal) and $\exists$ (existential) quantifiers.

Contributions

The first contribution of this work is the formal definition of a rule-based language for specifying Synthetic Biology designs. The supported rules are grouped into the following types: *Counting*, *Pairing*, *Positioning*, *Orientation*, and *Interaction*. We formally define the semantic of each rule using First-Order Logic (FOL) and Boolean functions, and characterize the expressive power of the specification language. Efficient algorithms then enable (1) to enumerate all designs that comply with a specification, and (2) to verify if a design is compliant with a specification.

The second contribution is a demonstration of our language's applicability on a case study. We iteratively design a genetic Priority Encoder circuit comprised of five genetically wired transcriptional NOR gates [Tamsir et al. 2011]. To perform the case study, we implemented all constraints and an efficient approach to enumerate all valid designs in *miniEugene*, a member of the Eugene ecosystem [Bilitchenko et al. 2011]. This case study illustrates two different paradigms of utilizing our specification language: *rule-based* and *template-driven*. Implementation details are, however, out of the scope of this article. Both paradigms are then discussed, based on several criteria that we have discovered while performing the case study.

As a third contribution, we touch a methodology to approximate the number of all designs that comply with a specification. The benefits lie in (1) the evaluation and verification of a large number of designs without laborious manual work and (2) the production of designs that might have been overlooked by a human designer acting unaided.

In this article, we use the terminology of Synthetic Biology, such as promoter, ribosome binding site (RBS), coding sequences, or terminators. We assume that the reader of this manuscript is familiar with those concepts. For a more detailed explanation, we refer to Densmore and Hassoun [2012].

2. A FORMAL SPECIFICATION LANGUAGE OF SYNTHETIC BIOLOGY DESIGN CONSTRAINTS

In this section, we formally define a language for the specification of synthetic biology designs. The language allows the assertion of Boolean functions (also called predicates) and variables using FOL.

Let $\mathcal{A} = \{\alpha_1, \ldots, \alpha_k\}$ be a set of Boolean functions $\alpha : \mathbb{N} \rightarrow \{T, F\}$. A finite *design* of length N is a sequence of k-tuples $((\alpha_1(1), \ldots, \alpha_k(1)), \ldots, (\alpha_1(N), \ldots, \alpha_k(N)))$.

For example, consider $\mathcal{A} = \{lacI, P1, Ptrc2, R1, GFPmut3, Promoter, Terminator, Reporter, Repressor, forward, reverse\}$. Then, the predicate $Promoter(1)$ asserts all designs that have a *Promoter* in their first position. For design verification purposes, a given design with a promoter at the first position, will have the $Promoter(1)$ predicate evaluate to true.

Predicates can be negated using the logical $\neg$ (not) operator, as well as combined using the logical operators $\wedge$ (and), and $\vee$ (or). Any well-formed combination of the logical operators, predicates, and variables, we call a *specification*. Resultantly, a *design space* is the set of all designs that satisfy all constraints of a specification.

Our formal specification language focuses on the arrangement, the number of appearances, and the orientation of genetic elements in synthetic biology designs, as well as their relationships, such as regulatory interactions. Therefore, we restrict the set of predicates $\mathcal{A}$ to contain only predicates that allow the assertion of part names, part types, and the orientation of part. We restrict the FOL $\forall$ (universal) and $\exists$ (existential) quantifiers to only being used over the design indices i.

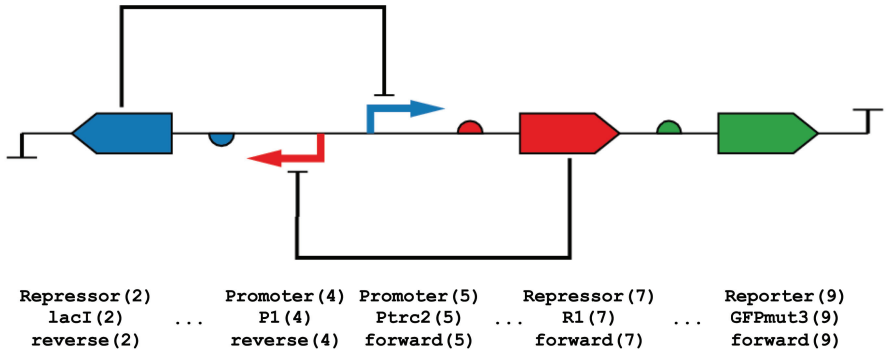


Fig. 1. The genetic Toggle-Switch design.

Table I. Primitive Counting Rule

Rule	Description	Formalization	Examples
α **MORETHAN** n	α must appear more than n times and not more than N times.	$n < \|\alpha\| \leq N$	`Promoter MORETHAN 1` `GFPmut3 MORETHAN 0`

Table II. Composite Counting Rules

Rule	Description	Formalization	Examples
CONTAINS α	α must appear at least once	$0 < \|\alpha\| \leq N$	`CONTAINS Promoter` `CONTAINS lacI`
α **EXACTLY** n	α must appear exactly n times.	$\|\alpha\| == n$ $0 \leq n \leq N$	`Promoter EXACTLY 2` `GFPmut3 EXACTLY 1`

In Figure 1, we graphically exemplify an instance of $\mathcal{A}$ and the predicates using the genetic Toggle-Switch design [Gardner et al. 2000]. In the following, any α and β refer to any $\alpha_i \in \mathcal{A}$, and n ranges over non-negative integers ($n \in \mathbb{N}$). The provided examples refer to the genetic Toggle-Switch design in Figure 1.

2.1. Counting Rules

Counting rules constrain the number of occurrences of genetic elements in a synthetic biology design. A genetic Toggle-Switch, for example, consists of two different repressor genes that control two promoters. Both promoters drive the expression of the downstream repressor genes. Furthermore, there is one reporter gene which is expressed if the Toggle-Switch is in the high state. In Table I, we explain, formalize, and exemplify the primitive *Counting* rule of our specification language. Since our specification language focuses on finite designs of length N, we constrain the MORETHAN rule's operand n to range over the interval $[0, N]$.

In Table II, we explain, formalize, and exemplify composite *Counting* rules. The CONTAINS α rule ensures that α appears at least once in a design, which is equal to the rule α MORETHAN 0. By using the logical operators $\wedge$ and $\neg$ the MORETHAN rule can be composed into complex rules. The α EXACTLY n rule specifies that α occurs exactly n times, which is equivalent to the specification: (α MORETHAN $n-1 \wedge \neg$ (α MORETHAN $n+1$)).

2.2. Pairing Rules

By further using the logical operator $\vee$, the primitive and composite *Counting* rules can be combined to ensure or restrict pairwise occurrences of genetic elements in a design [Chen et al. 2012]. For example, the constraint that the Toggle-Switch design

Table III. Pairing Rules

Rule	Description	Formalization	Examples
α **THEN** β	If α appears, then β must appear.	$(\|\alpha\| \geq 1) \implies (\|\beta\| \geq 1)$	`lacI THEN Ptrc2 P1 THEN R1`
α **WITH** β	α and β must appear.	$(\|\alpha\| \geq 1) \wedge (\|\beta\| \geq 1)$	`lacI WITH Ptrc2 R1 WITH P1`

must contain two different repressor genes can be formulated as a constraint that one repressor gene must be paired with a different repressor gene. Hence, we also introduce so-called *Pairing* rules in our specification language which we explain, formalize, and exemplify in Table III.

