Automatic Compilation from High-Level Bio-Languages to Genetic Regulatory Networks

Jacob Beal, Ting Lu, Ron Weiss

IWBDA, June 2010
Goal: High-Level Biological Design

High-Level Bio-focused Language

(defun band-detector (signal lo hi)
  (and (> signal lo)
       (< signal hi)))

(let ((v (diffuse (aTc) 0.8 0.05)))
  (green (band-detect v 0.2 1)))

Compile

Optimize

Simulate

Assemble

Genetic Regulatory Network
Tool Chain Vision

**Tool Chain**
- Programming language constructs
- Cell to cell communication
- State/time representation
- Analog computing components

**Feedback Loop**
- Add new abstractions in to the library
- Enhance the language with new constructs

**Stages for Engineering Cells**

1. **Organism Level Description**
   - High level simulator

2. **High Level Description**
   - Coarse chemical simulator

3. **Abstract Genetic Regulatory Network**
   - Detailed chemical simulator

4. **BioBrick Sequence**
   - Manufacturing and analyze new parts
   - Register new parts into the database

5. **Assembly Instructions**
   - Update simulation models to reflect the actual behavior
   - Update the parts database with newly discovered characteristics
   - Analyze the success/failure of assembly technique

6. **Testing**

7. **Cells**

**Tool Chain**
- Compiler
- Motif mapping
- Optimizations for regulatory network
- Constraint analysis

**Tool Chain**
- Part characterization databases
- Mapping between abstract and actual parts

**Tool Chain**
- Assembly workflow generator

**Tool Chain**
- Assembly robot
Tool Chain Vision

Stages for Engineering Cells

Tool Chain
- Programming language constructs
- Cell to cell communication
- State/time representation
- Analog computing components

Organism Level Description

High Level Description

Feedback Loop
- Add new abstractions in to the library
- Enhance the language with new constructs

Abstract Genetic Regulatory Network

Coarse chemical simulator

- Represent/handle new constraints
- Additional optimization techniques

BioBrick Sequence

Detailed chemical simulator

- Manufacture and analyze new parts
- Register new parts into the database

Assembly Instructions

Testing

- Update simulation models to reflect the actual behavior
- Update the parts database with newly discovered characteristics
- Analyze the success/failure of assembly technique

Cells

- Assembly workflow generator
- Assembly robot

- Part characterization databases
- Mapping between abstract and actual parts
Outline

• Compositional Design
• Motif-Based Compilation
• Simulation Results
Computation via Transcription Network

DNA promoter

regulatory protein

RNA polymerase

ribosome

RNA

Protein

Decay
Computation via Transcription Network

- RNA polymerase
- ribosome
- DNA promoter
- regulatory protein
- Protein
- RNA
- Decay
- Signal = Concentration

Stabilizes at decay = production

Alternatives:
- PoPS
- RNA concentration
Abstract GRN Design Space
Abstract GRN Design Space

\[ d[X] = R_i \cdot \frac{1 + K_Y^{-1}([Y]/D_Y)^{H_Y}}{1 + ([Y]/D_Y)^{H_Y}} \cdot \frac{1 + K_Z([Z]/D_Z)^{H_Z}}{1 + ([Z]/D_Z)^{H_Z}} \cdot \frac{\log(2)}{t_X[X]} \]

- \( R < 10 \text{ nM/s} \)
- \( K \in [2, 1000] \)
- \( H \in [1, 4] \)
- \( D \in [10, 10000] \text{ nM} \)
- \( t > 300 \text{ s} \)
- \( [X] \in [0, 1000] \text{ nM} \)
Major Challenge: Interference

- Effective part characteristics changed by:
  - Cellular context (endogenous pathways, synthetic parts)
  - Expression noise
- Our approach: noise-rejection
  - Digital - static discipline: $V_{\text{low, out}} < V_{\text{low, in}} < V_{\text{high, in}} < V_{\text{high, out}}$

But part variance makes a uniform standard impossible!
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Parameterized Standards Families

