High-Level Languages for Synthetic Biology

Jacob Beal

November, 2010

BBN: Fusun Yaman, Aaron Adler, Richard Schantz
MIT: Ron Weiss, Jonathan Babb, Noah Davidsohn
BU: Doug Densmore, Swapnil Bhatia
Agenda

- What's “high-level language”, and why bother?
- State of the Art
- Proto & Tool Chain
Definition

A “high-level” programming language is any language that abstracts away many of the details of how a computation will be implemented.
High-Level Language in SynBio

PoPS→PoPS→PoPS

‘Can I have three inverters?’

PoPS

‘I need a few DNA binding proteins.’

DNA

TAATACGACTCACTATAGGGAGA

‘Get me this DNA.’

PoPS

‘Here’s a set of PDP inverters, 1→N, that each send and receive via a fungible signal carrier, PoPS.’

PoPS→PoPS

‘Here’s a set of DNA binding proteins, 1→N, that each recognize a unique cognate DNA site, choose any.’

PoPS

‘Here’s your DNA.’

PoPS→PoPS

Zif268, Pervelich & Pabo c. 1991

‘Get me this DNA.’
High-Level Language in SynBio

Colony-Level Description

“Can I have this network of parts?”

“Here’s a set of parts, 1-N, that implement your network”

Systems

‘Can I have three inverters?’

‘Here’s a set of PDP inverters, 1→N, that each send and receive via a fungible signal carrier, PoPS.’

Devices

‘I need a few DNA binding proteins.’

‘Here’s a set of DNA binding proteins, 1→N, that each recognize a unique cognate DNA site, choose any.’

Parts

‘Get me this DNA.’

‘Here’s your DNA.’

DNA

TAATACGACTCACTATAGGGAGA
Why bother with HLLs?

- **Accessibility**: knowledge in software
- **Scalability**: routine work automated; higher portability
- **Reliability**: less human code; verification

*Is SynBio there yet? Maybe...*
HLL Design

• How will a SynBio system be described?

• How will a high-level description be transformed to *in vivo* execution?
HLL Design

• How will a SynBio system be described?
HLL Design

- How will a SynBio system be described?
HLL Design

- How will a SynBio system be described?
HLL Design

- How will a SynBio system be described?
HLL Design

• How will a SynBio system be described?

Syntax Is Nothing

Visual Basic

Python

Matlab

Java

C Programming Language
HLL Design

• How will a SynBio system be described?

**Syntax Is Nothing**

**Semantics Is Everything**
HLL Design

• How will a SynBio system be described?

Syntax Is Nothing
Semantics Is Everything

OBEY YOUR DOMAIN
What do we really need?

- Primitives
- Means of Combination
- Means of Abstraction

Everything else follows...
Agenda

- What's “high-level language”, and why bother?
- State of the Art
- Proto & Tool Chain
Computation via Transcription Network

- Regulatory protein
- RNA polymerase
- DNA promoter
- Ribosome
- RNA
- Protein
- Decay
Computation via Transcription Network

Stabilizes at $\text{decay} = \text{production}$
GenoCAD [Cai et al., '07]

CFG to generate/check part sequences
Eugene [Densmore et al., '10]

Device xor(cp, lacI, tetR, rpType1, gfp, rpType2, gfp);

Circuit “parts list” w. constraints, variables
GEC [Pederson & Phillips, '09]

Transcriptional logic programming
(def 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)))
Proto [Beal & Bachrach, '08]

- High-level primitives map to GRN design motifs

\[(\text{primitive and (boolean boolean) boolean}) \text{ : grn-motif } ((\text{P high R- } \text{ arg0 } \ ?X \ T) \ (\text{P high R- } \text{ arg1 } \ ?X \ T) \ (\text{P high R- } \ ?X \ \text{outputs} \ T))\]
Agenda

- What's “high-level language”, and why bother?
- State of the Art
- Proto & Tool Chain
Abstract GRN Design Space
Abstract GRN Design Space

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

Inputs:
- \([none]\)

Parameters:
- \(R < 10 \text{ nM/s}\)
- \(K \in [2,1000]\)
- \(H \in [1,4]\)
- \(t > 300 \text{ s}\)
- \([X] \in [0,1000] \text{ nM}\)
- \(D \in [10,10000] \text{ nM}\)
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...
Prototype Tool Chain

**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

**Cells**

**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
Motif-Based Compilation

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

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

```
arg0                  output
```

- Diagram of logical operator implementation
Motif-Based Compilation

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

(primitive green (scalar) scalar :side-effect :bb-template ((P R+ arg0 RBS GFP outputs T)))
Motif-Based Compilation

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

```
(primitive IPTG () scalar
  :bb-template ((P 0.193 RBS LacI T)
    (rxn LacI (IPTG 180000) -> LacI*)
    (P 0.193 R- LacI RBS outputs T)))
```

![Diagram of Motif-Based Compilation](image)
Motif-Based Compilation

- Functional program gives dataflow computation:

