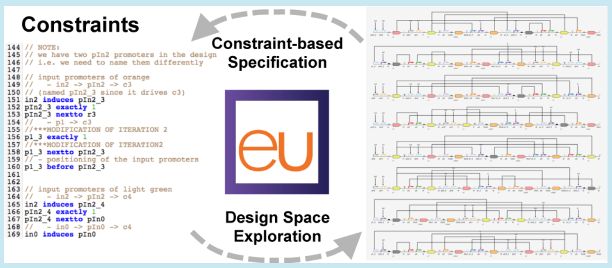
Web-Based Software Tool for Constraint-Based Design Specification of Synthetic Biological Systems

Ernst Oberortner[†] and Douglas Densmore*[,†,‡]

[†]Department of Electrical and Computer Engineering and [‡]Center of Synthetic Biology (CoSBi), Boston University, Boston, Massachusetts 02215, United States

ABSTRACT: *miniEugene* provides computational support for solving combinatorial design problems, enabling users to specify and enumerate designs for novel biological systems based on sets of biological constraints. This technical note presents a brief tutorial for biologists and software engineers in the field of synthetic biology on how to use *miniEugene*. After reading this technical note, users should know which biological constraints are available in *miniEugene*, understand the syntax and semantics of these constraints, and be able to follow a step-by-step guide to specify the design of a classical synthetic biological system—the genetic toggle switch.[1] We also provide links and references to more information on the *miniEugene* web application and the integration of the *miniEugene* software library into sophisticated Computer-Aided Design (CAD) tools for synthetic biology (www.eugenecad.org).



```
// NOTE:
// we have two pIn2 promoters in the design
// i.e. we need to name them differently

// input promoters of orange
//    - in2 -> pIn2 -> c3
// (named pIn2_3 since it drives c3)
in2 induces pIn2_3
pIn2_3 exactly 1
pIn2_3 nextto r3
//    - p1 -> c3
//***MODIFICATION OF ITERATION 2
p1_3 exactly 1
//***MODIFICATION OF ITERATION2
p1_3 nextto pIn2_3
// - positioning of the input promoters
p1_3 before pIn2_3


// input promoters of light green
//    - in2 -> pIn2 -> c4
in2 induces pIn2_4
pIn2_4 exactly 1
pIn2_4 nextto pIn0
//    - in0 -> pIn0 -> c4
in0 induces pIn0
```

■ THE MOTIVATION BEHIND *miniEugene*

Combinatorial design is emerging as one potential design paradigm for synthetic biology.[2] Taking a library of biological parts and combining them to create designs can be a powerful approach since relatively small libraries can encode many possible designs. The enumeration of these designs can be performed rapidly *in silico*. In order for combinatorial design to be automated and reproducible, however, the specification of biological parts and the constraints on their composition needs to be formalized.

The Eugene language—first released in 2009—was the result of an iGEM project to explore the notions of part and constraint specification.[3] After five years of development, the Eugene language has grown to include declarative features for efficient constraint-based exploration of designs and imperative features (such as conditional branches, loops, and function prototyping) for supporting the automated generation of part libraries and constraints. In addition, Eugene supports data exchange standards and repositories (such as GenBank, the Synthetic Biology Open Language (SBOL), the iGEM Registry) for the import and export of biological data and designs.

While these features of Eugene are powerful, many cannot be used to their full potential until a set of baseline commands is written by a software developer. Also, Eugene relies on the definition of a part library with specific part-level properties. Developing this library can be onerous and prevents early design space explorations in the initial prototyping stages. Furthermore, a reduced set of commands is needed to address the problem of introducing nondeveloper users to Eugene. *miniEugene* provides this set of simple yet powerful commands and enables beginners to prune a design space starting from a basic understanding of the biological parts involved. In this technical note, we demonstrate the use of *miniEugene* in a specifically developed web application. We also exemplify the grammar of the *miniEugene* language, moving toward the goal of standardizing the communication of designs for synthetic biology and their associated biological constraints.

