

Automatic Compilation from High-Level Languages to Genetic Regulatory Networks

Jacob Beal
BBN Technologies
Cambridge, MA 02138
Email: jakebeal@bbn.com

Ting Lu
MIT
Cambridge, MA 02139
Email: tinglu@mit.edu

Ron Weiss
MIT
Cambridge, MA 02139
Email: rweiss@mit.edu

Designing an engineered genetic regulatory network to produce a desired behavior is an extremely difficult task. Even given a combinatorial library of standard biological parts, such as BioBricks[1], there are a wide range of potential interactions between DNA, signaling molecules, expression machinery, metabolic resources, etc. As a result, a designer must solve a complicated multi-dimensional constraint and optimization problem in order to produce a working system.

Our goal is to improve the design of complicated biological systems, by finding programming abstractions and compilation techniques suitable for biological systems and then applying compiler and optimization algorithms adapted from electronic computers. A biological system designer would thus begin by expressing desired system function using a biologically-focused high-level programming language. The compiler transforms this design systematically into a genetic regulatory network, optimizing to conserve scarce biological resources (e.g. metabolic load, applicable BioBrick parts). This design can then be simulated and finally realized in cells with DNA assembled using standard protocols such as BioBricks or BglBricks[2].

A number of other projects are also attempting to address problems in biological systems design, mostly either through modelling standards, such SBML[3] and CellML[4], or means to simplify biological model building, such as Antimony[5], little b[6], and ProMoT[7]. A few tools, like Eugene[8] and GenoCAD[9] are beginning to offer the ability to do “assembly-language” level composition of synthetic biology elements. Perhaps the most similar to this project is GEC[10], which attempts to automate the design process via iterative simulation.

We have chosen to work with designs expressed in the Proto spatial computing language[11], as it seems particularly well-matched to this goal:

- The Proto dataflow computation model matches well with the continuous parallel expression of proteins in genetic regulatory networks.

- Proto’s spatial primitives offer a path toward construction of complex multicellular systems, such as tissues and biofilms.
- We have previously shown that genetic regulatory networks generated from Proto programs are good targets for standard compiler optimization techniques[12].

At present, our compiler takes Proto programs expressed in a limited subset of the language and uses design motifs to transform them into a genetic regulatory network that uses chemical constants within the envelope of experimentally verified synthetic biological systems.

Motif-based compilation associates each high-level primitive with an abstract genetic regulatory network pattern. Using an extension to the Proto language, we associate high-level language primitives with design motifs. For example, a logical “not” operation is declared as follows:

```
(primitive not (boolean) boolean
:bb-template
((P 0.193 R- arg0 outputs T))
```

The first line declares the primitive operation “not” to be a function that takes a boolean as input and returns a boolean as output. The remainder of the expression annotates this high-level function with a BioBrick motif consisting of a single regulatory region. First comes a promoter (P) annotated with the regulatory region’s constitutive expression rate of 0.193 molecules per second. This promoter is repressed (R-) by `arg0`, the first input argument to the function. Then comes a placeholder for proteins representing the function’s output, followed by a terminator T.

More complex operations can be mapped to a motif involving multiple regulatory regions, such as this two-input logical “and”:

```
(primitive and (boolean boolean) boolean
:bb-template
```

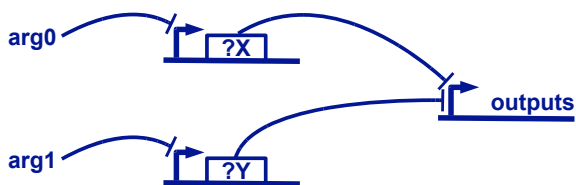


Fig. 1. Motif-based compilation associates each high-level primitive with an abstract genetic regulatory network pattern, such as this implementation of a two-input logical “and” operation. The compiler transforms each operation in the program to a motif, linking them together according to the flow of data in the computation.

```
( (P 0.193 R- arg0 ?X T)
  (P 0.193 R- arg1 ?Y T)
  (P 0.193 R- ?X R- ?Y outputs T) )
```

In this case, the local variables ?X and ?Y create connections between the three regulatory regions in the motif, implementing the logical “and” as a “nor” gate with inverters on each input (Figure 1).

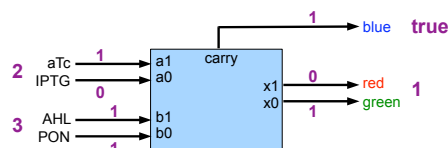
Given a set of such mappings and a program, the compiler transforms each operation in the program to a motif, linking them together according to the flow of data in the computation. The resulting genetic regulatory network can be output in several forms, including a set of generated MATLAB files that can be used for simulation of the system. Currently this models the system with ordinary differential equations, but in future work we plan to output stochastic reactions as well.