- Identify “standards family” parameter relation
- Create library of characterized part variants
- Adjust part choice to match on junctions
Experimental Input to Family Relation

Model constrained by characterization experiments...
Draft Simulation-Based Standards Family Generation

Need: K, H, D, R, t, max [X]

- Choose max [X]
- Assuming large K, amplification of 0.5K when [X]/D ≈ 1
- H → D, static discipline (higher H is better)
- Steady state max: [X] = production*t/log(2)
- production ≈ K*R

Example:
max [X] = 500 nM  
R = 0.193  
K = 101  
H = 3  
t = 1800 s  
D = 190
Draft Simulation-Based Standards Family Generation

Need: K, H, D, R, t, max [X]

- Choose max [X]
- Assuming large K, amplification of 0.5K when [X]/D \approx 1
- H \rightarrow D, static discipline (higher H is better)
- Steady state max: 
  \[ [X] = \text{production} \times \frac{t}{\log(2)} \]
- production \approx K \times R

Example:

max [X] = 500 nM  
R = 0.193  
\( t = 1000 \text{ s} \)  
K = 101  
D = 80
Draft Simulation-Based Standards Family Generation

Need: $K, H, D, R, t, \text{max } [X]$

- Choose max $[X]$
- Assuming large $K$, amplification of 0.5$K$ when $[X]/D \approx 1$
- $H \rightarrow D$, static discipline (higher $H$ is better)
- Steady state max: $[X] = \text{production} \times t / \log(2)$
  - production $\approx K \times R$

Driver for experimental part creation & characterization!

Example:
- $\text{max } [X] = 500 \text{ nM}$
- $H = 3$
- $R = 0.193$
- $t = 1000 \text{ s}$
- $K = 101$
- $D = 80$
Outline

- Compositional Design
- **Motif-Based Compilation**
- Simulation Results
Motif-Based Compilation

• High-level primitives map to GRN design motifs
  • e.g. logical operators:

  \[(\text{primitive not (boolean) boolean :bb-template ((P 0.193 R- arg0 RBS outputs T))})\]

---

(arg0) input

output
Motif-Based Compilation

- High-level primitives map to GRN design motifs
  - e.g. logical operators, actuators:

  \[
  \text{(primitive green (scalar) scalar :side-effect :bb-template ((P R+ arg0 RBS GFP outputs T)))}
  \]

![Diagram of a motif-based compilation process with placeholders for `arg0`, `GFP`, and an output arrow.]
Motif-Based Compilation

• High-level primitives map to GRN design motifs
  • e.g. logical operators, actuators, sensors:

\[
\text{(primitive IPTG () scalar :bb-template ((P 0.193 RBS LacI T) (rxn LacI (IPTG 180000) \rightarrow \text{LacI}*) (\text{P 0.193 R- LacI RBS outputs T})))}
\]

\[
\begin{array}{c}
\text{IPTG} \\
\downarrow \\
\text{LacI} \\
\end{array} \quad \begin{array}{c}
\text{output} \\
\end{array}
\]
Motif-Based Compilation

- Functional program gives dataflow computation:
  
  \[(\text{green} \ (\text{not} \ (\text{IPTG})))\]

- Operators translated to motifs:

- Standards family sets chemical constants
- Optimizers simplify network
Motif-Based Compilation

• Functional program gives dataflow computation:

\[(\text{green} \ (\text{not} \ (\text{IPTG})))\]
Motif-Based Compilation

• Functional program gives dataflow computation:

\[(\text{green} \ (\text{not} \ (\text{IPTG})))\]
Motif-Based Compilation

- Operators translated to motifs:
  - Standards family sets chemical constants

\[\text{IPTG} \rightarrow \text{not} \rightarrow \text{green}\]
Motif-Based Compilation

- Operators translated to motifs:
  - Standards family sets chemical constants

![Diagram showing the relationship between IPTG, LacI, A, not, and green]