The semantic of the THEN rule is formally defined that an occurrence of α implies ($\Rightarrow$) the occurrence of β. The logical term $p \Rightarrow q$ is equivalent to $\neg p \vee q$. Hence, the rule α THEN β is equivalent to (α MORETHAN $1 \Rightarrow \beta$ MORETHAN 1), which is equivalent to ($\neg$ (α MORETHAN 1) $\vee$ β MORETHAN 1). The WITH rule enforces that both α and β must occur and can be formalized as (CONTAINS $\alpha \wedge$ CONTAINS β). The main difference between the THEN and the WITH rule lies in enforcing or restricting the occurrence of β. If using the THEN rule, then β can appear without the occurrence of α.

2.3. Positioning Rules

The organism-wide effects of nucleotides influences and controls the predictability and robustness of genetic programs. Therefore, the precise ordering of parts of DNA sequences in a composition plays an integral role in the function and behavior of biological systems. In genetic circuits regulatory effects depend on the spatial arrangement of parts in a design [Chen et al. 2012]. In the Toggle-Switch design, for example, each repressor gene has an upstream promoters and one immediate upstream RBS. The translation-transcription process of the each promoter and the RBS drives the transcription-translation process of the corresponding repressor genes. The promoters and the RBS are translated BEFORE the translation of the repressor genes. *Positioning* rules enable to constrain the arrangement of genetic parts on a strand of DNA. The *Positioning* rules of our specification language are explained, formalized, and exemplified in Table IV.

A notable similarity of all *Positioning* rules is that none of them constrains the appearance of its operands α and β in a design. The rationale behind was to define a set of primitive rules that can be logically composed into more complex rules. For example, a rule p ALL_BEFORE g does not imply that the promoter p and the gene g appear in a design. The logical composition CONTAINS $p \wedge$ CONTAINS $g \wedge p$ ALL_BEFORE g ensures that p and g appear in the design and that all p's are positioned before the first appearance of g. Also, *Positioning* rules do not incorporate the orientation of its operands α and β.

2.4. Orientation Rules

When composing genetic parts together, the orientation of the parts matters. In the Toggle Switch, for example, the two promoters $P1$ and $Ptrc2$ face different orientations to control the expression of the $lacI$ and $R1$ repressor genes. Furthermore, the promoters, RBSs, genes, and terminators have the same orientation. In Table V, we explain, formalize, and exemplify the *Orientation* rules of our specification language.

2.5. Interaction Rules

Interaction rules specify regulatory interactions in a genetic design. We differentiate between desired and observed interactions. Desired interactions must be statically specified in the design of a synthetic biological systems [Cardinale and Arkin 2012]. A system's dynamic behavior, however, is subject to various unknowns, such as context,

Table IV. Positioning Rules

Rule	Description	Formalization	Examples
STARTSWITH α	If α appears in the design, then α must appear at the first position.	$(\lvert\alpha\rvert \geq 1) \implies \alpha(1)$	`STARTSWITH Terminator`
ENDSWITH α	If α appears in the design, α must appear at the last position.	$(\lvert\alpha\rvert \geq 1) \implies \alpha(N)$	`ENDSWITH Terminator`
α **ALL_AFTER** β	All β components must precede all α components.	$\forall i(\beta(i) \implies \forall j(\alpha(j) \implies (i \leq j)))$	`forward ALL_AFTER reverse`
α **SOME_AFTER** β	At least one β component must precede at least one α component.	$(\exists i\alpha(i)) \implies \exists j(\beta(j) \wedge (j \leq i))$	`RBS SOME_AFTER R1`
α **ALL_BEFORE** β	All α components must precede all β components.	$\forall i(\beta(i) \implies (\forall j(\alpha(j) \implies (j \leq i)))$	`reverse ALL_BEFORE forward`
α **SOME_BEFORE** β	If α appears in the design, then at least one α component must precede at least one β component.	$(\exists i\alpha(i)) \implies (\exists j(\beta(j) \wedge (j \geq i)))$	`R1 SOME_BEFORE RBS`
α **ALL_NEXTTO** β	All β components must appear immediately before or after all α components.	$\forall i(\beta(i) \implies (\forall j(\alpha(j) \implies (j = i-1 \vee j = i+1)))$	`lacI ALL_NEXTTO T1T2`
α **SOME_NEXTTO** β	At least one β component must appear immediately before or after at least one α component.	$(\exists i\alpha(i)) \implies \exists j(\beta(j) \wedge (j = i-1 \vee j = i+1))$	`RBS SOME_NEXTTO R1`

environmental signals, fluctuations, and stochastic effects [Randall et al. 2011]. To learn and understand those unknowns and their influences on the robustness and stability of biological systems, the dynamically observed regulatory interactions must be captured.

In our specification language, *Interaction* rules are considered as characteristics or meta-information of a synthetic biological system's design. Therefore, we introduce the set $\mathcal{P}$ that contains all genetic parts of the design and the set $\mathcal{R}$ that contains either statically specified or dynamically observed interactions. In Table VI, we explain, formalize, and exemplify the three *Interaction* rules and their semantics of our specification language.

3. EXPRESSIVE POWER

Given the *Counting*, *Pairing*, *Positioning*, *Orientation*, and *Interaction* rules, we now characterize the expressive power of our specification language. The expressive power of our specification language is defined as the set of biological design spaces that are describable with the provided rules. We compare the expressive power of the described *Counting*, *Pairing*, *Positioning*, *Orientation*, and *Interaction* rules against

Table V. Orientation Rules

Rule	Description	Formalization	Examples
ALL_FORWARD α	If α appears in the design, then all occurrences of α must be forward oriented.	$\lvert\alpha\rvert \geq 1 \implies$ $\forall i\ \alpha(i) \implies$ $forward(i)$	`ALL_FORWARD GFPMut3` `ALL_FORWARD R1`
ALL_REVERSE α	If α appears in the design, then all occurrences of α must be reverse oriented.	$\lvert\alpha\rvert \geq 1 \implies$ $\forall i\ \alpha(i) \implies$ $reverse(i)$	`ALL_REVERSE P1` `ALL_REVERSE lacI`
SOME_FORWARD α	If α appears in the design, then at least one occurrence of α must be forward oriented.	$\lvert\alpha\rvert \geq 1 \implies$ $\exists i\ \alpha(i) \wedge forward(i)$	`SOME_FORWARD` `Promoter`
SOME_REVERSE α	At least one occurrence of α must be reverse oriented.	$\lvert\alpha\rvert \geq 1 \implies$ $\exists i\ \alpha(i) \wedge reverse(i)$	`SOME_REVERSE` `Promoter`
ALL_FORWARD	All elements in the design must be forward oriented.	$\forall i\ forward(i)$	`ALL_FORWARD`
ALL_REVERSE	All elements in the design must be reverse oriented.	$\forall i\ reverse(i)$	`ALL_REVERSE`

Table VI. Interactions Rules

Rule	Description	Formalization	Examples
g **REPRESSES** p	The gene g and the promoter p have a *represses* regulatory interaction.	$g, p \in \mathcal{P}$, $(g, represses, p) \in \mathcal{R}$	`R1 REPRESSES P1` `lacI REPRESSES` `Ptrc2`
in **INDUCES** p	An inducer in induces/ activates the promoter. p	$in, p \in \mathcal{P}$, $(in, induces, q) \in \mathcal{R}$	`in1 INDUCES P1` `in2 INDUCES` `Ptrc2`
p **DRIVES** g	The gene g has an upstream promoter p, which has the same orientation and there is no intervening terminator that has the same orientation as g and p.	$\forall j \exists i\ p(i) \wedge g(j) \implies$ $Promoter(i) \wedge Gene(j) \wedge$ $(forward(i) \wedge forward(j) \wedge$ $\forall k : ((i-1<k) \wedge (k<j+1)$ $\implies (Terminator(k) \implies$ $reverse(k))))$ $\vee$ $(reverse(i) \wedge reverse(j) \wedge$ $\forall k : ((i-1<k) \wedge (k<j+1)$ $\implies (Terminator(k) \implies$ $forward(k))))$	`Ptrc2 DRIVES R1` `P1 DRIVES lacI`

the expressive power of star-free languages, that is, regular languages without the Kleene Star operator [Hopcroft et al. 2006]. In the following fact and proof, T denotes *true* and F denotes $false$.

