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

Curtis Madsen[1], Prashant Vaidyanathan[1], Cristian-Ioan Vasile[2], Rachael Ivison[3], Junmin Wang[3], Calin Belta[2,3], and Douglas Densmore[1,4]

[1]Department of Electrical & Computer Engineering, Boston University, Boston, MA
[2]Division of Systems Engineering, Boston University, Boston, MA
[3]Graduate Program in Bioinformatics, Boston University, Boston, MA
[4]Biological Design Center, Boston University, Boston, MA
{ckmadsen,prash,cvasile,rivison,dawang,cbelta,dougd}@bu.edu

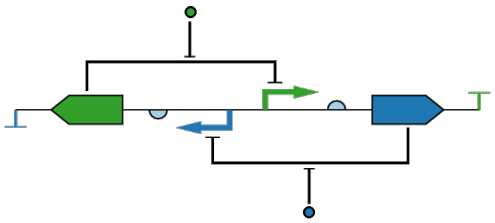
1. INTRODUCTION

The goal of synthetic biology is to allow biologists and engineers to design and build new biological systems. One way this task is achieved is though the composition of DNA segments representing genetic parts and modules. In synthetic biology, parts represent promoters, ribosome binding sites, genes, terminators, etc. while modules are comprised of these parts and include gates, switches, and oscillators. Each of these constructs has a function which can be specified in a formal way using a language such as the hardware description language Verilog. It has been shown that it is possible to reliably synthesize genetic circuits specified in this language using well established methods from logic synthesis in digital electronics [4].

Although previous approaches are very good at predicting the behavior of a designed circuit, they are Boolean in nature and do not include information about the performance of a design. Genetic circuit designs often contain real-time and real-valued constraints that can affect the dynamics of the system with varying levels of magnitude. To improve on previous approaches and include performance metrics in design specifications, temporal logics such as *signal temporal logic* (STL) [3] can be used. STL adds the ability to create specifications that include parameters intrinsic to genetic components, interactions with complex environments and other components, and timing of interactions and events.

For example, the genetic toggle switch shown in Figure 1(a) can be described by the STL formula in Figure 1(b). This STL formula states that the toggle switch starts in a state where both TetR and aTc are above a value of 30 for 200 time units. Within 200 time units, TetR falls below 30 and stays in that state for 200 time units. At this point, IPTG is added to the system and is held at a value above 30 for 200 time units. TetR is then expected to rise above 30 within 400 time units following the introduction of IPTG.

In the work presented here, we utilize an extension to STL called STLb that includes syntax and semantics for composition of genetic components [6]. Using *temporal logic inference* (TLI) [2], we can use experimental data to characterize genetic modules with STLb specifications. Our method can then build a design space tree by trying different compositions of the characterized modules. To improve efficiency, the design space tree is automatically pruned using biological constraints for assembly rules and failure mode checks.



(a)

$[G_{[0,200)}(\text{aTc} > 30 \wedge \text{TetR} > 30)] \wedge [F_{[0,200)}G_{[0,200)}(\text{TetR} \leq 30)] \wedge [G_{[400,600)}(\text{IPTG} > 30)] \wedge [F_{[400,800)}(\text{TetR} > 30)]$

(b)

Figure 1: The genetic toggle switch. (a) A physical realization of the genetic toggle switch. (b) An STL specification for the genetic toggle switch.

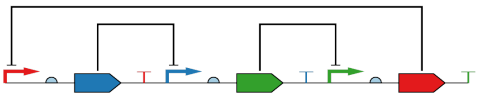
2. WORKFLOW

Given a library of modules and some experimental data, the methodology presented here can be used to characterize the modules with STLb specifications. These modules can be composed using a tree-based search and prune design space exploration technique to produce a genetic circuit that implements a desired performance specification given in STL.

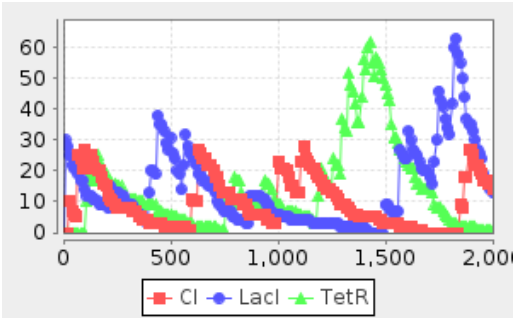
2.1 Characterization of Modules

Our method uses experimental characterization data along with the structural specification of the genetic module to construct a mathematical model representing its function. This mathematical model is simulated to produce traces representing possible behaviors of the system. These traces are passed through TLI to produce an STLb specification that captures the behavior of the module. However, TLI requires not only a set of traces for the desired behavior of a system but also requires a set of undesirable or unachievable traces. To address this problem, we have devised an automated method that is capable of producing this set by perturbing the set of traces produced during simulation.

