Engineering Living Systems: Closing the Loop in the Design-Build-Test Cycle using the Semantic Web

Bryan Bartley

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Herbert Sauro, Chair

James Bassingthwaighte

Daniel Cook

Program Authorized to Offer Degree:
Bioengineering

University of Washington

Abstract

Engineering Living Systems: Closing the Loop in the Design-Build-Test Cycle using the Semantic Web

Bryan Bartley

Chair of the Supervisory Committee:
Associate Professor Herbert Sauro
Bioengineering

The cell can do many wondrous things: produce valuable chemical compounds, assemble beautiful structures, move with purpose, and process information. For the synthetic biologist, these behaviors might be programmed at a genetic level in order to yield useful applications in health, energy, and environment. However, these applications are still in their infancy and synthetic biology remains a very empirical science due to our limited ability to reliably predict how genetic systems behave. Moreover, synthetic biologists struggle to build off the work of their predecessors, as much valuable knowledge and experience gained in the laboratory is not easily shared or reproducible.

To address these scientific challenges, I've developed computer-aided technologies, data exchange standards, and semantic web infrastructure that enable synthetic biologists to design, build, and test genetic systems on an industrial scale. In this dissertation, I highlight the fundamental engineering principles of standardization, modularity, and abstraction, and demonstrate how I am applying these principles in the fields of genetic engineering and synthetic biology.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY AND ABBREVIATIONS

ABSTRACTION HIERARCHY: The purpose of an abstraction hierarchy is to hide details and manage complexity. Abstraction hierarchies help engineers work at any one level of complexity without regard for the details that define other level

APPLICATION PROGRAMMING INTERFACE (API): A set of high-level, specialized functions for programming contained in a software library that apply to a specific kind of software application

BIOBRICK: DNA sequences which conform to a restriction-enzyme assembly standard which have been popularized through the iGEM synthetic biology competition

BIOLOGICAL PARTS: DNA sequences which are assumed to be interchangeable, ie, they can be re-used in new genetic contexts and expected to behave predictably. Just as interchangeable components have become essential for modern-day manufactured goods, interchangeable biological parts are expected to revolutionize genetic engineering

CHASSIS: In engineering a chassis is a frame on which mechanical or electronic components are attached. In synthetic biology a chassis is cell host for a synthetic DNA construct.

CLASS: A general category by which things are classified. In object-oriented programming, a class is a template for a data object

COMPUTER-AIDED DESIGN (CAD): The use of computer software to facilitate the generation, modification, and optimization of a part or a composition of parts for an engineering application

COMPUTER-AIDED TECHNOLOGIES (CAX): The use of computer software to aid in the design, analysis, or manufacture of products

DATA EXCHANGE STANDARD: A social agreement about how information should be organized in a common data format, so that users of information systems (e.g., databases, software) can share information with others using the same model.

DATA MODEL: A model that prescribes how data are structured or organized and how those data describe real world objects

DATA OBJECT: A container or record of data which can be manipulated through programmatic methods

FLUORESCENT REPORTER: A protein or other molecule with fluorescent properties which can be used to measure biological processes

GENETIC DESIGN AUTOMATION: A term for software tools that can be used to design synthetic DNA sequences. The term is derived from electronic design automation (EDA), software tools which are used to design integrated circuits

HOST: A host, as the term is used here, is a cell which harbors a plasmid.

INTERNATIONAL GENETICALLY ENGINEERED MACHINES (IGEM): A student team competition in synthetic biology. Teams attempt to solve problems by building genetically engineered biological systems with standard, interchangeable DNA parts.

MOLECULAR CLONING: A set of experimental methods in molecular biology that are used to assemble recombinant DNA molecules and to direct their replication within host organisms

OBJECT-ORIENTED PROGRAMMING: A programming paradigm based on the concept of data objects, which may contain data in the form of fields, often known as properties

ONTOLOGY: A system for classifying terms and concepts within a particular domain of knowledge. Computational ontologies enable automated reasoning by software tools.

PARTS REPOSITORIES: Online databases containining information about biological parts

PLASMID: A circular DNA molecule which replicates independently of the genome in a bacterial host. Plasmids are commonly used as substrates for genetic engineering because they are easily manipulated by molecular biology techniques.

PROVENANCE: Automated tracking and storage of electronic records for capturing successive stages of a worfklow

"THE REGISTRY": The Registry of Standard Biological Parts. An online resource containing data about thousands of standard biological parts contributed by students participating in the iGEM competition.

RESOURCE DESCRIPTION FRAMEWORK (RDF): A standard model for data exchange that makes it easier to integrate and query data on the world wide web.

RIBOSOME BINDING SITE (RBS): A region of DNA to which a ribosome binds. The affinity of the ribosome to this sequence affects the translational efficiency of a protein coding sequence.

SEMANTIC WEB: "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation" – Scientific American, May 2001

STANDARD ASSEMBLY: A molecular cloning protocol popularized by the iGEM competition which uses restriction enzymes and standardized plasmid sequences to enable the composition of complex DNA assemblages

SYNTHETIC BIOLOGY: An interdisciplinary field at the intersection of biology and engineering with the goal of creating minimal life or reprogramming the genetic code of organisms using DNA synthesis and DNA assembly

SYNTHETIC BIOLOGY OPEN LANGUAGE (SBOL): A data exchange standard for synthetic biology that allows networked software tools, parts repositories, and scientific instruments to share data

VARIANT: A DNA molecule which differs from its expected sequence because of mutations

WEB OF REGISTRIES: A networked infrastructure of biological parts repositories in individual labs or institutions which supports cross-institutional queries.

ACKNOWLEDGMENTS

I thank the following individuals for their contributions to the development of this dissertation.

- Michal Galdzicki (Arzeda Corp.) for providing valuable perspective and helping me realize my creative vision.
- Kiri Choi for assistance developing software applications discussed in this dissertation.
- Professors Daniel Cook and John Gennari for listening with an open ear.
- Professor Maitreya Dunham, thanks for the assistance with building chemostats.
- Professor Paul Wiggins, who taught me how to hack a microscope and increased my confidence as an experimentalist.
- Professor Jim Bassingthwaighte, an exemplary role model of a life dedicated to science.
- Professor Herbert Sauro, with whom I share a mutual fascination in many things, including science fiction, ancient Roman history, and the biochemical basis of life.
- And former members of the Sauro Systems & Synthetic Biology Lab: Wil Copeland, Kyung Kim, and Sean Sleight. It has been good working with you all.
- Thanks also to Bioengineering graduate advisors Dorian Varga and Peggy Sharp who have shepherded me through this lengthy and challenging endeavor.

- I'd also like to acknowledge the UW Veterans Benefits Office and Department of Veterans Affairs. Veterans benefits have been a crucial source of support in many different ways. Semper Fi.

DEDICATION

To my wife, Dr. Mandy Gulla

PREFACE

This dissertation includes excerpts from published papers, conference abstracts, manuscripts, and other sources of my writing. It starts by introducing the field of synthetic biology and principles of engineering as they apply to this emerging field of bioengineering. The chapters are organized along the theme of the design-build-test method for engineering problem solving and concludes by discussing how to "close the loop" in the design-build-test cycle using computer-aided technologies and data exchange standards.

Chapter 1

ENGINEERING LIVING SYSTEMS WITH SYNTHETIC BIOLOGY

1.1 Introduction to Synthetic Biology

Synthetic biology was originally founded more than 100 years ago as a biophysical discipline that sought explanations for the origins of life from chemical and physical first principles. Since then modern synthetic biology has been reinvented as an engineering discipline that exploits advances in DNA synthesis and assembly. These technologies make it possible to reprogram organisms at the genetic level, potentially with transformative applications. Section 2 of this chapter presents a survey of synthetic biology and perspective for the future of the field in the article *Engineering Living Systems from Biophysical Principles*.