We use the following fact as starting point [McNaughton and Papert 1971].

FACT 1. *Let Σ be a nonempty finite set. For every $\sigma \in \Sigma$, define a predicate $\hat{\sigma} : \{1, \ldots, n\} \to \{T, F\}$ such that it is true if and only if σ appears at position i in the string. The set of strings described by first-order logical sentences with predicates $\hat{\sigma}$ as defined here and the less-than relation ($<$) are precisely those described by star-free regular expressions.*

We now show that our specification language is as powerful as FOL and the less-than ($<$) relation, and thus, by Fact 1, no more powerful than star-free regular expressions.

PROOF SKETCH. Since our design and design space definition uses k predicates $\alpha_1, \ldots, \alpha_k$, and our design is defined as a sequence of tuples, we must first show that this can be mapped to the single symbol predicates σ as defined in Fact 1.

Let $\Sigma = \{T, F\}^k$, and for each $w \in \Sigma$, we define the predicate γ_w to be true at position i in the design if and only if w is identical to the tuple at position i in the design. Thus, we have a set of predicates $\Gamma = \{\gamma_w : w \in \{T, F\}^k\}$ in one-to-one correspondence with the symbols that can occur at each position in a design, exactly like the setting in Fact 1.

Next, we must show that every rule in Table IV can be expressed in first-order logical sentences using predicates γ_w and the less-than ($<$) relation. Clearly, the first-order logic formalism is identical in both settings, so we only need to show that the $\leq$ relation and terms such as α and β can be expressed using the predicates γ_w and the less-than ($<$) relation. Every $i \leq j$ relation rules can be expressed as $(\neg(i < j) \wedge \neg(j < i)) \vee i < j$ over $\mathbb{N}$.

Recall that any α, β in Table IV is one of the $\alpha_i \in \mathcal{A}$. The constraint $\alpha_i(j)$ for a quantified variable j is satisfied by any tuple with a T value in its ith component. The set of predicates Γ contains 2^{k-1} predicates, all requiring a T in the ith component of a tuple in the design. Thus, the constraint $\alpha_i(j)$ can be replaced with the equivalent constraint constructed by a disjunction of the predicates in the set $\{\gamma_w : w \in \{T, F\}^{i-1}T\{T, F\}^{k-i}\}$.

The NEXTTO rules can be expressed by denying the possibility of an index between those of the two parts in question—which is expressible in FOL with the less-than ($<$) relation.

Since the EXACTLY and MORETHAN rules (see Tables I and II) allow for a constant n, they are expressible using (respectively) exactly n or at least n variables and asserting the presence of α at these indices, and translating that constraint into a constraint using the γ_w predicates as described previously. The n (respectively, $n+1$ for MORETHAN) indices can be asserted to be all distinct by listing all of their permutations, which is possible to express using the $<$ relation.

The *Orientation* rules defined in Table V are expressible using FOL with the less-than ($<$) relation and the given predicates. The DRIVES rule can clearly be written using FOL and the γ_w predicates as discussed above. The INDUCES and REPRESSES rules in Table VI do not affect the design space—so they can be ignored.

Thus, every design space described by *Counting*, *Pairing*, *Positioning*, *Orientation*, and *Interaction* rules is describable by FOL with the less-than ($<$) relation, and therefore by star-free regular expressions. □

We have shown that the expressive power of our specification language is no greater than that of a star-free language, that is, a regular language without the Kleene star operator. This raises the following questions: What biological rule complexity and design intent does our language not capture? Let's consider the following design pattern, for example: $p_1, r_1, c_1, t_1, p_1, r_1, c_1, t_1$, where p_1 denotes a promoter, r_1 denotes an RBS, c_1 denotes a coding sequence, and t_1 denotes a terminator. The *Positioning* rules of our specification language are not expressive enough to specify only this design pattern. The infinite class of all transcriptionally valid designs, that is, all designs with one or more transcriptional units, each unit containing one or more genes, is also not describable using these rules. To find a way how to deal with such problems, we explore in the next section the applicability and expressivity of our specification language in a case study.

4. CASE STUDY: ITERATIVELY DESIGNING A GENETIC PRIORITY ENCODER CIRCUIT

Now, we demonstrate our specification language in a case study designing an artificial synthetic biological system. The goal is is to manifest that our minimal specification language is expressive enough to design large genetic circuits.

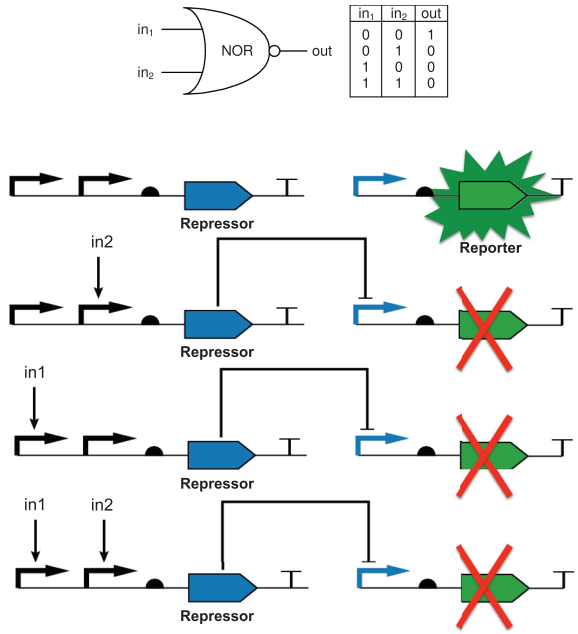


Fig. 2. Logic and genetic NOR gates.

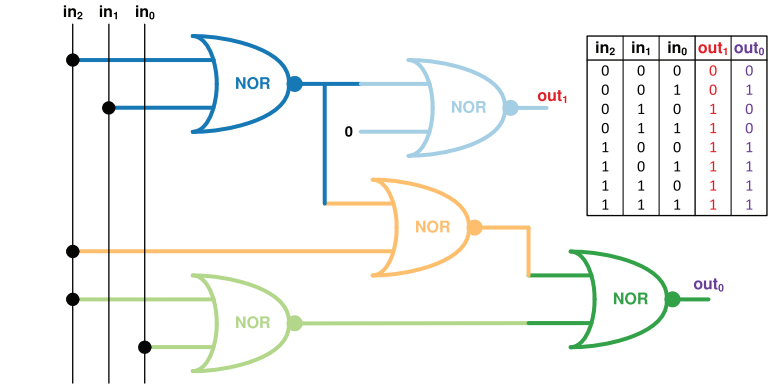
We performed the case study using *miniEugene*, which is part of the Eugene ecosystem of software languages and library especially tailored for rule-based design specifications of synthetic biological systems [Bilitchenko et al. 2011]. miniEugene allows the specification of the rules presented in our specification language. Both, Eugene and miniEugene are open-source (BSD-3 Clause License) and freely accessible via the www.eugenecad.org web site (Creative Commons Attribution 4.0 International License (CC BY 4.0)).