■ THE *miniEugene* WEB APPLICATION

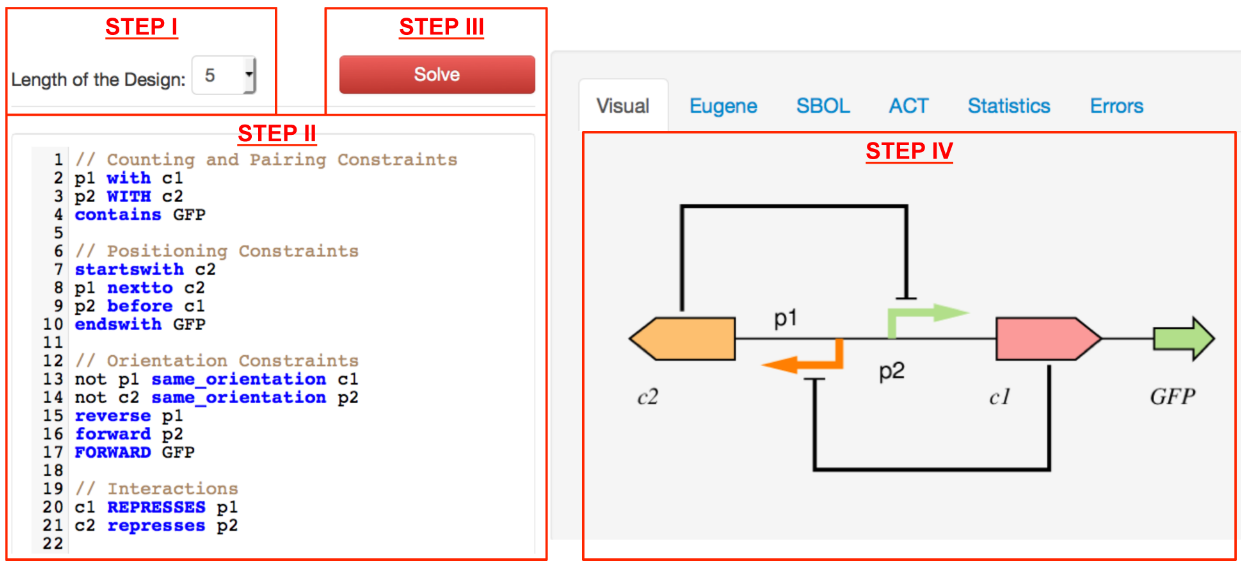
The *miniEugene* web application (accessible via www.eugenecad.org) provides an editor for the specification of *miniEugene* constraints and leverages the *miniEugene* software library to execute commands. In Figure 1, we illustrate the steps involved in using the *miniEugene* web application to specify the design for the genetic toggle switch.[1]

STEP I—"Specify Design Length". *miniEugene* requires the specification of the LENGTH OF THE DESIGN. For example, the toggle switch design has five available positions for placing the parts p_1, p_2, c_1, c_2 and *GFP* from the part library. The *miniEugene* web application provides a drop-down menu to choose the LENGTH OF THE DESIGN. For the toggle switch design specification, we select 5 from the drop-down menu.

STEP II—"Specify Design Constraints".

- In **lines 2—4**, we specify COUNTING and PAIRING constraints to ensure the correct number of occurrences for parts in the design: the two promoters (p_1, p_2) must occur pairwise WITH the two repressors (c_1, c_2), and the design CONTAINS one reporter (*GFP*).
- In **lines 7—10**, we ensure the biologically valid POSITIONING of the parts, that is, c_1, p_2, p_1, c_2, *GFP*.

Received: April 1, 2014
Published: November 26, 2014



```
 1 // Counting and Pairing Constraints
 2 p1 with c1
 3 p2 WITH c2
 4 contains GFP
 5
 6 // Positioning Constraints
 7 startswith c2
 8 p1 nextto c2
 9 p2 before c1
10 endswith GFP
11
12 // Orientation Constraints
13 not p1 same_orientation c1
14 not c2 same_orientation p2
15 reverse p1
16 forward p2
17 FORWARD GFP
18
19 // Interactions
20 c1 REPRESSES p1
21 c2 represses p2
22
```

Figure 1. Designing the genetic toggle switch with *miniEugene*.

