Utilizing Signal Temporal Logic to Characterize and Compose Modules in Synthetic Biology

Curtis Madsen, Prashant Vaidyanathan, Cristian-Ioan Vasile, Rachael Ivison, Junmin Wang, Calin Belta, and Douglas Densmore
Introduction

• One of the fundamental goals in the field of synthetic biology is to reliably engineer biological systems to respond to environmental conditions according to a pre-determined genetic program.

• Using Boolean logic functions, synthetic biologists have successfully engineered living cells to perform certain functions\(^1\).

• However, it has been difficult to realize the full potential of genetically encoded logic in practical applications without the ability to specify timing and performance of genetic circuits.

Performance Specifications

- To remedy this issue, we propose using performance specifications.
- For example, we can give a performance specification for a traffic light:
  - A traffic light should remain green *until* a pedestrian requests a walk signal. *Within 5 seconds* of receiving the request, the traffic light should change to yellow for 2 seconds, and then change to red for 30 seconds before switching back to green.
Performance Specifications

- Temporal logic can be used to write performance specifications as it allows for reasoning about behavior over time.
- In particular, we use Signal Temporal Logic (STL) as it allows for the specification of requirements on signals at specific times leading to a level of expressiveness necessary for genetic circuit design.

\[
\begin{align*}
[G \ [0,200) \ (aTc > 30 \land TetR > 30)] \land \\
[F \ [0,200) \ G \ [0,200) \ (TetR \leq 30)] \land \\
[G \ [400,600) \ (IPTG > 30)] \land \\
[F \ [400,800) \ (TetR > 30)]
\end{align*}
\]
Temporal Logic Syntax

• Logical Operators:
  • Conjunction: $\phi \land \psi$
  • Disjunction: $\phi \lor \psi$
  • Implication: $\phi \rightarrow \psi$
  • Negation: $\neg \phi$

• Temporal Operators:
  • Until: $\phi U \psi$
  • Future (Eventually): $F \phi$ (or $\diamond \phi$)
  • Globally: $G \phi$ (or $\square \phi$)
**Until Operator**

\[ p \mathrel{\mathrm{U}} q \]

- \( q \) holds at the current or a future position, and \( p \) has to hold until that position. At that position \( p \) does not have to hold any more.
Until Operator

\[ p \cup q \]

- q holds at the current or a future position, and p has to hold until that position. At that position p does not have to hold any more.
Until Operator

$p \mathcal{U} q$

- $q$ holds at the current or a future position, and $p$ has to hold until that position. At that position $p$ does not have to hold any more.
Future (Eventually) Operator

$F \ p$

- **Future**: $p$ eventually has to hold (somewhere on the subsequent path).
Future (Eventually) Operator

F p

- **Future**: p eventually has to hold (somewhere on the subsequent path).
Future (Eventually) Operator

$Fp$

- **Future**: $p$ eventually has to hold (somewhere on the subsequent path).
Globally Operator

G p

- **Globally**: p has to hold on the entire subsequent path.
Globally Operator

G p

- **Globally**: p has to hold on the entire subsequent path.
Globally Operator

\( G \ p \)

- **Globally**: \( p \) has to hold on the entire subsequent path.
Repressor Incoherent FeedForward Loop (RIFFL)
Potential Behaviors

- Depending on which repressor modules are used, different behaviors can be achieved.
Signal Temporal Logic (STL)

- These behaviors must be encoded in STL.
- For instance, when “In” is greater than 100, “Out” always eventually rises above 50 within 1000 time units.
- Also, when “In” is 0, “Out” always remains below 50.

This corresponds to the following STL:

\[ \text{[G [0,10000) (In > 100)] } \to [\text{G [0,10000) F [0,1000) (Out > 50)] } \land \text{ [G [0,10000) (In \leq 0)] } \to [\text{G [0,10000) (Out \leq 50)]} ] \]
Characterization – Temporal Logic Inference (TLI)

• Using supervised learning, TLI can be used to “learn” an STL formula from time series data.