4.1. A Description of the Genetic Priority Encoder Circuit

A transcriptional NOR gate can be designed as a genetic device consisting of several genetic parts (see Figure 2). It has two inducible promoters that initiate—depending on the input signals—the transcription of a reporting gene, such as a green fluorescent protein (GFP). Only if both input signals are not present (i.e., "off") in the NOR gate's environment, then a repressor gene allows a downstream promoter to initiate transcription. If both input signals are present (i.e., "on") or just one of them, then the downstream promoter is repressed and hence, the reporter will not be transcribed.

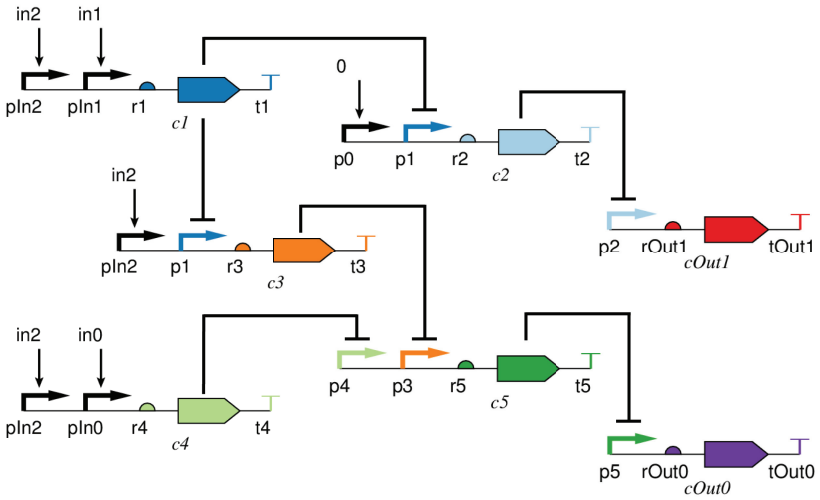
Bugaj and Schaffer [2012] as well as Ruder et al. [2011] highlight recent advances of genetic circuits using synthetic biology in therapeutics, making it possible to implement and perform desired function depending on novel input signals. Shankar and Pillai [2011] discuss synthetic biology as an aid in effective and cheaper drug synthesis. A synthetic biological system could be designed to produce customized drugs for individual patient treatment.

Lets imagine a biological system that responds to three input signals—in_0, in_1, and in_2. These three inputs can, for example, correspond to biomarkers which report increasing progression of a disease of a patient. For example, in_2 is only present in the latest stages of the disease and in_0 in the earliest stage. A genetic Priority Encoder



in_2	in_1	in_0	out_1	out_0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

(a) A NOR-based combinational logic Priority Encoder



(b) Genetic Priority Encoder

Fig. 3. The logic and genetic Circuits of the Priority Encoder.

could produce drugs to treat patients with a high concentration if the third input signal (in_2) is present, medium if in_1 is present, and low if in_0 is present.

In Figure 3(a) we present a traditional combinatorial logic circuit, demonstrating what such a Priority Encoder would look like. Notice that out_1 and out_0 when taken together as a binary value produce the values 3, 2, or 1 depending on the presence of in_2, in_1, in_0, respectively. What is not shown is the additional logic that would process the out_1 and out_0 signals to produce a drug at the desired concentration.

4.2. The Case Study's Part Library

First, we define a library of of artificial genetic parts and their relationships among each other (such as orthogonal repressor-promoter pairs) or to specific input signals (such as signal molecules that induce the transcription-translation process). The case study's part library $\mathcal{P}$ consists of

—four inducible promoters: pIn_0, pIn_1, pIn_2, p_0,
—five repressible promoters: p_1, p_2, p_3, p_4, p_5,

—seven ribosome binding sites: r_1, r_2, r_3, r_4, r_5, $rOut_0$, $rOut_1$,
—five repressor genes: c_1, c_2, c_3, c_4, c_5,
—two reporter genes: $cOut_0$, $cOut_1$,
—seven terminators: t_1, t_2, t_3, t_4, t_5, $tOut_0$, $tOut_1$.

The four input signals in_0, in_1, in_2, and 0 are intended to INDUCE the four inducible promoters pIn_0, pIn_1, pIn_2, and p_0, respectively. The five repressor genes c_1, c_2, c_3, c_4, c_5 are intended the REPRESS the five repressible promoters p_1, p_2, p_3, p_4, p_5, respectively. The set of relations $\mathcal{R}$ in the case study's part library looks as follows:

$$\begin{aligned}\mathcal{R} := \{&(c_1, \mathit{represses}, p_1), (c_2, \mathit{represses}, p_2), (c_3, \mathit{represses}, p_3),\\ &(c_4, \mathit{represses}, p_4), (c_5, \mathit{represses}, p_5), (in_0, \mathit{induces}, pIn_0),\\ &(in_1, \mathit{induces}, pIn_1), (in_2, \mathit{induces}, pIn_2), (0, \mathit{induces}, p_0)\}.\end{aligned}$$

4.3. The Case Study's Iterations

Due to the complexity of the Priority Encoder circuit, we iteratively specify its design and try to reuse the specified rules of every iteration in the subsequent iteration.

—*Iteration* 1. In the first iteration, we focused on designing the basic part compositions, that is, RBS-GENE-TERMINATOR triplets. We categorize the compositions into repressing and reporting cassettes. A repressing cassette contains a repressor gene that has an upstream RBS and one downstream terminator immediately next to it. A reporting cassette has an equivalent structure except that the repressor gene is being replaced with a reporter gene.
—*Iteration* 2. In the second iteration, we focus on extending the RBS-GENE-TERMINATOR triplets with promoters that are receptive to the corresponding input signals. Then, we "wire" the individual cassettes into devices for the output signals out_1 and out_0.
—*Iteration* 3. In the final iteration, we reuse the rules specified in Iterations 1 and 2 and "wire" the out_1 and out_0 devices.

In the following, we demonstrate the execution of those three iterations following two different approaches. In both approaches, we focus on designs with only forward oriented parts using the ALL_FORWARD rule (see Table V).

4.4. Approach I—Rule-Based Approach

In this approach, we iteratively utilize rules of the formal specification language to design a genetic Priority Encoder circuit.

Iteration 1. In the first iteration, we specify the structure of the RBS-GENE-TERMINATOR compositions. miniEugene requires the specification of the length of the composition which here is three (3). Then, we utilized *Counting*, *Positioning*, and *Orientation* rules to specify the compositions, as illustrated in Figure 4. The two RBS-REPRESSOR-TERMINATOR compositions refer to the dark blue and light blue NOR gates of the genetic Priority Encoder circuit (see Figure 3(b)). For example, the r_1-c_1-t_1 triplet consists of the r_1 RBS part, the repressor gene c_1, and a terminator t_1—exactly in this order and all being forward oriented. Similarly, the $rOut_1$-$cOut_1$-$tOut_1$ composition for the Priority Encoder's out_1 signal contains one RBS $rOut_1$, one reporting gene $cOut_1$, and one terminator $tOut_1$.