1.1.1 Engineering Principles for Synthetic Biology: Standardization, Modularity, and Abstraction

The emergence of synthetic biology as a new field of engineering began in the year 2000 with publication of two seminal papers which demonstrated that cellular transcriptional networks could be engineered according to principles of electronic circuit design. The first genetic circuit to be built was the toggle switch [12], a two gene network that could switch between on and off states in response to different chemical inputs, a simple form of digital logic. The second genetic circuit demonstrated how oscillatory behavior could be engineered into transcription dynamics following the same principles as an electronic ring oscillator [9]. In

his landmark 2005 paper Foundations for Engineering Biology, Stanford bioengineer Drew Endy set forth the following definition of synthetic biology as an engineering discipline:

> Synthetic biology seeks to combine a broad expansion of biotechnology applications with...an emphasis on the development of foundational technologies that make the design and construction of engineered biological systems easier.

Endy goes on to describe some key engineering principles which have since become heavily emphasized in the synthetic biology literature, including standardization, modularity, and abstraction [10, 26, 22]. Standardization in other engineering disciplines allows interchangeable components to be easily combined to form larger systems. A closely related concept, modularity, implies that devices assembled from standardized components will function in a predictable manner through defined inputs and outputs. The principle of abstraction involves the organization of components into hierarchical compositions. Abstraction allows the engineer to simplify problems in terms of high-level, essential details while hiding low-level, nonessential details. Standardization, modularity, and abstraction are all important for scalable engineering of complex technologies. These principles are a recurring theme throughout this dissertation. In particular, this dissertation emphasizes my contributions to the development of a standard for synthetic biology called the Synthetic Biology Open Language.

In engineering practice, there are many different kinds of standards. Perhaps the most familiar kind of standard defines the physical or mechanical characteristics of components for easier assembly of composite systems. For example mechanical fasteners come in standard shapes and sizes which make it easier to assemble, re-use, or replace components in mechanical systems. Discussed in Ch. 3, the BioBrick Assembly standard is an analogous standard for synthetic biology which specifies how DNA molecules encoding novel biological function may be assembled from interchangeable DNA components. Another kind of en-

gineering standard, called a data exchange standard, deals with how information is shared through the use of computer-aided technologies and networked databases. A data exchange standard specifies how data is structured in files and transmitted across networks so that a recipient software application can decode the data and present it to a human user. The purpose of this kind of standard is to help engineers share knowledge, learn from each other, and build off the experience of their predecessors.

The Synthetic Biology Open Language (SBOL) is a data exchange standard founded to help synthetic biologists share information about biological systems and ultimately engineer them on an industrial scale [11]. The word "open" in the acronym reflects the guiding philosophy of its development community which believes that synthetic biology needs large-scale collaboration and open sharing of knowledge in order to reduce the cost, time, and risks of developing new synthetic biology applications. Membership of the SBOL development community consists of over 130 developers representing academia, industry, and government. Currently there are 4 open source software libraries, about 30 software tools, and at least 3 biological parts repositories (SynBioHub [21], the Inventory of Composable Elements [15], and the Virtual Parts Repository [6]) which communicate using the SBOL data exchange standard.)

During my career as a graduate student I have contributed in many ways to development of the SBOL standard. Ch. 2 presents open-source software I have developed that support the SBOL standard. As an elected editor for the SBOL community, I led many community activities such as facilitating discussions, administering votes and elections, drafting formal governance rules, and teaching tutorials and workshops. In 2015, I co-authored the version 2.0 specification document [2] (see Appendix A). This specification document describes a data model which formalizes how data must be structured in SBOL files.

SBOL version 1.1 [11] standardized the exchange of structural specifications of synthetic genetic designs, improving over formats such as GenBank [] by capturing the hierarchical organization of the genetic code. This includes the organization of DNA components into composite DNA components that represent transcriptional units, operons, and plasmids, as well as abstract designs with sequences that are not yet fully determined. Version 2.0 of SBOL standard [25] extended the original data model in order to describe the biochemical function encoded by genetic constructs. Biological function is described in terms of interactions between DNA, RNA, proteins, and small molecules. SBOL 2.0 represents these biochemical interactions and organizes them into modules with defined inputs and outputs which can be composed together with other modules to design a biological system.

1.1.2 The Engineering Life Cycle: Design, Build, Test

Despite all the recent technological advances in molecular biology and biochemistry, biochemical systems in even the simplest of organisms are usually not predictable with the mathematical precision that engineering requires. The sources of this biological unpredictability are many. Unlike solid-state electronics, the cellular environment is diffusive, leading to a high-degree of molecular interactions and mathematically complex, non-linear dynamics. Moreover, unlike an integrated circuit for computer memory, DNA is not simply a static storage bank for the genetic code. When DNA is recombined into new genetic contexts, it may form unexpected secondary structures with unpredictable effects on gene expression levels, therefore violating assumptions of interchangeability and modularity. Given this unpredictability, synthetic biology continues to be an empirical science as much as rational engineering, and the relatively cheap availability of synthetic DNA enables synthetic biologists to learn by trial-and-error.

Therefore synthetic biologists have adopted from other engineering fields a formalized framework for problem-solving known as the design, build, test cycle. The design, build, test cycle is the scientific method applied in the context of engineering. Stages of the cycle include designing an initial prototype, testing that prototype, analyzing its performance against specific metrics, learning what worked and what did not work, designing a new prototype based on what was learned, and completing the cycle again. Ideally each cycle generates new understanding that feeds back into new cycles as alternative approaches or reformulated problems.

The design, build, and test cycle is especially important in the context of synthetic biology because cellular systems are inherently complex and often unpredictable. Engineering living systems at the genome-scale requires that teams of engineers re-use and build off the work of their predecessors, just like other engineering fields such as software development or aviation (see Fig. 1.1. Such large-scale engineering efforts may require many iterations through design-build-test cycles. Therefore, much of the foundational work in synthetic biology currently revolves around the development of computer-aided technologies which enable synthetic biologists to automate stages of the design-build-test process.

In 2017 I authored an enhancement proposal to the SBOL specification which will enable synthetic biologists to describe design-build-test workflows and document these workflows using SBOL files. This specification will extend SBOL's utility to support large-scale, design-build-test engineering efforts across teams and institutions. This proposal is nearing formal adoption by the community and planned to be released in an upcoming issue of the Journal of Integrative Bioinformatics. This work is discussed in Ch. 5.

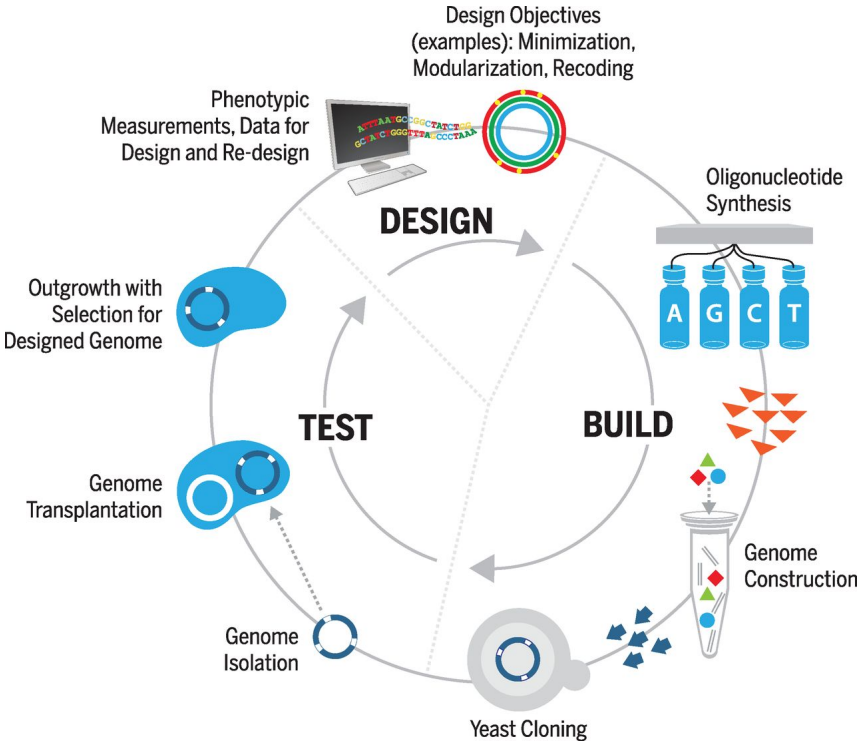


Figure 1.1: Many iterations through design-build-test cycles were employed to create the synthetic genome of *M. laboratorium* [19]