- In **lines 13—17**, we constrain the ORIENTATION of the parts. Given the POSITIONING of the repressor-promoter pairs—(c_1, p_1) and (c_2, p_2)—within the design, they are NOT allowed to have the SAME_ORIENTATION. In addition, we specify that the p_1 promoter must be REVERSE oriented, while the p_2 and *GFP* part must be FORWARD oriented.
- In **lines 20—21**, we specify desired regulatory INTERACTIONS: c_1 REPRESSES p_1 and c_2 REPRESSES p_2.

Pushing the "Solve" button after specifying each set of constraints helps to debug the design space exploration process and to review how the constraints prune the design space.

STEP III—"Design Generation". Now, we command the *miniEugene* constraint solver to enumerate all designs that comply with the specified constraints. This is done by clicking the "Solve" button. Currently, the *miniEugene* solver does not incorporate heuristics, which can lead to a design space exploration process that is costly in terms of time. Given the same set of constraints, however, the *miniEugene* solver always enumerates the same set of designs.

STEP IV—"Select Design Representation". Besides providing detailed statistics for the design space exploration process, the *miniEugene* web application can generate standardized representations for designs. First, *miniEugene* uses Pigeon[4] to graphically represent designs in terms of symbols that are compliant with the SBOL Visual standard.[5]

Second, *miniEugene* integrates the libSBOLj Java software library to export designs to SBOL,[6] thereby enabling the exchange of these designs with SBOL-compliant software tools. While the regulatory interaction information is not captured by the current version of SBOL (SBOL 1.1.0[7]), the SBOL community has recently proposed extensions to the SBOL data model[8] to address this shortcoming.

Third, *miniEugene* compiles a Eugene header file,[3] which contains the design and interaction specifications. This file provides a bridge to then proceed into the fully fledged Eugene environment for more detailed design specifications. For example, users can link the constraint operands (p_1, c_1, *GFP*, *etc.*) with additional characteristics defined in the Eugene language, such as static biological data (*e.g.*, DNA sequences) or experimentally measured dynamic characteristics (*e.g.*, expression level and rate of degradation).

■ THE BIOLOGICAL CONSTRAINTS PROVIDED BY *miniEugene*

In Table 1, we list all of the constraints currently supported by *miniEugene*, describe their semantics, and provide an example of using each one with parts from the toggle switch design (see Figure 1).

■ THE PROGRAMMATIC INTEGRATION OF *miniEugene*

miniEugene is open-source and freely available under the BSD 3-clause license (http://opensource.org/licenses/BSD-3-Clause). The *miniEugene* software library is implemented in Java and can be downloaded as Java ARchive (JAR) file format, enabling a rapid integration and utilization of *miniEugene*'s constraint specifications and combinatorial design space exploration functionalities in Java-based synthetic biology CAD tools. Additionally, the interfaces of the *miniEugene* software library are globally invokable via a web service that is based upon the XML-RPC data exchange protocol. More detailed information, such as examples, links, and documentation on the programmatic integration and utilization of the *miniEugene* software library and its supported interfaces is provided on the *miniEugene* Web site (www.eugenecad.org).

■ *miniEugene* IN USE AND FUTURE WORK

The *miniEugene* software library is being used in research projects, software prototypes, and synthetic biology CAD tools. Recently, we published the use of the *miniEugene* web application for the iterative design of a complex biological system, the genetic priority encoder.[9] The Boston University 2014 iGEM team worked toward the physical implementation of the priority encoder circuit with the support of the *miniEugene* web application (http://2014.igem.org/Team:BostonU). The web application is also being used by undergraduate and graduate researchers for the combinatorial design of genetic regulatory networks.