Iteration 2. The out_1 signal is produced by the light blue NOR gate, which is induced by a constant 0 signal and the repressor gene of the dark blue NOR gate. The two input signals in_2 and in_1 bind to the two inducible promoters of the dark blue NOR gate. We reused the rules specified in Iteration 1 and adapted the length of the out_1 design to 14. We attach promoters using *Counting* and *Positioning* rules and specify *Interaction* rules for the desired regulatory interactions.

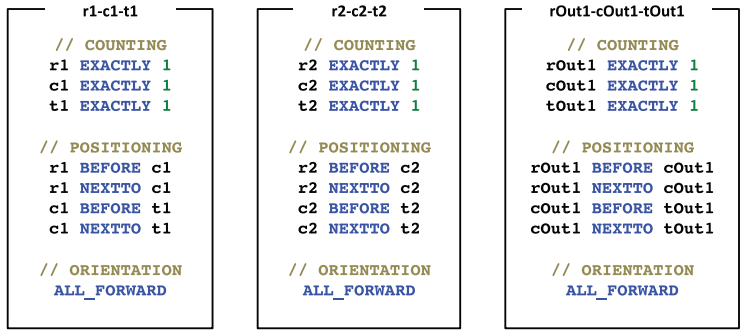


Fig. 4. Rules for specifying RBS-GENE-TERMINATOR Compositions.

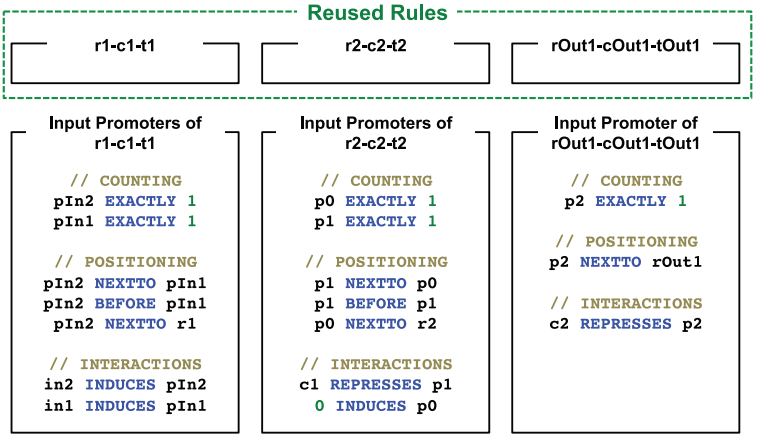


Fig. 5. Rules for specifying the Priority Encoder's *out*1 signal.

We illustrate the newly specified rules in Figure 5. Equivalently, we specify the out_0 signal of the genetic Priority Encoder circuit, whose length is 19.

Iteration 3. In the final iteration, we connected the out_1 and out_0 signals by specifying the desired repressing interaction between the c_1 repressor of the dark blue NOR gate the p_1 input promoter of the orange NOR gate. In miniEugene, we must first specify the length of the Priority Encoder circuit which is 33.

Following the rule-based approach, we discovered one limitation of our language. Now, there are two instances of the p_1 promoter and the *Positioning* rules are not expressive enough in such situations. Therefore, we have to duplicate the p_1 promoter and modify the reused rules accordingly. $p1\_2$ drives the expression of the c_2 repressor gene, and $p1\_3$ drives the expression of the c_3 repressor gene.

4.5. Approach II—A Template-Driven Approach

In the second approach, we utilize *templates*, which are abstracted and composite *Positioning* rules. *Templates* enable us to specify "patterns" regarding the order and arrangement of genetic elements. Therefore, we offer in miniEugene so-called *Template Constraints* which are divided into the following two types.

—A *Template* restricts or ensures the total order of the elements in a design, enabling to restrict or ensure the exact positioning of the genetic elements in the entire design.

r1-c1-t1

```
// TEMPLATE
TEMPLATE r1,c1,t1
// ORIENTATION
ALL_FORWARD
```

r2-c2-t2

```
// TEMPLATE
TEMPLATE r2,c2,t2
// ORIENTATION
ALL_FORWARD
```

rOut1-cOut1-tOut1

```
// TEMPLATE
TEMPLATE rOut1,cOut1,tOut1
// ORIENTATION
ALL_FORWARD
```

Fig. 6. Templates for specifying RBS-GENE-TERMINATOR Compositions.

out1 Signal

```
// TEMPLATES
SEQUENCE pIn2,pIn1,r1,c1,t1
SEQUENCE p0,p1,r2,c2,t2
SEQUENCE p2,rOut1,cOut1,tOut1

// ORIENTATION
ALL_FORWARD

// INTERACTIONS
in2 INDUCES pIn2
in1 INDUCES pIn1
0 INDUCES p0

c1 REPRESSES p1
c2 REPRESSES p2
```

out0 Signal

```
// TEMPLATES
SEQUENCE p1,pIn2,r3,c3,t3
SEQUENCE pIn2,pIn0,r4,c4,t4
SEQUENCE p3,p4,r5,c5,t5
SEQUENCE p5,rOut0,cOut0,tOut0

// ORIENTATION
ALL_FORWARD

// INTERACTIONS
in2 INDUCES pIn2
in0 INDUCES pIn0

c3 REPRESSES p3
c4 REPRESSES p4
c5 REPRESSES p5
```

Fig. 7. Templates for specifying the Priority Encoder's out_1 and out_0 signals.

—A *Sequence* enables to group genetic elements and to constrain their total order within the set, but not within the entire design.

Iteration 1. In the first iteration, we specify templates to design the RBS-GENE-TERMINATOR compositions. As shown in Figure 6, we define a miniEugene TEMPLATE constraining the total order of the r_1-c_1-t_1 triplet.

Iteration 2. To combine the templates of the first iteration, we first need to convert the templates into Sequences. In Figure 7, we illustrate the miniEugene scripts to specify the out_1 and out_0 signals of the genetic Priority Encoder circuit. For example, the design of the out_1 signaling device has a length of 14 and three SEQUENCE constraints. As a result and as demonstrated in the subsequent section, we receive all possible permutations of the three out_1 and four out_0 compositions.

Iteration 3. In the final iteration, we only need to combine the two miniEugene scripts of Iteration 2 and set the length N of the genetic Priority Encoder design to 33.

4.6. Enumerated Designs

In both approaches—rule-based and template-driven—miniEugene returns an equivalent set of designs. In Figure 8, we visualize an excerpt of the enumerated designs using Pigeon [Bhatia and Densmore 2013].

In the first iteration, we specified the RBS-GENE-TERMINATOR compositions. Following the rule-based approach we had to specify seven rules for each of the seven compositions. Following the template-driven approach, we had the specify for each of the seven compositions one template.