1.2 *Engineering Living Systems From Biophysical Principles*

Biophysical *Journal*
Biophysical Perspective

Synthetic Biology: Engineering Living Systems from Biophysical Principles

Bryan A. Bartley,[1] Kyung Kim,[1] J. Kyle Medley,[1] and Herbert M. Sauro[1,*]
[1]Department of Bioengineering, University of Washington, Seattle, Washington

ABSTRACT Synthetic biology was founded as a biophysical discipline that sought explanations for the origins of life from chemical and physical first principles. Modern synthetic biology has been reinvented as an engineering discipline to design new organisms as well as to better understand fundamental biological mechanisms. However, success is still largely limited to the laboratory and transformative applications of synthetic biology are still in their infancy. Here, we review six principles of living systems and how they compare and contrast with engineered systems. We cite specific examples from the synthetic biology literature that illustrate these principles and speculate on their implications for further study. To fully realize the promise of synthetic biology, we must be aware of life's unique properties.

The cell can do many wondrous things: produce valuable chemical compounds, assemble beautiful structures, move with purpose, and process information. For the synthetic biologist, who reengineers biology, knowledge about how to manipulate these processes might yield useful applications in health, energy, and the environment. The biophysicist, on the other hand, is perhaps more motivated by curiosity, a scientific search for the underlying physical laws that make living systems so different from non-living classical systems. Both are noble and worthy pursuits, and perhaps one cannot succeed without the other.

The philosophical and experimental foundations of synthetic biology were first laid in 1911 by French biophysicist Stéphane Leduc (1). Leduc explicitly argued that no boundary separates life from ordinary, physical phenomena, a point that he illustrated by growing chemical gardens, inorganic crystals exhibiting a striking and perhaps non-coincidental resemblance to living systems. Throughout the subsequent century, many biophysicists contributed to the discovery of the genetic code and establishment of the central dogma of molecular biology. The invention of recombinant DNA technology led the geneticist Wacław Szybalski to herald a new age of synthetic biology starting in the 1970s.

Today, driven partly by cost-efficient DNA synthesis, assembly, and sequencing, contemporary synthetic biology has been redefined largely as an engineering discipline rather than a biophysical discipline, emphasizing principles like standardization (2), modularity (3), digital logic (4), and mathematically predictable behavior (5). The synthetic biology toolkit includes a variety of programmable DNA (6), RNA (7), and protein regulatory elements (8). Many of these biochemical elements have been derived from natural biological elements but some are entirely synthetic. Currently, synthetic biology encompasses cell-free (9), bacterial, mammalian (10), and multicellular systems (11), but in principle it could apply to living systems at any scale, including ecological systems (12). Exciting applications include bioremediation of industrial waste, providing a sustainable source of food in future space missions, or creating model systems to study mechanistic theories of development. Evolution has tapped a mere fraction of the design space of biology, whereas the synthetic biologist is limited only by creativity and the laws of biophysics.

The extent to which engineering principles may apply to biology is still an open question. Although synthetic biology is often touted as a revolutionary technology for the "century of biology," real-world applications have found limited success (13). Biological systems are fundamentally different from most engineered systems. Here, we attempt to categorize and summarize some key principles that define biological systems: modularity, stochasticity, evolvability, robustness, and analog computation. This is in contrast to our own engineered systems, which behave deterministically and neither replicate nor evolve. Some features of living systems, such as modularity and control, are familiar principles to engineers, but require special consideration in the context of biology. Here, we discuss key phenomena of

Submitted July 13, 2016, and accepted for publication February 16, 2017.
*Correspondence: hsauro@u.washington.edu
Editor: Brian Salzberg.
http://dx.doi.org/10.1016/j.bpj.2017.02.013

living systems and speculate on their implications for synthetic biology. The unique challenges of engineering living systems might redefine engineering as much as they will redefine biology.

Modularity

Synthetic biology takes a bottom-up, forward-engineering perspective inspired partly by electronics and software design. From this perspective, modules should behave predictably when combined with other modules and connect with each other through defined inputs and outputs. The genome clearly demonstrates modular composition in the form of genes, operons, and horizontal transfer elements, but it also hides subtleties that confound the engineer. A module of DNA may hide cryptic regulatory elements or overlapping protein coding sequences. When recombined in new contexts, modules may form unexpected secondary structures with adjacent sequences. To modularize DNA, synthetic biologists have invented clever techniques for refactoring (14) and decoupling sequences (15) to prevent undesired interactions.

Biochemical systems have modules analogous to familiar electronic devices, such as switches, oscillators, filters, amplifiers, and feedback loops (16), which have in turn inspired biological modules for amplified biosensing (17), precisely timed genetic programs (18), and oscillators mimicking the circadian rhythm (19). Unfortunately, attempts to assemble modules into more complex systems have presented considerable challenges.

One reason that modular systems in synthetic biology are difficult to assemble is because their characteristics are not well understood or well matched. In the simplest case, signal ranges must match between connected units (4). In electronics, this is achieved by using a common voltage rail. Another issue, a phenomenon similar to impedance bridging in electronics, can occur when transcriptional modules are connected. As a result of extra load from a downstream module, the response of an upstream module may slow down, thus violating the assumption of modularity. In synthetic biology, this phenomenon is called retroactivity (20,21), and it may occur in transcriptional networks when transcription factors (TFs) drive too many target promoters (Fig. 1). Borrowing analogies from circuit engineering, Mishra et al. (3) successfully implemented load drivers using a buffer pool of TFs, which can be rapidly activated and deactivated by phosphorylation to compensate for retroactive dynamics. Op-amps in integrated circuits use similar control principles. Load drivers may enable more sophisticated systems for synthetic biology in future studies.

Systems biology provides a complementary perspective on modularity that may prove essential for building large-scale systems. The systems biology approach is generally characterized by a top-down, reverse-engineering approach, often using network-graph theory (22). One way to find modules in a network is to algorithmically search for simple, frequently occurring graph patterns called motifs. This has led to identification of several types of modules, including autoregulation, feedback and feedforward loops, and single-input modules (23). Another approach for finding modules in a network is to search for subnetworks with more connections internal to the module than between modules (24). Analyzed from this perspective, *Escherichia coli*'s protein-protein interaction network is organized into a central core with functionally distinguishable modules for translation, transcription, cell division, DNA synthesis, and DNA maintenance. Less prominent, peripheral modules are associated with this core. Modules are also defined by strong internal interactions (due to high binding-affinity for example), whereas connections between modules are typified by weaker interactions.

Some have argued that creation of large-scale biological networks will be difficult without a modular, engineering approach to synthetic biology (3). Taken to a logical extreme, a fully engineered cell might not function anything like cells observed in nature and indeed may challenge the very definition of life. On the other hand, synthetic biologists might need to consider how living systems are integrated (25). Circuits depend on host resources, creating a closed loop in which resource depletion feeds back and affects global transcription and translation processes that a circuit depends on. Although there is empirical evidence of common principles of resource allocation in several cell types (e.g., bacteria and yeast) (26), currently there is no theory that predicts the coupled behavior of a circuit with its host. Modular design of living systems may require careful consideration of constraints both at the local level in terms of well-matched inputs/outputs and at the global level in terms of network and resource integration. Synthetic and systems biology must work toward a unified perspective.