• The current implementation of TLI works by finding optimal STL primitives and parameters:
  • $G_{[t_0,t_1]} x_i < k, \ F_{[t_0,t_1]} x_i > k,$
  • $G_{[t_0,t_1]} x_i > k, \ F_{[t_0,t_1]} x_i < k,$ ...
  • $t_0, t_1, k$ found by simulated annealing

\[ \Phi \]

\[ \begin{align*}
(F_{[12.6,10000]} & \ y > 282) \\
(G_{[382,9730]} & \ y < 191) \\
(F_{[192,10000]} & \ y > 167) \end{align*} \]

\[ \begin{array}{c}
\text{desired} \quad \text{undesired} \\
\text{desired} \quad \text{undesired}
\end{array} \]

\[ = \Phi \]

Composability in Synthetic Biology

- DNA segments representing genetic parts and modules can be composed to create genetic circuits.
STLb

• STL with added functionality:
  • Concatenation ($\bullet$) – allows one STL formula to connect to another in sequence.
  • Inputs – signals that are annotated as drivers of the formula.
  • Outputs – signals that are annotated as being produced by the formula.
  • Mapping – a collection of assignments among the inputs and outputs of STL formulae.

• For instance, $\phi$ is composed of $\phi_1$ and $\phi_2$:

\[
\begin{align*}
\phi_1 & : i_1 \rightarrow o_1, o_2 \\
\phi_2 & : i_1 \rightarrow o_1, o_2, o_3
\end{align*}
\]
**STLb**

- STL with added functionality:
  - Concatenation (•) – allows one STL formula to connect to another in sequence.
  - Inputs – signals that are annotated as drivers of the formula.
  - Outputs – signals that are annotated as being produced by the formula.
  - Mapping – a collection of assignments among the inputs and outputs of STL formulae.

- For instance, $\phi$ is composed of $\phi_1$ and $\phi_2$:
  - Concatenation – $\phi(x_1, y_1, y_2, y_3) = \phi_1(i_1, o_1, o_2) \cdot \phi_2(i_1, i_2, o_1, o_2, o_3)$. 

![Diagram](image-url)
STLb

• STL with added functionality:
  • Concatenation (•) – allows one STL formula to connect to another in sequence.
  • Inputs – signals that are annotated as drivers of the formula.
  • Outputs – signals that are annotated as being produced by the formula.
  • Mapping – a collection of assignments among the inputs and outputs of STL formulae.

• For instance, $\phi$ is composed of $\phi_1$ and $\phi_2$:
  • Concatenation – $\phi(x_1, y_1, y_2, y_3) = \phi_1(i_1, o_1, o_2) \cdot \phi_2(i_1, i_2, o_1, o_2, o_3)$.
  • Input mapping – ($\phi: x_1 = \phi_1: i_1$).
STLb

- STL with added functionality:
  - Concatenation (•) – allows one STL formula to connect to another in sequence.
  - Inputs – signals that are annotated as drivers of the formula.
  - Outputs – signals that are annotated as being produced by the formula.
  - Mapping – a collection of assignments among the inputs and outputs of STL formulae.

- For instance, $\phi$ is composed of $\phi_1$ and $\phi_2$:
  - Concatenation – $\phi(x_1, y_1, y_2, y_3) = \phi_1(i_1, o_1, o_2) \cdot \phi_2(i_1, i_2, o_1, o_2, o_3)$.
  - Input mapping – ($\phi: x_1 = \phi_1: i_1$).
  - Output mapping – ($\phi: y_1 = \phi_2: o_1 \land (\phi: y_2 = \phi_2: o_2) \land (\phi: y_3 = \phi_2: o_3)$.

![Diagram of STLb](attachment:diagram.png)
SLb

- STL with added functionality:
  - Concatenation (\(\cdot\)) – allows one STL formula to connect to another in sequence.
  - Inputs – signals that are annotated as drivers of the formula.
  - Outputs – signals that are annotated as being produced by the formula.
  - Mapping – a collection of assignments among the inputs and outputs of STL formulae.

- For instance, \(\phi\) is composed of \(\phi_1\) and \(\phi_2\):
  - Concatenation – \(\phi(x_1, y_1, y_2, y_3) = \phi_1(i_1, o_1, o_2) \cdot \phi_2(i_1, i_2, o_1, o_2, o_3)\).
  - Input mapping – \(\phi: x_1 = \phi_1: i_1\).
  - Output mapping – \(\phi: y_1 = \phi_2: o_1\) \(\land\) \(\phi: y_2 = \phi_2: o_2\) \(\land\) \(\phi: y_3 = \phi_2: o_3\).
  - Internal mapping – \(\phi_1: o_1 = \phi_2: i_1\) \(\land\) \(\phi_1: o_2 = \phi_2: i_2\).
**STL♭**

- **STL with added functionality:**
  - Concatenation (•) – allows one STL formula to connect to another in sequence.
  - Inputs – signals that are annotated as drivers of the formula.
  - Outputs – signals that are annotated as being produced by the formula.
  - Mapping – a collection of assignments among the inputs and outputs of STL formulae.