The Cello (http://www.cellocad.org) software prototype incorporates the *miniEugene* software library and uses its

Table 1. List of All Currently Provided Constraints in *miniEugene*

Counting Constraints	**to constrain the number of occurrences of components.**	
CONTAINS α	α must occur at least once.	CONTAINS *p1*
α MORETHAN k	α must occur more than k times.	*p1* MORETHAN *0*
α EXACTLY k	α must occur exactly k times.	*p1* EXACTLY *1*
α SAME_COUNT β	α must occur as many times as β.	*p1* SAME_COUNT *p2*
Pairing Constraints	**to constrain the pair-wise occurrences of components.**	
α WITH β	Both components (α and β) must occur.	*p1* WITH *p2*
α THEN β	If α does occur, then β must occur.	*c2* THEN *GFP*
Positioning Constraints	**to constrain the (relative and absolute) positions of components.**	
STARTSWITH α	α must occur at the first position.	STARTSWITH *p1*
ENDSWITH α	α must occur at the last position.	ENDSWITH *GFP*
[k] EQUALS α	α must occur at the k-th position.	[0] EQUALS *p1*
[k] EQUALS [l]	The names of the components at position k and l must be equal.	[0] EQUALS [4]
α ALL_BEFORE β	If α and β occur, then all α must occur at any position BEFORE the first occurrence of β.	*p1* ALL_BEFORE *p2*
α BEFORE β	(same as α ALL_BEFORE β)	*c1* BEFORE *GFP*
α SOME_BEFORE β	If α and β occur, then at least one α must occur at any position BEFORE the first occurrence of β.	*p1* SOME_BEFORE *p2*
α ALL_AFTER β	If α and β occur, then all α must occur at any position AFTER the last occurrence of β.	*p2* ALL_AFTER *p1*
α AFTER β	(same as α ALL_AFTER β)	*GFP* AFTER *c2*
α SOME_AFTER β	If α and β occur, then at least one α must be at any position AFTER the last occurrence of β.	*c2* SOME_AFTER *c2*
α ALL_NEXTTO β	If α and β occur, then all α must occur immediately NEXT TO (either left or right) all occurrences of β.	*p1* ALL_NEXTTO *c2*
α NEXTTO β	(same as α ALL_NEXTTO β)	*p2* NEXTTO *c1*
α SOME_NEXTTO β	If α and β occur, then at least one α must occur immediately NEXT TO (either left or right) at least one occurrence of β.	*GFP* SOME_NEXTTO *c1*
Orientation Constraints	**to constrain the direction/orientation of the components.**	
ALL_FORWARD	all components in the designs must be forward oriented.	ALL_FORWARD
ALL_REVERSE	all components in the design must be reverse oriented.	ALL_REVERSE
ALTERNATE_ORIENTATION	the orientation of the components must alternate.	ALTERNATE_ORIENTATION
ALL_FORWARD α	All occurrences of α must be forward oriented.	ALL_FORWARD *p2*
FORWARD α	(same as ALL_FORWARD α)	FORWARD *c1*
SOME_FORWARD α	At least one occurrence of α must be forward oriented.	SOME_FORWARD *GFP*
ALL_REVERSE α	All occurrences of α must be reverse oriented.	ALL_REVERSE *p1*
REVERSE α	(same as ALL_REVERSE α)	REVERSE *c2*
SOME_REVERSE α	At least one occurrence of α must be reverse oriented.	REVERSE *c2*
α SAME_ORIENTATION β	All occurrences of α must have the same orientation as all β	*p2* SAME_ORIENTATION *c2*
Interaction Constraints	**to constrain and/or specify interactions among the components.**	
α INDUCES β	α induces β.	*in1* INDUCES *pIn1*
α DRIVES β	α drives the expression of β, that is α and β must have the some orientation, α must occur upstream to β, and there must be no terminator between α and β.	*p1* DRIVES *c2*
α REPRESSES β	α represses β.	*c2* REPRESSES *p2*

constraints and design space exploration capabilities during the automated design of genetic regulatory circuits. The Eugene language also uses and integrates the *miniEugene* software library for specifying constraints and solving combinatorial design problems regarding the composition of genetic parts. Eugene, and hence *miniEugene*, have been integrated into the Vector NTI Express Designer (Life Technologies) and the jS Device Editor (JBEI).[10] Eugene has also been used for complex gene cluster analysis by MIT, BU, and Broad Institute researchers.[11]