Stochasticity

One of Erwin Schrödinger's (27) most important legacies as a biophysicist is his proposition that life at the molecular level is dictated by statistical mechanics, which only appear orderly and determined when averaged over populations. Famously, he predicted that intracellular concentrations fluctuate due to Poisson processes. Though Schrödinger's fundamental insight about stochasticity was correct, biological "noise" turns out to be more complicated.

Today, synthetic biology combined with single-cell measurement techniques such as fluorescence microscopy and microfluidics allow us to observe the stochastic nature of biology (28,29). In some gene regulatory networks, the level of noise may be significant because copy numbers of some TFs may be small ($<$10 copy per cell for *E. coli* (30)). Thus, copy numbers in individual single cells can vary significantly due to random bursts of synthesis and degradation or to random segregation of molecules during cell division.

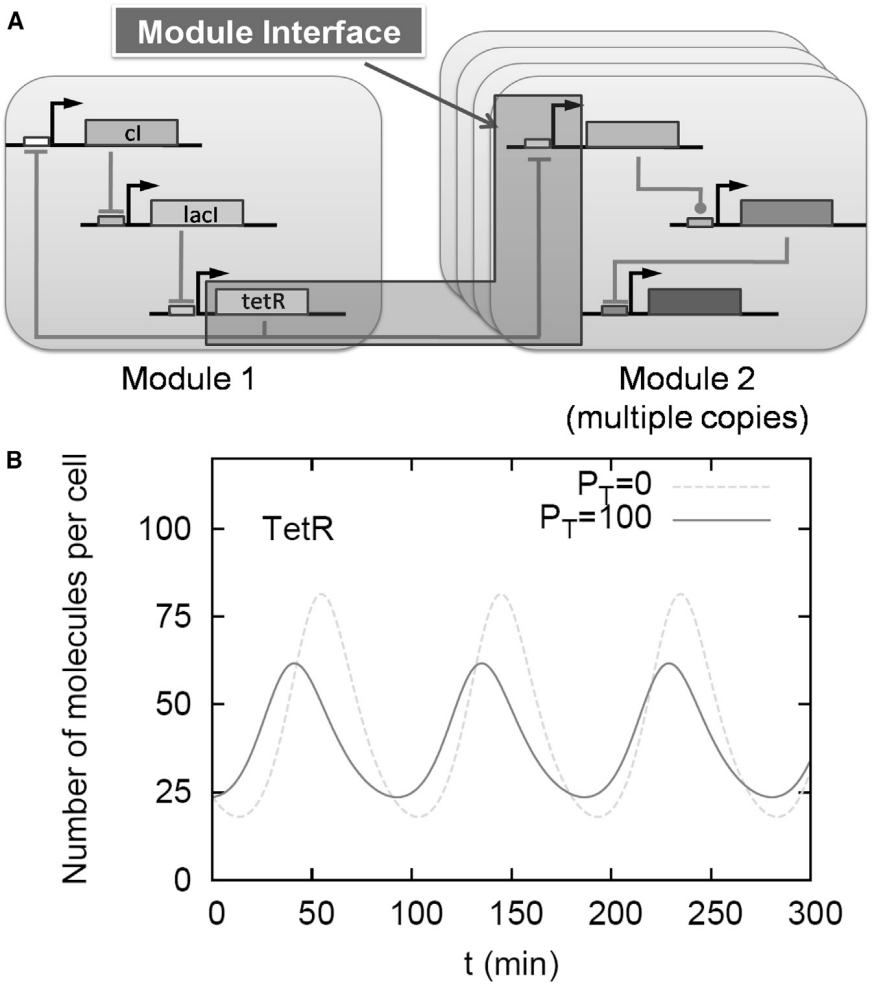


FIGURE 1 (*A*) A repressilator module (*Module 1*) regulates multiple copies of a downstream module (*Module 2*) by binding a transcription factor from Module 1 to operator sites in Module 2. (*B*) As the number of downstream binding sites increases, the amplitude of the oscillation changes. P_T is the number of operator binding sites. This effect is termed retroactivity (20). The figure is adapted from (21).

Describing noise quantitatively has been an important topic in synthetic biology (31). At the single-cell level, the stochastic dynamics of the gene regulatory networks are often mathematically described by the master equations and/or Langevin equations (32). Biochemical systems exhibit complex patterns of noise (33,34). Consequently, noise signals can be a result of nonlinear behavior in chemical reactions and, in addition, can propagate along connected pathways of proteins or small molecules. However, most noise studies have focused on linear systems and finding analytical solutions to noise propagation. These studies have provided a good mechanistic understanding of noise propagation, although the solutions can become quickly intractable as the number of state variables increases. To overcome this, systems-level numerical analysis of noise propagation, such as stochastic control analysis (35) and mass fluctuation kinetics (36), have been proposed, but more studies, focusing on numerical analysis at the systems level, need to be conducted.

In recent decades, it has been discovered that discrete phenotypes may also result from stochasticity in cellular dynamics, even in genetically identical populations (37). Stochastic decisions among different cellular phenotypes, as demonstrated in the simple model of Fig. 2, can result in specialized functional roles in various types of cells, including competence of soil bacteria (38), persistence of *E. coli* (39), and differentiation of stem cells (28). The probabilistic nature of these systems is a matter of great interest and curiosity, which Schrödinger would have enjoyed.

Studies of how stochasticity works in native systems has led synthetic biologists to begin applying stochasticity as a design principle. For example, *Bacillus subtilis* was genetically engineered to control expression noise in a critical factor that regulates DNA uptake competency, thus controlling the proportion of a population exhibiting this phenotype (40). *Saccharomyces cerevisiae* has also been genetically modified to investigate the effect of gene expression noise on cell survival under acute environmental stress, where larger expression noise of an antibiotic resistance gene was shown to help the cell survive (41). Circuits enabling precise control of noise decoupled from mean protein expression levels might be used to tune fitness and differentiation capabilities in cell populations (42).

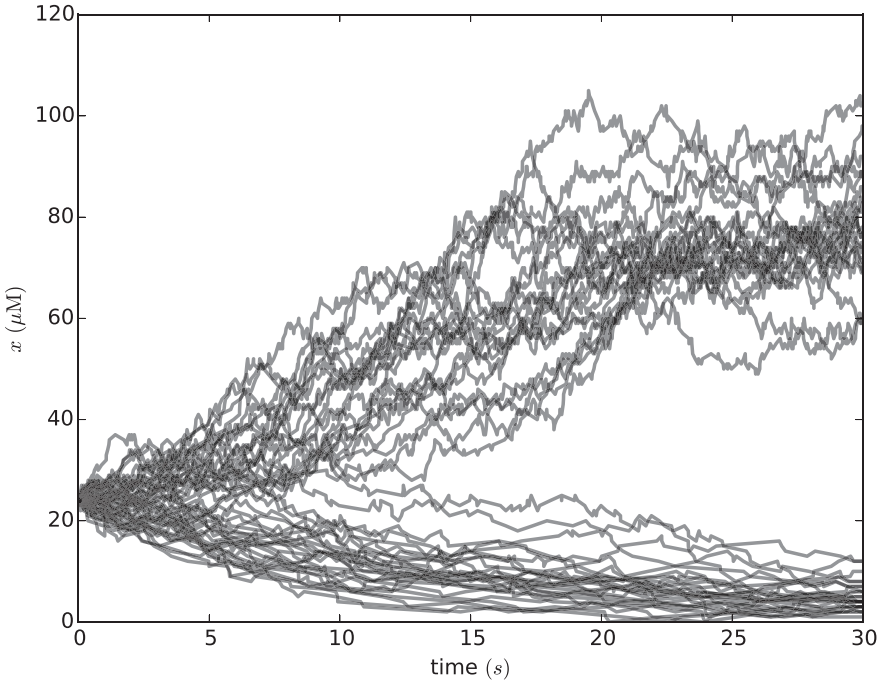


FIGURE 2 Example of bistability in a stochastic system (75). A stochastic simulation reveals two trajectories. A deterministic system would be confined to just one of these trajectories for a given parameterization, but stochasticity allows the system to exhibit both behaviors. Trajectories of 50 replicates are shown, each starting with the same initial conditions but with a different random seed. A brief description of this model and the Python code for reproducing the simulation are available in the Supporting Material.