In the second iteration, we composed the RBS-GENE-TERMINATOR compositions into the *out*1 and *out*0 signals of the Priority Encoder circuit. For the *out*1 signal, there are in total six possible permutations of arranging the compositions. For the *out*0 signal, there are in 24 permutations. In Figure 8, we visualize five randomly chosen *out*1 and *out*0 compositions. Following the rule-based approach, we had to specify 41 and 55 rules for the *out*1 and *out*0 signals, respectively. Following the template-driven approach

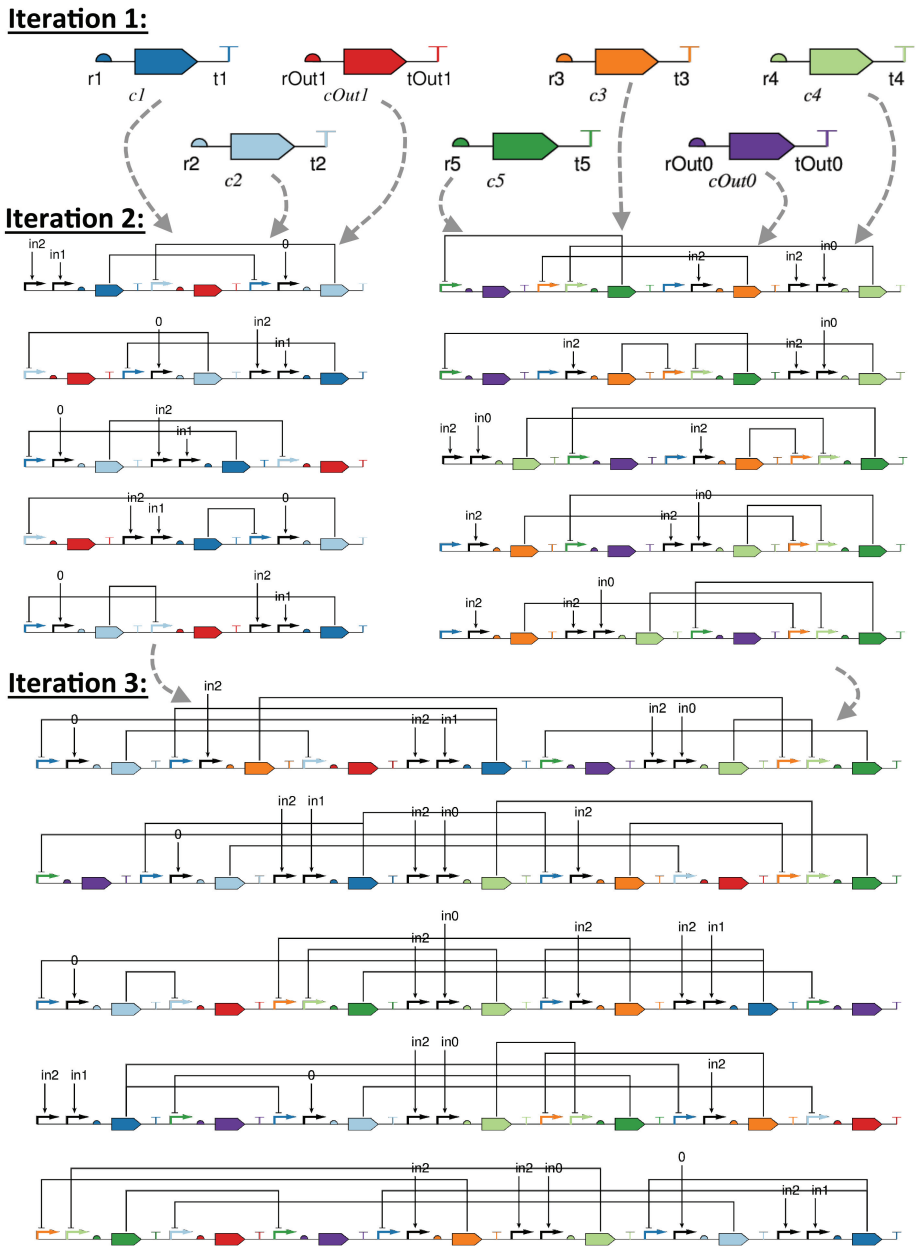


Fig. 8. Visualization of each Iteration's enumerated designs.

for the *out*1 signal, we had to specify three sequences, one *Orientation* rules, and five *Interaction* rules. For the *out*0 signal, we had to specify four sequences, one *Orientation* rule, and five *Interaction* rules.

In the final iteration, we had to specify the rules to arrange all seven compositions. As a result, there are 5,040 possible permutations (7!) and we visualize five randomly chosen in Figure 8. Following the rule-based approach, we had to specify 97 rules.

Following the template-driven approach, we specified the seven templates, and had to add one *Orientation* rule, and nine *Interaction* rules which reflect the regulatory interactions in the genetic Priority Encoder design of Figure 3(b).

4.7. Discussion on the Rule-Based and Template-Driven Paradigms

Now, we discuss the forces and consequences of the rule-based and template-driven paradigms based on questions that emerged while performing the case studies.

Q-1. How does each paradigm incorporate *a priori* knowledge of the partial or total order of arranging parts?

Following the rule-based paradigm, *Positioning* rules can be neglected if there's no *a-priori* knowledge on the order of the parts, leading to a more complex design space exploration. The expressivity of the template-driven paradigm as presented in this paper requires *a priori* knowledge of the parts order. In general, *a priori* knowledge about the *Positioning* of parts reduces the design space.

Q-2. How does each paradigm support human users?

We discovered, that a larger number of rules is difficult to manage for humans, increasing the feasibility that conflicting rules are being specified. *Templates*, on the other hand, provide a more user-friendly and manageable solution for the specification of, mainly *Positioning* rules.

Q-3. How does each paradigm ease debugging conflicting rules?

A large number of Rules is hard to debug for humans. Since *templates* are syntactic sugar for *Positioning* rules, it makes it only easier to detect and debug conflicts regarding *Positioning* rules.

Q-4. How does each paradigm affect reusability in an iterative design process?

A rule-based approach eases the composition of designs by building only the union of each design's rules. This enhances the rule's reusability, but might also need manual (or automated) modifications. templates ease the (hierarchical) composition of designs. However, if templates and their related rules are used to compose designs, then manual (or automated) modifications might be needed.

Q-5. How does each paradigm impact the enumeration or verification of rule-compliant designs?

The efficiency of evaluating rules depends on the underlying model, algorithm, and data structures. The complexity of enumerating the designs depends on the type of the allowed constraints [Chen et al. 2006].

5. A METHODOLOGY TO APPROXIMATE THE NUMBER OF RULE-COMPLIANT DESIGNS

In this section, we describe how the number of valid designs grow compared to the length of each design, N, and the size of the user specified part library, k. Without any rules, there are $2^N k^N$ possible designs, with the 2^N term due to the directionality of parts. Since rules can only eliminate designs, this is an upper bound for the number of valid devices. Thus, in all cases, the number of valid designs grows at most polynomial in the size of the part library. However, it can grow exponentially in the length of the design. Because the number of valid designs can quickly grow larger than what a computer can generate, we present counting as a way to explore large design spaces, as *a priori* knowledge of the number of valid designs is useful for determining statistical significance when only some designs have been generated as well as calculating the largest design length that can be worked with.