- For instance, $\phi$ is composed of $\phi_1$ and $\phi_2$:
  - Concatenation – $\phi(x_1, y_1, y_2, y_3) = \phi_1(i_1, o_1, o_2) \cdot \phi_2(i_1, i_2, o_1, o_2, o_3)$.
  - Input mapping – $(\phi: x_1 = \phi_1: i_1)$.
  - Output mapping – $(\phi: y_1 = \phi_2: o_1) \land (\phi: y_2 = \phi_2: o_2) \land (\phi: y_3 = \phi_2: o_3)$.
  - Internal mapping – $(\phi_1: o_1 = \phi_2: i_1) \land (\phi_1: o_2 = \phi_2: i_2)$.

Note: The mapping can be applied to other STL operators, not just concatenation.
<table>
<thead>
<tr>
<th>Genetic Modules</th>
<th>Name</th>
<th>STL Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Diagram" /></td>
<td>( m_1 )</td>
<td>( \phi_1 )</td>
</tr>
<tr>
<td><img src="image2" alt="Diagram" /></td>
<td>( m_2 )</td>
<td>( \phi_2 )</td>
</tr>
<tr>
<td><img src="image3" alt="Diagram" /></td>
<td>( m_3 )</td>
<td>( \phi_3 )</td>
</tr>
<tr>
<td><img src="image4" alt="Diagram" /></td>
<td>( m_4 )</td>
<td>( \phi_4 )</td>
</tr>
<tr>
<td><img src="image5" alt="Diagram" /></td>
<td>( m_5 )</td>
<td>( \phi_5 )</td>
</tr>
<tr>
<td><img src="image6" alt="Diagram" /></td>
<td>( m_6 )</td>
<td>( \phi_6 )</td>
</tr>
</tbody>
</table>
Design Space Exploration
Design Space Exploration

Design

STL Formula

\[
\Phi_1 \land \Phi_2 \land \Phi_3 \land \Phi_4 \land \Phi_5 \land \Phi_6
\]
Design Space Exploration

Design

STL Formula

$\Phi_3$
Design Space Exploration

STL Formula

\( \Phi_3 \land \Phi_2 \)
Design Space Exploration

\[ \Phi_3 \land \Phi_2 \land \Phi_4 \land \Phi_6 \]

STL Formula

\[ \Phi_3 \land \Phi_4 \]
Design Space Exploration

Design

STL Formula

$\Phi_3 \land \Phi_6$
Design Space Exploration

Design

STL Formula

\[ \Phi_3 \bullet (\Phi_2 \land \Phi_4) \]
Design Space Exploration

Design

STL Formula

\( \Phi_3 \land (\Phi_2 \land \Phi_6) \)
Design Space Exploration

Root

Design

STL Formula

$\Phi_3 \land (\Phi_4 \land \Phi_6)$
Design Space Exploration

Design STL Formula

\( \Phi_3 \land (\Phi_6 \land \Phi_1) \)
Design Space Exploration

Design

STL Formula

$$\Phi_3 \cdot ((\Phi_2 \land (\Phi_6 \cdot \Phi_1))$$
Design Space Exploration

Design

STL Formula

\[ \Phi_3 \land (\Phi_4 \land (\Phi_6 \land \Phi_1)) \]
Design Space Exploration

\[
\Phi_3 \land (\Phi_2 \land \Phi_4 \land (\Phi_6 \land \Phi_1))
\]
Constraint Pruning

Library

\[ m_1 \cdot \Phi_1 \]

\[ m_2 \cdot \Phi_2 \]

\[ m_3 \cdot \Phi_3 \]
Constraint Pruning

**Library**

- $m_1 \Phi_1$
- $m_2 \Phi_2$
- $m_3 \Phi_3$

**Cross Talk**

- $\Phi_2 \Phi_3 \Phi_3$

**Root**
Possible RIFFL Circuit Designs
Future Work

• Currently, TLI requires both desirable and undesirable traces.
  • We are working on a method that only requires desirable traces.

• We are adding constraints to help prune the potential design space of the composed genetic circuits.

• We are currently testing these methods on mammalian and bacterial synthetic biology examples.
Acknowledgements

Prashant Vaidyanathan
Cristian-Ioan Vasile
Rachael Ivison
Junmin Wang
Calin Belta
Douglas Densmore

This work is supported by the National Science Foundation under grant CPS Frontier 1446607.