Bulk measurements of cell populations can be affected by stochasticity at the single-cell level. Genetic circuits can be designed to exploit this effect. Noise-induced bistability, ultra-sensitive response, and linearization are some ways of exploiting gene expression noise that we and others have proposed (43). An intuitive way to understand this effect is to consider the nonlinear response of the stochastic signals based on Jensen's inequality (36,43). Further studies on how stochasticity affects gene regulation at the population level need to be conducted by systematically controlling gene expression noise (44).

Evolvability and robustness

Evolvable systems can reproduce, generate variation, and evolve through Darwinian selection (45). These properties may be artificially exploited to accelerate evolution and search for new biological traits, a methodology called directed evolution. In contrast, evolutionary processes may also be manipulated to *prevent* mutation and variation. Living systems must balance evolvability with evolutionary robustness to adapt to changing conditions without compromising viability.

Since living systems are still poorly understood in many ways, directed evolution remains as important to synthetic biology as rational engineering approaches. Directed evolution has been studied for decades, providing synthetic biologists with a diverse set of molecular components, including fluorescent reporter proteins, RNA aptamers that bind specific ligands (46), and enzymes with new catalytic properties (47). More recently, synthetic biologists are now employing directed evolution to fine-tune entire networks and pathways. Multiplex automated genome engineering (48) and trackable multiplex recombineering (49) are two such techniques that leverage multiple, simultaneous mutations across the genome to evolve new phenotypes. Potentially, directed evolution can be accelerated even further by exploiting mechanisms for natural transformation, whereby microorganisms acquire exogenous DNA from the environment (50) and mine new traits from metagenomic DNA sources (51).

Evolvability also refers to the evolutionary potential of populations, not only in terms of their present genetic variation, but also in terms of their ability to tune their own mutation rates and search out optimal genotype spaces where random mutations are more likely to generate viable phenotypes. The evolvability of gene and protein networks can be demonstrated by random recombination. *E. coli*'s transcription networks were re-wired by randomizing TFs and target promoters (52). Out of 628 reconfigured networks, ~80% exhibited no significant change in growth compared to controls, ~15% exhibited significantly altered growth phenotypes, and ~5% were nonviable. Peisajovich recombined protein signaling domains in yeast cells, producing 66 novel mating phenotypes (53). These studies illustrate that native networks can be varied in a way that leads to evolutionary innovation while simultaneously maintaining robustness to mutational failure. Synthetic biologists still have much to learn about how principles of evolvable systems might be factored into design strategies.

The electrical engineer typically need not worry about the physical dimensions of a capacitor suddenly changing, nor does the software engineer need worry about lines of code rewriting themselves, but for the synthetic biologist, random

mutation is a fact of life that makes engineering life especially challenging. Synthetic systems place a parasitic load on metabolic resources that often reduces the fitness of a host organism (54). Due to the fitness penalty associated with synthetic circuits, they can be rapidly selected out from fast-growing, well-mixed populations as mutants take over. Currently, circuits in *E. coli* behave reliably on the scale of days to weeks before loss-of-function mutants take over the population, a unit of time called evolutionary half-life (55). This has led synthetic biologists to search for strategies to make circuits more robust to failure. One way to stabilize circuits is by careful consideration of metabolic load. Minimizing protein expression or using strains optimized for protein expression is one important design rule for building gene networks (56). We found evidence for a critical threshold of protein expression that can be tolerated to maintain robust circuit life, but precise quantification is still an ongoing effort (57). In eukaryotes the effects of cellular burden and evolutionary failure have not been systematically evaluated to our knowledge. Cell-based therapies using synthetic biology will require extensive testing to establish safety and reliability in vivo.

Some have proposed that mutational failures can be delayed by modifying cells to slow down evolution (58). For example, the fidelity of DNA replication is an evolvable function that itself may be tuned by mutating genes for DNA polymerase. Fidelity might further be increased by combining with other improvements, such as upregulating DNA repair pathways. However, these strategies may only delay the inevitable. Models of evolutionary failure currently undergoing validation in our lab suggest that a 10-fold increase in fidelity increases the evolutionary half-life by only a few days in *E. coli* (Fig. 3). Slowing evolutionary processes may not stabilize synthetic circuits enough for long-term application.

A final strategy for stabilizing synthetic systems is to couple host survival to the synthetic circuit. Gonzalez (59) built a synthetic gene circuit for controlling stress response in which stress or response could be controlled independently by adjusting antibiotic and inducer concentrations,