For example, consider the repressilator module shown in Figure 2(a). Using experimental data, a mathematical model for this module can be constructed and simulated resulting in the simulation traces shown in Figure 2(b). This plot shows how LacI, TetR, and λCI oscillate due to the repression ring relationship they have with each other. Using TLI, the SLTb specification for the repressilator shown in



(a)



(b)

$$[G_{[0,400)}F_{[0,900)}(\text{LacI} > 15)] \wedge [G_{[0,1200)}F_{[0,900)}(\text{LacI} \leq 15)] \wedge$$
$$[\neg G_{[0,400)}F_{[0,700)}(\text{LacI} > 15)] \wedge [\neg G_{[0,1200)}F_{[0,700)}(\text{LacI} \leq 15)] \wedge$$
$$[G_{[0,400)}F_{[0,900)}(\text{TetR} > 15)] \wedge [G_{[0,1200)}F_{[0,900)}(\text{TetR} \leq 15)] \wedge$$
$$[\neg G_{[0,400)}F_{[0,700)}(\text{TetR} > 15)] \wedge [\neg G_{[0,1200)}F_{[0,700)}(\text{TetR} \leq 15)] \wedge$$
$$[G_{[0,400)}F_{[0,900)}(\lambda\text{CI} > 15)] \wedge [G_{[0,1200)}F_{[0,900)}(\lambda\text{CI} \leq 15)] \wedge$$
$$[\neg G_{[0,400)}F_{[0,700)}(\lambda\text{CI} > 15)] \wedge [\neg G_{[0,1200)}F_{[0,700)}(\lambda\text{CI} \leq 15)]$$

(c)

Figure 2: A diagram showing the steps involved in characterizing a module for the repressilator. (a) A genetic module representing the physical realization of the repressilator. (b) Time series data for the repressilator showing oscillations in the three signals. (c) The STL$\flat$ specification resulting from applying TLI to the data in (b).

Figure 2(c) is generated. This specification can be read as: each signal (LacI, TetR, and λCI) will always eventually rise above a value of 15 within 800 time units and will always eventually fall below a value of 15 within 800 time units.

2.2 Design Space Exploration

The genetic modules in our library can easily be composed using the STL$\flat$ specifications obtained from our characterization method. However, genetic modules may not behave as expected due to physical properties of genetic systems being hard to quantify [5]. Genetic components can also fail due to unanticipated nonmodularity that arises when genetic components are used in new genetic and environmental contexts [1]. To help catalog these scenarios, we have developed grammars for known failure modes, and after each iteration of testing assigned modules *in vivo*, the results of both successful and unsuccessful tests are used to fine-tune a set of rules we use to prune out undesirable or impossible combinations of modules. To name a few pruning rules, our grammars are able to eliminate module combinations that introduce cross-talk, that introduce secondary structures, and that are prone to undesirable homologous recombinations. They additionally consider different ways that modules can be combined and how these different combinations can lead to failure modes such as terminators on one strand of DNA affecting transcription on the other strand.

Figure 3 shows an example of a possible design space tree generated from a set of three modules. In this example, including m_1 and m_2 in the same design would lead to cross-talk as they both produce the same protein. Branches in

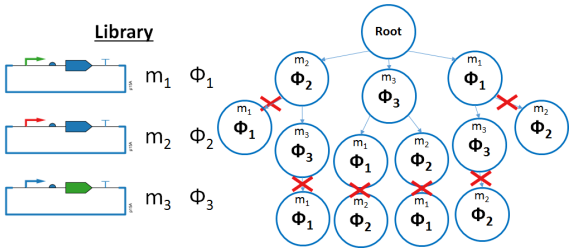


Figure 3: An example of a design space tree that could be generated from a library of three modules and their characterizations (m_1, ϕ_1 through m_3, ϕ_3). In this example, m_1 and m_2 produce the same protein, and therefore, they cannot be composed together in the same design due to problems with cross-talk. As such, branches that would contain both m_1 and m_2 are pruned indicated by the red X's on the design space tree.

the design space tree that include both of these modules are pruned and no more exploration is done on these paths. In the worst case, the pruning algorithm is unable to remove any branches; however, in this example, it is able to cut the design space in half by reducing a 15 node tree to 8 nodes.

3. DISCUSSION

The workflow presented here can be used to: 1) characterize genetic modules with STL$\flat$ specifications, and 2) efficiently explore the design space of an STL specification given a library of characterized modules. Once a set of designs are found, they can be compared against a desired specification using the distance metric found in [6], and the best design can be synthesized in the wet-lab. With this methodology, synthetic biologists will be able to convert physical modules that are currently being stored in a fridge in their laboratory into STL specifications. They will then be able to use these modules to automatically explore the design space of and construct more complex genetic circuit designs.

4. ACKNOWLEDGEMENTS

This work has been funded by the Office of Naval Research under Grant Nos. N00014-11-1-0725 and 014-001-0303-5 and the National Science Foundation under grant CPS Frontier 1446607.

5. REFERENCES

[1] J. A. Brophy et al. Principles of genetic circuit design. *Nature methods*, 11(5):508–520, 2014.

[2] Z. Kong et al. Temporal logic inference for classification and prediction from data. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, pages 273–282, 2014.

[3] O. Maler et al. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[4] A. A. Nielsen et al. Genetic circuit design automation. *Science*, 352(6281):aac7341, 2016.

[5] P. Vaidyanathan et al. A framework for genetic logic synthesis. *Proceedings of the IEEE*, 103(11):2196–2207, 2015.

[6] C.-I. Vasile et al. Compositional signal temporal logic with applications to synthetic biology. In *IEEE Conference on Decision and Control (CDC)*, 2016 (Submitted).