- IPTG
- LacI
- A
- not
- green
Motif-Based Compilation

- Operators translated to motifs:
  - Standards family sets chemical constants

![Diagram](image.png)
Motif-Based Compilation

- Operators translated to motifs:
  - Standards family sets chemical constants
Motif-Based Compilation

- Optimizers simplify network
Motif-Based Compilation

- Optimizers simplify network
Motif-Based Compilation

- Optimizers simplify network
Outline

- Compositional Design
- Motif-Based Compilation
- Simulation Results
Simulation Results

- Prototype compiler generates GRNs that simulate correctly for a limited language subset.
- Example: 2-bit adder

```
  a1  a0  b1  b0  x1  x0  carry
```

Simulation Results

- Prototype compiler generates GRNs that simulate correctly for a limited language subset
- Example: 2-bit adder

\[\begin{array}{ccc}
2 & \rightarrow & a1 \\
& & a0 \\
3 & \rightarrow & b1 \\
& & b0 \\
\end{array}\]

\[\begin{array}{ccc}
\text{carry} & \rightarrow & x1 \\
& & x0 \\
\end{array}\]

\[\begin{array}{c}
\text{true} \\
\uparrow \\
1 \\
\end{array}\]
Simulation Results

- Prototype compiler generates GRNs that simulate correctly for a limited language subset
- Example: 2-bit adder
Simulation Results

- Prototype compiler generates GRNs that simulate correctly for a limited language subset
- Example: 2-bit adder

```
2
aTc  1
IPTG  0
3
C4HSL  1
3OC12HSL  1
```

```
1
```

```
true
```

```
blue
```

```
red
```

```
green
```

```
1
```

```
0
```

```
1
```

```
1
```

```
```

```
carry
```

```
a1
a0
```

```
b1
b0
```

```
x1
x0
```
Simulation Results

• Prototype compiler generates GRNs that simulate correctly for a limited language subset

• Example: 2-bit adder

(macro xor (a b)
  (muxor (muxand ,a (not ,b))
    (muxand ,b (not ,a))))

(macro 2bit-adder (a1 a0 b1 b0)
  (all
    (green (xor ,a0 ,b0)) ; x_0 low bit
    (let ((c0 (muxand ,a0 ,b0))
      (x1 (xor ,a1 ,b1)))
      (red (xor x1 c0)) ; x_1 high bit
      (blue (muxor (muxand x1 c0) ; carry bit (muxand ,a1 ,b1))))))

(2bit-adder (aTc) (IPTG) (C4HSL) (3OC12HSL))
Simulation Results

- Compiled 2-bit adder (unoptimized)
  - 60 signal chemicals
  - 52 regulatory regions
- Generated ODE simulation in MATLAB
On to optimization...

- Adapted classical techniques can be powerful:

\[
\text{(def band-detector (signal lo hi)} \\
\hspace{1cm} (\text{and (> signal lo)} \\
\hspace{2cm} (< \text{signal hi})) \) \\
\text{(let (v (diffuse (aTc) 0.8 0.05)))} \\
\text{(green (band-detect v 0.2 1)))}
\]
Tool Chain Vision

Stages for Engineering Cells

Tool Chain
- Programming language constructs
- Cell-to-cell communication
- State/time representation
- Analog computing components

Organism Level Description
- High level simulator

High Level Description
- Coarse chemical simulator

Abstract Genetic Regulatory Network
- Detailed chemical simulator

BioBrick Sequence
- Assembly workflow generator

Assembly Instructions
- Assembly robot

Feedback Loop
- Add new abstractions in to the library
- Enhance the language with new constructs

- Represent/handle new constraints
- Additional optimization techniques

- Manufacture and analyze new parts
- Register new parts into the database

- Update simulation models to reflect the actual behavior
- Update the parts database with newly discovered characteristics
- Analyze the success/failure of assembly technique

Testing

Cells
Contributions

• Parameterized standards identify all chemical parameters that can produce digital logic in transcriptional networks.
• Prototype motif-based compiler automatically maps high-level programs into GRNs.
• Automatically generated MATLAB ODE simulations verify that GRNs implement program specification.