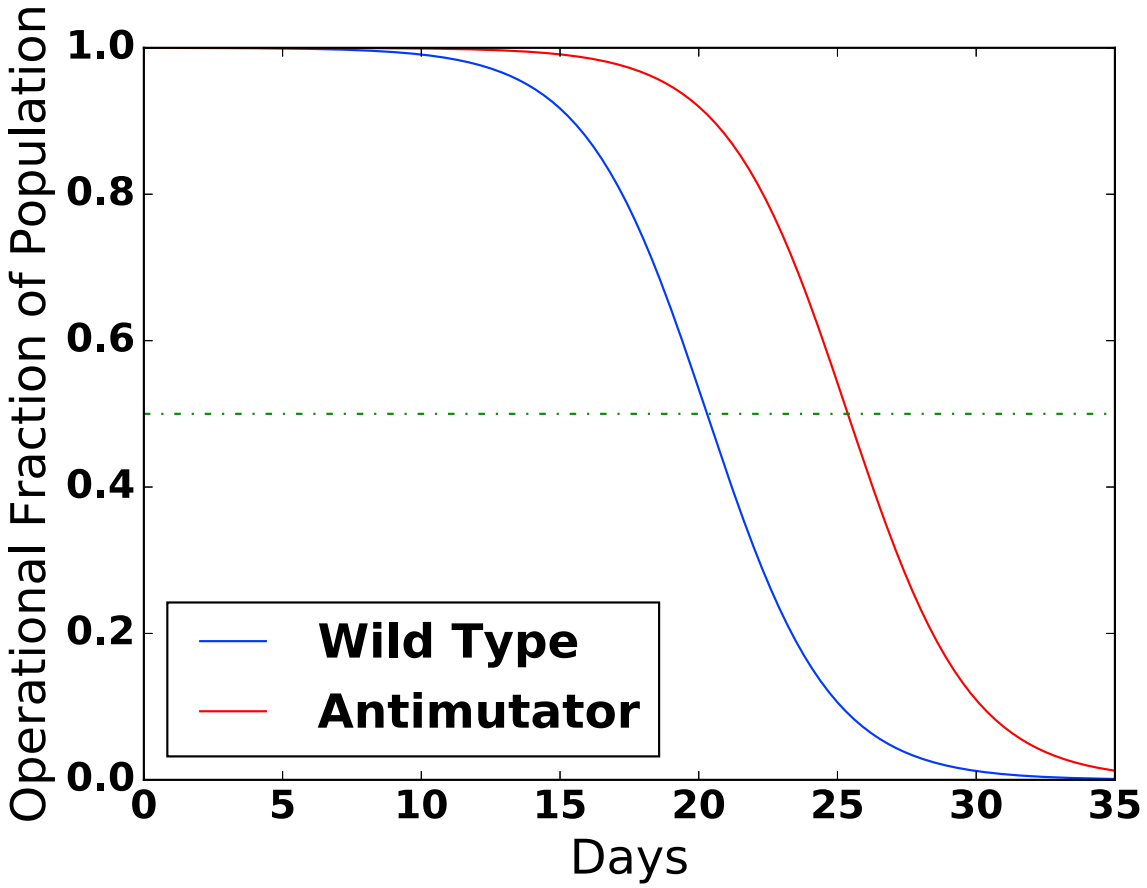


FIGURE 3 Simulations comparing evolutionary failure rates in wild-type versus antimutator strains. A population of *E. coli* cells carrying a synthetic circuit will give rise to loss-of-function mutants over time, thus decreasing the operational fraction of the population. The evolutionary half-life is the time it takes for the operational population to be reduced by half (55). In wild-type strains, DNA polymerase makes an error in ~1 of 10^9 bases, whereas antimutator polymerases make 10-fold fewer mistakes. The antimutator strain is predicted to prolong circuit life by ~5 days. The simulations assume a 20 min doubling time for a host with no operational burden and a 1% growth-rate burden for hosts carrying a circuit. A brief description of this model and the Python code for reproducing the simulation are available in the Supporting Material.

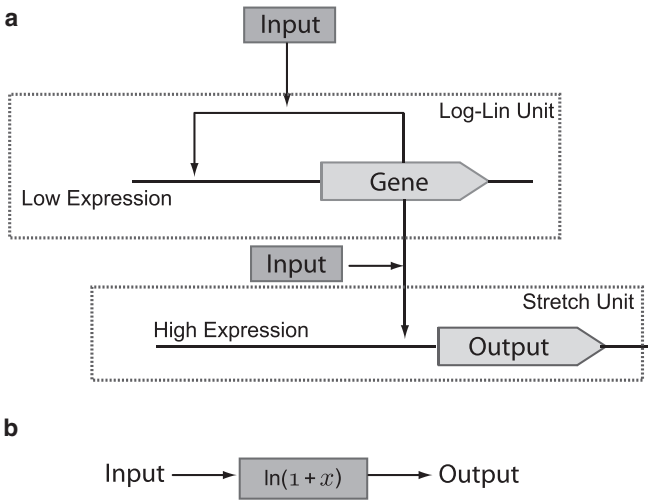


FIGURE 4 (*A*) Analog computation using two plasmids. The positive-feedback unit drives the circuit over a wide output range but narrow input range. When coupled to the shunt or stretch plasmid, the range of input concentrations to which the circuit responds can be extended many orders of magnitude. (*B*) The circuit output approximates the $\ln(1+x)$ function using a continuous range of protein concentrations.

respectively. Although mutations selectively disable a module if it confers only cost and no benefit, they can also activate a beneficial module that is otherwise dormant. In the field of metabolic engineering, computational algorithms have been devised that predict gene knockout targets or heterologous reaction pathways that force a host to produce a desired target while simultaneously improving growth rate, but these have been experimentally validated only to a limited degree (60).

Robustness of synthetic networks will be critical to many transformative applications of synthetic biology outside the laboratory. Developmental programs for synthetic tissues will need to be robustly designed to mitigate tumorigenic mutations. Ecological-scale applications using genetically modified populations may require robust mechanisms for mutualistic containment with native organisms (12). A grand theoretical challenge for synthetic biology is to discover how natural systems can be augmented with synthetic systems that produce the engineered behavior while simultaneously conferring an optimal advantage for the integrated whole system. Otherwise, our attempts to engineer applications with long-term reliability or economy of scale may inevitably be thwarted by evolution.

Analog computation

One of the most intriguing aspects of synthetic biology is the possibility of engineering computational and signal-processing capabilities into living cells. Computational tasks performed by biochemical networks include amplification, timing, signal integration, and information storage (61). Synthetic computation includes Boolean logic decisions, analog signal processing, neural-like architectures[1,3] and possibly entirely new ways to compute with data.

Digital computation has been the dominant approach to biological circuit design so far. The first to be built was the toggle switch (5), consisting of a two-gene network where each gene product inhibited the expression of the other. This is possibly the simplest circuit that can carry out a computation, in this case the capability of storing one bit of information. Other digital circuits have since been constructed, with recent work by Nielsen et al. (4) highlighting the level of complexity that digital biological circuits have now achieved.

Computing Boolean logic using TFs has presented challenges not found in solid-state transistors. First, biological circuitry operates in a diffusive environment instead of being hard wired, so parts must be sufficiently unique (orthogonal) to avoid molecular cross talk. By mining parts from prokaryotic genomes, Stanton et al. (62) isolated a library of 16 well-matched, orthogonal repressors for layered NOT and NOR gates. This library contains enough components for important biotechnological applications but not complex computation. Another significant issue is speed of operation. In electronics, we are accustomed to nanosecond transitions in logic states. Unfortunately, gate transitions in gene regulation operate on the minutes to hour scale. Still another complication with many TF gates to date is costly energy consumption and leakage. However, exciting advances in synthetic TFs are making these limits obsolete. Zinc-finger proteins, transcription activator-like effectors, and clustered regularly interspaced short palindromic repeats (CRISPR)-Cas9 are special families of DNA-binding proteins with highly programmable specificity and orthogonality (63) that are generating a wave of academic, media, and commercial interest. In addition, logic circuits based on RNA (7) instead of proteins may improve response times and decrease energy demand. These advances ensure that the computational power of living logic circuits will continue to grow.

In contrast to digital circuits, where the concentration of a species only represents two states, analog circuits represent ranges of values using continuous ranges of concentrations. In cases where energy, resources, and molecular components are limiting, analog circuits can enable more complex computations than can digital circuits. Using a relatively small set of components, Daniel et al. (64) were the first to engineer a synthetic circuit that performed analog computation by producing a log-linear response, $y = \log(x+1)$. They obtained this by coupling positive autoregulation of a TF to a shunt promoter on a high copy plasmid. The shunt promoter sequesters the TF, preventing the positive feedback from reaching saturation while simultaneously driving expression of the output over a wide dynamic range (Fig. 4). This combination yields the approximate extended log-linear relationship. Additionally, the researchers used a number of log-linear circuits to emulate arithmetic addition, division, and power laws.

Although some natural circuits, such as those controlling cellular differentiation, exhibit binary decision-making, many natural systems exhibit graded, analog responses to environmental inputs. For example, homeostatic regulation of blood-sugar concentration involves many variable hormonal inputs. Analog feedback control has long been explored as a means of engineering medical devices for regulating blood sugar in diabetics. The same approach could work for therapies based on synthetic biology.

Control

Biochemical control mechanisms govern important biological processes like adaptation, differentiation, and development and are found at every scale of life ranging from simple molecular switches to pheromonal regulation of populations. For synthetic biology to succeed, we will need to understand the principles of control in natural systems (65) to interface our own synthetic systems with them. Recently, Xu et al. (66) implemented bistable, oscillatory control over sink and source reactions in a biofuel synthesis pathway, allowing the host organism to regulate the tradeoff between product yield and growth. Controlled developmental programs are another important application of synthetic biology. Cao et al. (11) demonstrated morphogenetic control of scale-invariant patterns in engineered bacteria. Scale invariance allows developing organisms to maintain certain morphological features in proportion to the size of individuals. The Cao platform models this phenomenon by employing a circuit with negative feedback to generate a ring pattern that scales with colony size. Eventually, the ability to tune such developmental programs may lead to personalized synthetic organ replacements.