We have transformed a number of Proto programs into genetic regulatory networks using this compiler, and simulation in MATLAB verifies that the behavior of the genetic regulatory network correctly implements the original program, even for highly complex systems such as the two-bit adder shown in Figure 2, which the compiler currently transforms into an unoptimized network of 60 signal chemicals and 52 regulatory regions.

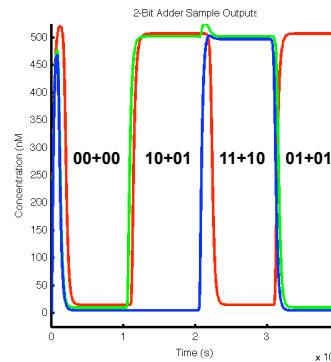
In future work, we plan to connect the compiler to existing parts libraries such that a genetic regulatory network can be compiled all the way to a DNA sequence and assembly instructions for constructing this sequence from standardized parts and to verify the behavior of compiled programs *in vivo*. We also plan to implement compiler optimizations, which we expect will improve generated program size radically, and to expand coverage to programs with extent in space and time.

REFERENCES

- [1] R. P. Shetty, D. Endy, and J. Thomas F Knight, “Engineering biobrick vectors from biobrick parts,” *Journal of Biological Engineering*, vol. 2, no. 5, 2008.
- [2] J. C. Anderson, J. E. Dueber, M. Leguia, G. C. Wu, J. A. Goler, A. P. Arkin, and J. D. Keasling, “Bglbricks: A flexible standard for biological part assembly,” *J. Biol. Eng.*, vol. 4, 2010.



(a) High-Level 2-Bit Adder



(b) Simulated chemical implementation

Fig. 2. A two-bit adder takes two 2-bit numbers as input and produces a 2-bit number as output, plus a carry bit that is true when the sum is above 3. An example, adding 2+3 to produce 5 (1+carry) is shown in purple in (a), with inputs encoded as small-molecule signals and outputs as fluorescent proteins. ODE chemical simulations of the compiled genetic regulatory network show correct implementation, as demonstrated by the examples in (b; offset vertically for visibility).

- [3] A. Finney, M. Hucka, B. J. Bornstein, S. M. Keating, B. M. Shapiro, J. Matthews, B. K. Kovitz, M. J. Schilstra, A. Funahashi, J. Doyle, and H. Kitano, “Software infrastructure for effective communication and reuse of computational models,” in *System Modeling in Cell Biology: From Concepts to Nuts and Bolts*, Z. Szallasi, J. Stelling, and V. Perival, Eds. MIT Press, 2006.
- [4] A. Garny, D. Nickerson, J. Cooper, R. W. dos Santos, A. Miller, S. McKeever, P. Nielsen, and P. Hunter, “Cellml and associated tools and techniques,” *Philos. Transact. A: Math Phys Eng Sci*, vol. 366, no. 1878, pp. 3017–43, September 2008.
- [5] L. P. Smith, F. T. Bergmann, D. Chandran, and H. M. Sauro, “Antimony: a modular model definition language,” *Bioinformatics*, vol. 25, no. 18, pp. 2452–54, 2009.
- [6] A. Mallavarapu, M. Thomson, B. Ullian, and J. Gunawardena, “Programming with models: modularity and abstraction provide powerful capabilities for systems biology,” *Journal of The Royal Society Interface*, vol. 6, no. 32, pp. 257–270, 2009.
- [7] S. Mirschel, K. Steinmetz, M. Rempel, M. Ginkel, and E. D. Gilles, “Promot: Modular modeling for systems biology,” *Bioinformatics*, vol. 25, no. 5, pp. 687–689, 2009.
- [8] Berkeley Software 2009 iGem Team, “Eugene,” http://2009.igem.org/Team:Berkeley_Software/Eugene, October 2009, Retrieved May 10, 2010.
- [9] M. Czar, Y. Cai, and J. Peccoud, “Writing dna with genocad,” *Nucleic Acids Research*, vol. 37, no. W40-7, 2009.
- [10] M. Pedersen and A. Phillips, “Towards programming languages for genetic engineering of living cells,” *Journal of the Royal Society Interface*, 2009.
- [11] J. Beal and J. Bachrach, “Infrastructure for engineered emergence in sensor/actuator networks,” *IEEE Intelligent Systems*, pp. 10–19, March/April 2006.
- [12] —, “Cells are plausible targets for high-level spatial languages,” in *Spatial Computing Workshop*, 2008.