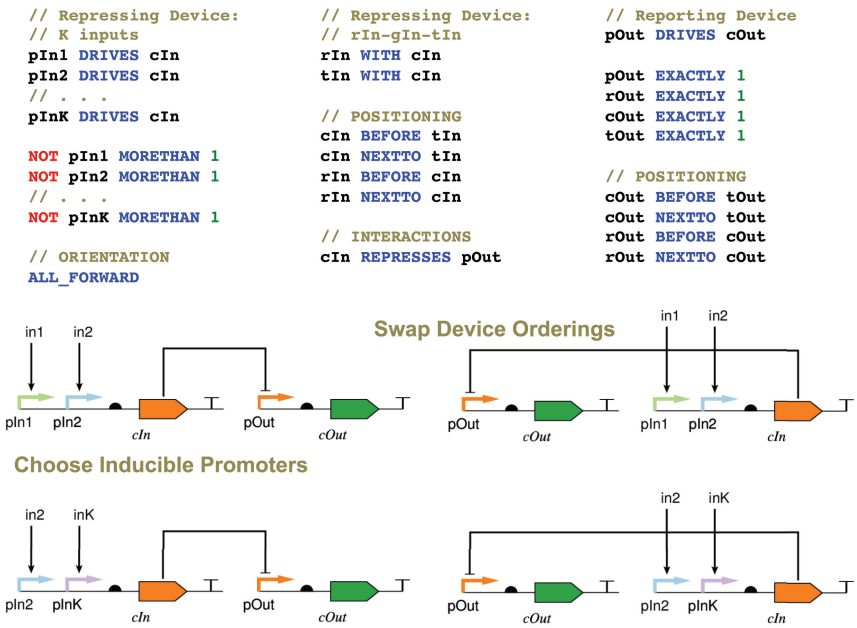


Fig. 9. Rules and enumerated transcriptional NOR gates.

Now, we count the number of rule-compliant NOR-gates. In Figure 9, we demonstrate the rules for specifying all possible $\overline{N}$ input transcriptional NOR gates that can be made up from $\overline{k}$ possible inducible promoters, where $\overline{N} = N - 7$. We restrict $\overline{N} \geq 1$ and $\overline{k} \geq 1$ to have a non-empty solution set. Notice that two decisions can be made that uniquely describe each design—the inducible promoters chosen and the order of the repressing and reporting device. In Figure 9, the decisions are annotated with *Choose Inducible Promoters* (up-down) and *Swap Device Orderings* (left-right), respectively. The former has $\overline{N}$-permutations of $\overline{k}$ possible choices and the latter has two (2) possible choices. Since the decisions are independent the total number of designs is the product of the two which gives a total of

$$\frac{(\overline{k}!)}{(\overline{k} - \overline{N})!} * 2$$

possible designs.

In Table VII, we display the number of valid designs after a single rule is applied to parts α and β, with an ALL_FORWARDS rule added for cases where it is not yet known how to calculate the number of designs without it. To calculate the EXACTLY rule, notice that once α has been placed there are $(k-1)^{N-n}$ ways to place the other parts. Multiplying this with the $\binom{N}{n}$ unique ways to place α in n spots gives the total number of designs. The EXACTLY rule can be used to calculate other counting rules by summing over n, such as the MORETHAN rule.

We now demonstrate how to calculate the number of valid designs with a BEFORE rule applied. To calculate the number of valid designs, consider the case where part α appears exactly m times and the last appearance of α is at position i. Then, we can divide the design into two pieces: the first one ranging from position 1 to $i-1$ and the second one ranging from position $i+1$ to N. The first piece has exactly $m-1$ instances

Table VII. Calculating the Design Space Size based on Rules ($\alpha \neq \beta$, $n \in \mathbb{N}$)

Rule	Amount of Designs
Unrestricted	$2^N k^N$
DESIGN TEMPLATES	$2k_1 \times 2k_2 \times \cdots \times 2k_N$
α **EXACTLY** $n \wedge$ **ALL_FORWARD**	$\binom{N}{n}(k-1)^{N-n}$
CONTAINS α	$k^N - (k-2)^N$
α **ALL_BEFORE** $\beta \wedge$ **ALL_FORWARD**	$(k-1)^N + \sum_{m=1}^{N} \sum_{i=m}^{N} \binom{i-1}{m-1}(k-2)^{i-m}(k-1)^{N-i}$
α **MORETHAN** $n \wedge$ **ALL_FORWARD**	$\sum_{i=n+1}^{N} \binom{N}{i}(k-1)^{N-i}$

of α, and since the rest of the positions cannot be filled with α or β, which—using the EXACTLY formula—results in $\binom{i-1}{m-1}(k-2)^{i-m}$ possible designs. The second piece can be anything but α, resulting in $(k-1)^{N-i}$ possible designs. The multiplication gives the total number of designs with m instances of α with the last instance appearing at position i. Now if m ranges from 1 to N, i must range from m to N. Summing over these bounds and adding in the number of designs that have no instances of α gives the final solution.

If a design specification consists of rules not listed in Table VII, then we can define an upper bound of the number of designs. Therefore, we can use the listed rule calculations, along with related rules and those modified by the negation operator (NOT). For example, if $D(R)$ returns the number of designs specified by rule R, then the upper bound of a conjunction of rules can be defined as: $D(R_1 \wedge R_2 \wedge \cdots \wedge R_r) \leq \min(D(R_1), D(R_2), \ldots, D(R_r)) \leq D(\text{UNRESTRICTED})$. For a disjunction of rules, we can define the upper bound as follows: $D(R_1 \vee R_2 \vee \cdots \vee R_r) \leq D(R_1) + D(R_2) + \cdots + D(R_r) \leq r\,D(\text{UNRESTRICTED})$.

6. A SURVEY OF RULE-BASED BDA TOOLS FOR SYNTHETIC BIOLOGY

This section provides a brief survey of BDA tools that enable the specification of rules. We survey the tools according their application in the synthetic biology design process and what types of rules are supported.

GenoCAD [Czar et al. 2009] and the Genetic Engineering of Cells (GEC) are languages enabling the management of libraries of genetic parts and to specify and constrain their part compositions. GenoCAD's utilization of context-free grammars for rule-based part compositions, supporting (1) the specification of hierarchical compositions and (2) more expressive *Positioning*, *Counting*, *Pairing*, and *Orientation* rules. Recently, GenoCAD's applicability as a rule-based design tool has been published in [Purcell et al. 2013]. GEC enables the specification of *Interaction* rules among genes and proteins. Programs in GEC are compiled into sequences of genetic parts using logical programming and knowledge about chemical reactions.

The LifeTechnologies' VectorNTI Express Designer and DeviceEditor [Chen et al. 2012] are BDA tools that incorporate Eugene, and thereby rules, into the design process of synthetic biological systems. Both tools offer features to manage libraries of genetic parts. For combinatorial design problems, Vector NTI Express Designer utilizes Eugene to automatically generate all possible combinations of parts into devices and systems. In DeviceEditor, the DNA constructs under design are specified as a sequence of "bins", each being populated with selected parts from the part library. DNA

constructs can either be “circular” or “linear”. In linear DNA constructs, the sequence of bins is comparable to a *template*, enabling the specification of *Positioning* rules. The applicability of *Positioning* rules on circular constructs needs, however, further investigation. DeviceEditor also enables to constrain the number of occurrences of particular genetic parts, hence supporting *Counting* and *Pairing* rules. Also, *Orientation* rules are supported since the user can map DNA sequences onto parts and specify the part’s orientation.