A form of logic control that has not been investigated much in synthetic biology is fuzzy, multi-input/multi-output (MIMO) control, despite the fact that such regulatory systems are common in transcription, translation, and cell-signaling networks (67). For example, the computational cortex of transcription networks appears to be a kind of MIMO gate called a dense overlapping regulon (23). This kind of control enables cells to make decisions based on the weighted sum of multiple inputs. Recently, Zadegan et al. (68) designed a fuzzy biosensor that responds to multiple microRNA inputs. Such a device could eventually be used to detect molecular profiles associated with different cancer types. Although MIMO control systems are usually less precise than layered, binary logic gates, they may be faster, more robust, and provide adaptive advantage in evolvable systems (69).

A defining feature of living systems is the presence of complex networks of feedback and feedforward control. Furthermore, some authors have proposed that feedback mechanisms are a necessary factor in the evolution of complex networks from prebiotic conditions (70). Positive feedback might enable primal replicators to maximize synthesis of biopolymers and gain selective advantage in prebiotic environments. In contrast, negative feedback may sustain primal replicators within bounds, preventing population collapse. Negative feedback may also play an important role in the evolution of hereditary memory in chemical networks by slowing the timescales of some reactions relative to others. These are fundamental biophysical hypotheses that can be tested in modern "chemical gardens" using synthetic biology.

An example of a modern analog of chemical gardens is synthetic RNA replicators that have reproduced and evolved under laboratory conditions (71). These systems could potentially serve as testbeds for demonstrating the evolution of even more complex reaction networks, perhaps consisting of RNA replicators and small RNA regulatory elements such as riboregulators (72). For contemporary synthetic biologists in the mold of Leduc, this is one of many untapped possibilities for investigating the fundamental principles of living systems.

Concluding remarks

Though synthetic biology was founded originally as a biophysical discipline, it has grown to encompass a large interdisciplinary community of scientists and engineers. It is a rapidly evolving field and there are many exciting areas of research that we have barely touched upon, including in vitro reconstituted systems (9), synthetic microbial ecosystems (73), and computational simulation of whole cells (74). In this article, we have focused mainly on unique biophysical properties of living systems that challenge our traditional approaches to engineering, especially when these properties are considered together. For example, modularity in evolvable systems may be unlike that in engineered systems. Synthetic circuits that exploit control of stochasticity and evolvability are also new frontiers for engineering. Many technical and societal challenges in this century of biology, whether related to medicine, environment, or renewable energy, will require a principled understanding of how synthetic and living systems can interoperate viably.

SUPPORTING MATERIAL

Supporting Material, one figure, and four tables are available at http://www.biophysj.org/biophysj/supplemental/S0006-3495(17)30221-7.

AUTHOR CONTRIBUTIONS

B.A.B. planned the overall structure and theme of the manuscript, wrote the sections on evolvability, robustness, and control, as well as parts of the introduction and conclusion, and edited the manuscript; K.K. wrote the section on stochastic systems and modularity; J.K.M. wrote part of the section on computation and part of the introduction and edited the manuscript; H.M.S. wrote the section on computation and part of the introduction and edited the manuscript. J.K.M. and B.A.B. performed simulations. Limited

space prevented us from citing all contributors to this important field, and we apologize for any published work we have been unable to cite.

ACKNOWLEDGMENTS

The authors acknowledge Mike Galdzicki (Arzeda Corp.) and Wilbert Copeland (CelGene) for useful discussions.

Research reported in this publication was partly supported by the National Institute of General Medical Sciences of the National Institutes of Health under award number R01GM081070, as well as by the National Science Foundation under grants MCB-1158573, MCB-1515280, EF-0827592, and DBI-1355909. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health or the National Science Foundation.

REFERENCES

1. Leduc, S. 1912. *La Biologie Synthétique*. A. Poinat, Paris.
2. Quinn, J. Y., R. S. Cox, 3rd, …, H. M. Sauro. 2015. Sbol visual: a graphical language for genetic designs. *PLoS Biol.* 13:e1002310.
3. Mishra, D., P. M. Rivera, …, R. Weiss. 2014. A load driver device for engineering modularity in biological networks. *Nat. Biotechnol.* 32:1268–1275.
4. Nielsen, A. A., B. S. Der, …, C. A. Voigt. 2016. Genetic circuit design automation. *Science.* 352:aac7341.
5. Gardner, T. S., C. R. Cantor, and J. J. Collins. 2000. Construction of a genetic toggle switch in *Escherichia coli*. *Nature.* 403:339–342.
6. Yang, S., S. C. Sleight, and H. M. Sauro. 2013. Rationally designed bidirectional promoter improves the evolutionary stability of synthetic genetic circuits. *Nucleic Acids Res.* 41:e33.
7. Chappell, J., K. E. Watters, …, J. B. Lucks. 2015. A renaissance in RNA synthetic biology: new mechanisms, applications and tools for the future. *Curr. Opin. Chem. Biol.* 28:47–56.
8. Khalil, A. S., T. K. Lu, …, J. J. Collins. 2012. A synthetic biology framework for programming eukaryotic transcription functions. *Cell.* 150:647–658.
9. Siegal-Gaskins, D., Z. A. Tuza, …, R. M. Murray. 2014. Gene circuit performance characterization and resource usage in a cell-free "breadboard". *ACS Synth. Biol.* 3:416–425.
10. Kis, Z., H. S. Pereira, …, R. Krams. 2015. Mammalian synthetic biology: emerging medical applications. *J. R. Soc. Interface.* 12:20141000.
11. Cao, Y., M. D. Ryser, …, L. You. 2016. Collective space-sensing coordinates pattern scaling in engineered bacteria. *Cell.* 165:620–630.
12. Solé, R. V., R. Montañez, and S. Duran-Nebreda. 2015. Synthetic circuit designs for earth terraformation. *Biol. Direct.* 10:37.
13. LaMonica, M. 2014. Why the promise of cheap fuel from super bugs fell short. *MIT Technol. Rev.*, Published online Feb. 5, 2014.
14. Temme, K., D. Zhao, and C. A. Voigt. 2012. Refactoring the nitrogen fixation gene cluster from *Klebsiella oxytoca*. *Proc. Natl. Acad. Sci. USA.* 109:7085–7090.
15. Mutalik, V. K., J. C. Guimaraes, …, D. Endy. 2013. Precise and reliable gene expression via standard transcription and translation initiation elements. *Nat. Methods.* 10:354–360.
16. Cameron, D. E., C. J. Bashor, and J. J. Collins. 2014. A brief history of synthetic biology. *Nat. Rev. Microbiol.* 12:381–390.
17. Nistala, G. J., K. Wu, …, K. D. Bhalerao. 2010. A modular positive feedback-based gene amplifier. *J. Biol. Eng.* 4:4.
18. Ellis, T., X. Wang, and J. J. Collins. 2009. Diversity-based, model-guided construction of synthetic gene networks with predicted functions. *Nat. Biotechnol.* 27:465–471.
19. Tigges, M., T. T. Marquez-Lago, …, M. Fussenegger. 2009. A tunable synthetic mammalian oscillator. *Nature.* 457:309–312.
20. Del Vecchio, D., A. J. Ninfa, and E. D. Sontag. 2008. Modular cell biology: retroactivity and insulation. *Mol. Syst. Biol.* 4:161.
21. Kim, K. H., and H. M. Sauro. 2011. Measuring retroactivity from noise in gene regulatory networks. *Biophys. J.* 100:1167–1177.
22. Newman, M. E. J. 2003. The structure and function of complex networks. *SIAM Rev.* 45:167–256.
23. Alon, U. 2007. Network motifs: theory and experimental approaches. *Nat. Rev. Genet.* 8:450–461.
24. Tamames, J., A. Moya, and A. Valencia. 2007. Modular organization in the reductive evolution of protein-protein interaction networks. *Genome Biol.* 8:R94.
25. Cardinale, S., and A. P. Arkin. 2012. Contextualizing context for synthetic biology—identifying causes of failure of synthetic biological systems. *Biotechnol. J.* 7:856–866.
26. Hui, S., J. M. Silverman, …, J. R. Williamson. 2015. Quantitative proteomic analysis reveals a simple strategy of global resource allocation in bacteria. *Mol. Syst. Biol.* 11:784.
27. Schrödinger, E. 1943. What is Life? The Physical Aspect of the Living Cell and Mind. Cambridge University Press, Cambridge, United Kingdom.
28. Kumar, R. M., P. Cahan, …, J. J. Collins. 2014. Deconstructing transcriptional heterogeneity in pluripotent stem cells. *Nature.* 516:56–61.
29. Singer, Z. S., J. Yong, …, M. B. Elowitz. 2014. Dynamic heterogeneity and DNA methylation in embryonic stem cells. *Mol. Cell.* 55:319–331.
30. Taniguchi, Y., P. J. Choi, …, X. S. Xie. 2010. Quantifying *E. coli* proteome and transcriptome with single-molecule sensitivity in single cells. *Science.* 329:533–538.
31. Balázsi, G., A. van Oudenaarden, and J. J. Collins. 2011. Cellular decision making and biological noise: from microbes to mammals. *Cell.* 144:910–925.
32. Gillespie, D. T. 2007. Stochastic simulation of chemical kinetics. *Annu. Rev. Phys. Chem.* 58:35–55.
33. McAdams, H. H., and A. Arkin. 1997. Stochastic mechanisms in gene expression. *Proc. Natl. Acad. Sci. USA.* 94:814–819.
34. Kim, K. H., K. Choi, …, H. M. Sauro. 2015. Controlling *E. coli* gene expression noise. *IEEE Trans. Biomed. Circuits Syst.* 9:497–504.
35. Kim, K. H., and H. M. Sauro. 2012. Adjusting phenotypes by noise control. *PLOS Comput. Biol.* 8:e1002344.
36. Gómez-Uribe, C. A., and G. C. Verghese. 2007. Mass fluctuation kinetics: capturing stochastic effects in systems of chemical reactions through coupled mean-variance computations. *J. Chem. Phys.* 126:024109.
37. Rao, C. V., D. M. Wolf, and A. P. Arkin. 2002. Control, exploitation and tolerance of intracellular noise. *Nature.* 420:231–237.
38. Maamar, H., and D. Dubnau. 2005. Bistability in the *Bacillus subtilis* K-state (competence) system requires a positive feedback loop. *Mol. Microbiol.* 56:615–624.
39. Maisonneuve, E., M. Castro-Camargo, and K. Gerdes. 2013. (p)ppGpp controls bacterial persistence by stochastic induction of toxin-antitoxin activity. *Cell.* 154:1140–1150.
40. Maamar, H., A. Raj, and D. Dubnau. 2007. Noise in gene expression determines cell fate in *Bacillus subtilis*. *Science.* 317:526–529.
41. Blake, W. J., G. Balázsi, …, J. J. Collins. 2006. Phenotypic consequences of promoter-mediated transcriptional noise. *Mol. Cell.* 24:853–865.
42. Murphy, K. F., R. M. Adams, …, J. J. Collins. 2010. Tuning and controlling gene expression noise in synthetic gene networks. *Nucleic Acids Res.* 38:2712–2726.
43. Kim, K. H., H. Qian, and H. M. Sauro. 2013. Nonlinear biochemical signal processing via noise propagation. *J. Chem. Phys.* 139:144108.
44. Wang, B., M. Barahona, and M. Buck. 2015. Amplification of small molecule-inducible gene expression via tuning of intracellular receptor densities. *Nucleic Acids Res.* 43:1955–1964.