In the future, we will support constraints to specify hierarchical composition of designs. For example, this could be accomplished by extending the CONTAINS constraint. In addition, we want to determine automatically the minimum length of a design based on the specified constraints and their operands, making the specification of the design's length optional. The *miniEugene* web application has a support forum that allows users to post questions, report bugs, and provide feedback, such as identifying limitations of the expressivity of the *miniEugene* constraints in light of novel biological designs and discoveries. As a result, we expect that the list of constraints and the community surrounding *miniEugene* and Eugene will grow quickly.

AUTHOR INFORMATION

Corresponding Author

*E-mail: dougd@bu.edu.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

The work was funded under National Science Foundation (NSF) grant 1147158 and by the Agilent Technologies' Applications and Core Technology University Research (ACT-UR) program. The authors thank Nicholas Roehner for his help in editing the manuscript and providing feedback on its contents. The authors also thank all other members of the CIDAR Lab for their intellectual contributions during collaborative discussions.

DEDICATION

We dedicate this manuscript to the memory of Allan Kuchinsky.

ABBREVIATIONS

SBOL,Synthetic Biology Open Language; XML-RPC,remote procedure call protocol using XML to encode its calls and HTTP as a transport mechanism; JAR,Java ARchive; CAD,computer aided design; iGEM,international genetically engineered machine

REFERENCES

(1) Gardner, T. S., Cantor, C. R., and Collins, J. J. (2000) Construction of a genetic toggle switch in *Escherichia coli*. *Nature 403*, 339−342.

(2) Endy, D. (2005) Foundations for engineering biology. *Nature 438*, 449−453.

(3) Bilitchenko, L., Liu, A., Cheung, S., Weeding, E., Xia, B., Leguia, M., Anderson, J. C., and Densmore, D. (2011) Eugene: A domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PLoS One 6*, e18882.

(4) Bhatia, S., and Densmore, D. (2013) Pigeon: A design visualizer for synthetic biology. *ACS Synth. Biol. 2*, 348−350.

(5) Quinn, J. et al. (2013) Synthetic Biology Open Language Visual (SBOL Visual), Version 1.0.0. *BioBricks Foundation Request for Comments (BBF RFC) #93*, DOI: 1721.1/78249.

(6) Galdzicki, M., et al. (2014) The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotechnol. 32*, 545−550.

(7) Galdzicki, M. et al. (2012) Synthetic Biology Open Language (SBOL) Version 1.1.0. *BioBricks Foundation Request for Comments (BBF RFC) #93*, DOI: 1721.1/78249.

(8) Roehner, N., Oberortner, E., Pocock, M., Beal, J., Clancy, K., Madsen, C., Misirli, G., Wipat, A., Sauro, H., and Myers, C. J. (2014) Proposed data model for the next version of the Synthetic Biology Open Language. *ACS Synth. Biol.*, No. 10.1021/sb500176h.

(9) Oberortner, E., Bhatia, S., Lindgren, E., Densmore, D. A rule-based design specification language for synthetic biology. *J. Emerg. Technol. Comput. Syst. (JETC)* (accepted).

(10) Chen, J., Densmore, D., Ham, T. S., Keasling, J. D., and Hillson, N. J. (2012) DeviceEditor visual biological CAD canvas. *J. Biol. Eng. 6*, 1.

(11) Smanski, M. J., Bhatia, S., Zhao, D., Park, Y., Woodruff, L. B. A., Giannoukos, G., Ciulla, D., Busby, M., Calderon, J., Nicol, R., Gordon, D. B., Densmore, D., and Voigt, C. A. (2014) Functional optimization of gene clusters by combinatorial design and assembly. *Nat. Biotechnol. 32*, 1241−1249.