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

- Functional program gives dataflow computation:

\[(\text{green} \ (\text{not} \ (\text{IPTG})))\]

![Diagram showing dataflow computation with nodes for IPTG, not, and green connected in sequence.](image-url)
Motif-Based Compilation

- Operators translated to motifs:

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

- Operators translated to motifs:

```
IPTG not green
```

```
LacI outputs arg0 outputs GFP outputs
```

IPTG not green

```
A B
```

```
GFP outputs
```
Motif-Based Compilation

- Operators translated to motifs:

![Diagram](image-url)
Optimization

Copy Propagation
Optimization

Copy Propagation

Dead Code Elimination
Optimization

Copy Propagation

Dead Code Elimination

Dead Code Elimination
Example Complex Compilation

• 2-bit adder:

```
(def xor (a b)
  (or (and a (not b))
      (and b (not a))))

(def 2bit-adder (a1 a0 b1 b0)
  (green (xor a0 b0)) ; low bit
  (let ((c0 (and a0 b0))
        (x1 (xor a1 b1)))
    (red (xor x1 c0)) ; high bit
    (blue (or (and x1 c0) ; carry bit
              (and a1 b1))))

(2bit-adder (aTc) (IPTG) (C4HSL) (3OC12HSL))
```
Example Complex Compilation

• 2-bit adder:
## Sample Optimization Results:

<table>
<thead>
<tr>
<th></th>
<th>Proteins</th>
<th>Functional Units</th>
<th>Promoters (Repressed/Activated/Constitutive)</th>
<th>Delay Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single-Not</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unoptimized</td>
<td>4</td>
<td>4</td>
<td>4 (2/1/1)</td>
<td>3</td>
</tr>
<tr>
<td>Optimized</td>
<td>3</td>
<td>3</td>
<td>3 (2/0/1)</td>
<td>2</td>
</tr>
<tr>
<td>% Improvement</td>
<td>25%</td>
<td>25%</td>
<td>33%</td>
<td>33%</td>
</tr>
<tr>
<td><strong>Three-Gate</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unoptimized</td>
<td>10</td>
<td>10</td>
<td>9 (7/1/1)</td>
<td>5</td>
</tr>
<tr>
<td>Optimized</td>
<td>4</td>
<td>5</td>
<td>4 (3/0/1)</td>
<td>2</td>
</tr>
<tr>
<td>% Improvement</td>
<td>60%</td>
<td>50%</td>
<td>55%</td>
<td>60%</td>
</tr>
<tr>
<td><strong>Quad-Not</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unoptimized</td>
<td>7</td>
<td>7</td>
<td>7 (5/1/1)</td>
<td>6</td>
</tr>
<tr>
<td>Optimized</td>
<td>2</td>
<td>2</td>
<td>2 (1/0/1)</td>
<td>1</td>
</tr>
<tr>
<td>% Improvement</td>
<td>71%</td>
<td>71%</td>
<td>71%</td>
<td>83%</td>
</tr>
<tr>
<td><strong>2-bit adder</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unoptimized</td>
<td>55</td>
<td>56</td>
<td>53 (37/15/1)</td>
<td>12</td>
</tr>
<tr>
<td>Optimized</td>
<td>26</td>
<td>23</td>
<td>24 (19/4/1)</td>
<td>7</td>
</tr>
<tr>
<td>% Improvement</td>
<td>52%</td>
<td>59%</td>
<td>55%</td>
<td>42%</td>
</tr>
</tbody>
</table>
Prototype Tool Chain

Stages for Engineering Cells

Organism Level Description

High Level Description

Coarse chemical simulator

Abstract Genetic Regulatory Network

Detailed chemical simulator

BioBrick Sequence

Assembly Instructions

Cells

Tool Chain

• Programming language constructs
• Cell to cell communication
• State/time representation
• Analog computing components

Compiler
• Motif mapping
• Graphical user interface

Constraint analysis

Part characterization databases
• Mapping between abstract and actual parts

Assembly robot

Feedback Loop

• Add new abstractions into 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
Abstract Feature Mapping

- Parts Database: bipartite regulation graph

Find a non-conflicting subgraph isomorphic to design
Initial Feature Matching Results

- \(a_1 = P_4\)
- \(b_1 = R_1\)
- \(a_2 = P_2\)
- \(b_2 = R_3\)
- \(b_3 = R_2\)
Prototype Tool Chain

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

Testing

Feedback Loop
- Add new abstractions into 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

Cells
Clotho: Planning for Assembly
Automated Assembly
Toward a broadly integrated future...

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 into the library
- Enhance the language with new constructs

Abstract Genetic Regulatory Network

High level simulator

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

Assembly robot

Assembly workflow generator

Part characterization databases
- Mapping between abstract and actual parts

Cells
Available Free Open-Source Tools

Proto
http://proto.bbn.com

High level design, simulation

Clotho
http://clothocad.org

Data management, automation
Backup material
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
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
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

![Graph showing 2-bit adder sample outputs]