7. CONCLUSION AND FURTHER WORK

This work builds the basis of a *rule-based* design paradigm for Synthetic Biology. In early iterations of engineering novel biological systems, a large number of possible compositions of parts, devices, or systems exists. Human competences cannot cope with enumerating all possible combinations and, moreover, not all of them will work in biology. Hence, computational support is required which incorporates “rules” inferred from biological discoveries into the design process.

In this work, we presented a *rule-based* design specification language. We (1) formally defined the semantics of design-specific rules based on the language of first-order logic and (2) characterize the expressiveness of the rules. We divide the rules into five categories: *Counting*, *Pairing*, *Positioning*, *Orientation*. and *Interaction*. We have implemented all the formalized rules, as specified in this work, in miniEugene which is available at www.eugenecad.org. We demonstrated its applicability on a case study in which we iteratively designed a genetic Priority Encoder circuit “wired” of transcriptional NOR gates. The case study and our approaches should act as guidelines how to utilize miniEugene, its rules, and design facilities.

Due to size and complexity of combinatorial design spaces, it is not always advisable to enumerate, generate, and physically build all designs up front. Therefore, we have also touched a strategy to approximate the number of rule-compliant designs. Such methods enable to estimate and approximate how many rule-compliant designs exist, how long it would take to physically build and test them, and how costly the experiments could be.

Sophisticated models, efficient algorithms, and standardized data structures influence the computational niche of synthetic biology. Although the contributions of this work are still in its infancy, we believe that the *rule-based* design paradigm will become more prominent in the specification, design, and communication of synthetic biological systems.

ACKNOWLEDGMENTS

We thank Haiyao “Cassie” Huang for her help on implementing several pieces of the Eugene language. We dedicate this article to Allan Kuchinsky who advised and supported the research and development of the Eugene language and the rules.

REFERENCES

E. Andrianantoandro, S. Basu, D. K. Karig, and R. Weiss. 2006. Synthetic biology: New engineering rules for an emerging discipline. *Molec. Syst. Biol.* 2. DOI:http://dx.doi.org/10.1038/msb4100073

Swapnil Bhatia and Douglas Densmore. 2013. Pigeon: A design visualizer for synthetic biology. *ACS Synthet. Biol.* 2, 6, 348–350. DOI:http://dx.doi.org/10.1021/sb400024s

Lesia Bilitchenko, Adam Liu, Sherine Cheung, Emma Weeding, Bing Xia, Mariana Leguia, J. Christopher Anderson, and Douglas Densmore. 2011. Eugene: A domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS ONE* 6, 4 (April), e18882. DOI:http://dx.doi.org/10.1371/journal.pone.0018882

Lukasz J. Bugaj and David V. Schaffer. 2012. Bringing next-generation therapeutics to the clinic through synthetic biology. *Curr. Opin. Chem. Biol.* 16, 3, 355–361. DOI:http://dx.doi.org/10.1016/j.cbpa.2012.04.009

Stefano Cardinale and Adam Paul Arkin. 2012. Contextualizing context for synthetic biology—identifying causes of failure of synthetic biological systems. *Biotech. J.* 7, 7, 856–866. DOI:http://dx.doi.org/10.1002/biot.201200085

Joanna Chen, Douglas Densmore, Timothy Ham, Jay Keasling, and Nathan Hillson. 2012. DeviceEditor visual biological CAD canvas. *J. Biol. Eng.* 6, 1, 1. DOI:http://dx.doi.org/10.1186/1754-1611-6-1

Su Chen, Tomasz Imielinski, Karin Johnsgard, Donald Smith, and Mario Szegedy. 2006. A dichotomy theorem for typed constraint satisfaction problems. In *Proceedings of the Symposium on the Theory and Applications of Satisfiability Testing (SAT'06)*. Armin Biere and Carla P. Gomes (Eds.), Lecture Notes in Computer Science, vol. 4121, Springer, Berlin Heidelberg, 226–239. DOI:http://dx.doi.org/10.1007/11814948_23

Michael J. Czar, Yizhi Cai, and Jean Peccoud. 2009. Writing DNA with GenoCAD. *Nucl. Acids Res.* 37, Web-Server-Issue (2009), 40–47.

Douglas Densmore and Soha Hassoun. 2012. Design automation for synthetic biological systems. *IEEE Design Test Comput.* 29, 3, 7–20.

D. Densmore, J. T. Kittleson, L. Bilitchenko, A. Liu, and J. C. Anderson. 2010. Rule based constraints for the construction of genetic devices. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. 557–560. DOI:http://dx.doi.org/10.1109/ISCAS.2010.5537540

Drew Endy. 2005. Foundations for engineering biology. *Nature* 438, 7067, 449–453. DOI:http://dx.doi.org/10.1038/nature04342

Michal Galdzicki, Kevin P. Claney, Ernst Oberortner, et al. 2014. The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotech.* 32, 6 (June), 545–550. DOI:http://dx.doi.org/10.1038/nbt.2891

T. S. Gardner, C. R. Cantor, and J. J. Collins. 2000. Construction of a genetic toggle switch in *Escherichia coli*. *Nature* 403, 6767, 339–342. DOI:http://dx.doi.org/10.1038/35002131

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation* (3rd Ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

Julius B. Lucks, Lei Qi, Weston R. Whitaker, and Adam P. Arkin. 2008. Toward scalable parts families for predictable design of biological circuits. *Curr. Opin. Microbiol.* 11, 6, 567–573. DOI:http://dx.doi.org/10.1016/j.mib.2008.10.002, Growth and Development: Eukaryotes/Prokaryotes.

R. McNaughton and S. Papert. 1971. *Counter-free automata*. Tech. Rep. 65, MIT.

Oliver Purcell, Jean Peccoud, and Timothy K. Lu. 2014. Rule-based design of synthetic transcription factors in eukaryotes. *ACS Synthet. Biol.* 3, 10, 737–744. DOI:http://dx.doi.org/10.1021/sb400134k

Priscilla E. M. Purnick and Ron Weiss. 2009. The second wave of synthetic biology: From modules to systems. *Nat. Reviews. Mole. Cell Biol.* 10, 410–22. DOI:http://dx.doi.org/10.1038/nrm2698

Adrian Randall, Patrick Guye, Saurabh Gupta, Xavier Duportet, and Ron Weiss. 2011. Chapter Seven – Design and connection of robust genetic circuits. In *Synthetic Biology, Part A*, Chris Voigt (Ed.), Methods in Enzymology, vol. 497, Academic Press, 159–186. DOI:http://dx.doi.org/10.1016/B978-0-12-385075-1.00007-X

Warren C. Ruder, Ting Lu, and James J. Collins. 2011. Synthetic biology moving into the clinic. *Science* 333, 6047, 1248–1252. DOI:http://dx.doi.org/10.1126/science.1206843

Sumitra Shankar and M. Radhakrishna Pillai. 2011. Translating cancer research by synthetic biology. *Molec. BioSyst.* 7, 6, 1802–1810. DOI:http://dx.doi.org/10.1039/C1MB05016H

Alvin Tamsir, Jeffrey J. Tabor, and Christopher A. Voigt. 2011. Robust multicellular computing using genetically encoded NOR gates and chemical "wires". *Nature* 469, 7329, 212–215. DOI:http://dx.doi.org/10.1038/nature09565

Received January 2014; revised May 2014; accepted June 2014