45. Sniegowski, P. D., and H. A. Murphy. 2006. Evolvability. *Curr. Biol.* 16:R831–R834.

46. Li, Y. 2015. A quarter century of in vitro selection. *J. Mol. Evol.* 81:137–139.

47. Bornscheuer, U. T., G. W. Huisman, …, K. Robins. 2012. Engineering the third wave of biocatalysis. *Nature.* 485:185–194.

48. Wang, H. H., F. J. Isaacs, …, G. M. Church. 2009. Programming cells by multiplex genome engineering and accelerated evolution. *Nature.* 460:894–898.

49. Warner, J. R., P. J. Reeder, …, R. T. Gill. 2010. Rapid profiling of a microbial genome using mixtures of barcoded oligonucleotides. *Nat. Biotechnol.* 28:856–862.

50. Dalia, A. B., E. McDonough, and A. Camilli. 2014. Multiplex genome editing by natural transformation. *Proc. Natl. Acad. Sci. USA.* 111:8937–8942.

51. Sommer, M. O., G. M. Church, and G. Dantas. 2010. A functional metagenomic approach for expanding the synthetic biology toolbox for biomass conversion. *Mol. Syst. Biol.* 6:360.

52. Isalan, M., C. Lemerle, …, L. Serrano. 2008. Evolvability and hierarchy in rewired bacterial gene networks. *Nature.* 452:840–845.

53. Peisajovich, S. G., J. E. Garbarino, …, W. A. Lim. 2010. Rapid diversification of cell signaling phenotypes by modular domain recombination. *Science.* 328:368–372.

54. Collins, J. J., M. Maxon, …, H. Sauro. 2014. Synthetic biology: how best to build a cell. *Nature.* 509:115–157.

55. Sleight, S. C., B. A. Bartley, …, H. M. Sauro. 2010. Designing and engineering evolutionary robust genetic circuits. *J. Biol. Eng.* 4:12.

56. Ceroni, F., R. Algar, …, T. Ellis. 2015. Quantifying cellular capacity identifies gene expression designs with reduced burden. *Nat. Methods.* 12:415–418.

57. Sleight, S. C., and H. M. Sauro. 2013. Visualization of evolutionary stability dynamics and competitive fitness of *Escherichia coli* engineered with randomized multigene circuits. *ACS Synth. Biol.* 2:519–528.

58. Renda, B. A., M. J. Hammerling, and J. E. Barrick. 2014. Engineering reduced evolutionary potential for synthetic biology. *Mol. Biosyst.* 10:1668–1678.

59. González, C., J. C. Ray, …, G. Balázsi. 2015. Stress-response balance drives the evolution of a network module and its host genome. *Mol. Syst. Biol.* 11:827.

60. Fong, S. S., A. P. Burgard, …, B. O. Palsson. 2005. In silico design and adaptive evolution of *Escherichia coli* for production of lactic acid. *Biotechnol. Bioeng.* 91:643–648.

61. Sauro, H. M., and B. N. Kholodenko. 2004. Quantitative analysis of signaling networks. *Prog. Biophys. Mol. Biol.* 86:5–43.

62. Stanton, B. C., A. A. Nielsen, …, C. A. Voigt. 2014. Genomic mining of prokaryotic repressors for orthogonal logic gates. *Nat. Chem. Biol.* 10:99–105.

63. Didovyk, A., B. Borek, …, J. Hasty. 2016. Transcriptional regulation with CRISPR-Cas9: principles, advances, and applications. *Curr. Opin. Biotechnol.* 40:177–184.

64. Daniel, R., J. R. Rubens, …, T. K. Lu. 2013. Synthetic analog computation in living cells. *Nature.* 497:619–623.

65. Rao, C. V., H. M. Sauro, and A. P. Arkin. 2004. Putting the control in metabolic control analysis. Proc. 7th Dynamics and Control of Process Systems (DYCOPS) Conf.

66. Xu, P., L. Li, …, M. Koffas. 2014. Improving fatty acids production by engineering dynamic pathway regulation and metabolic control. *Proc. Natl. Acad. Sci. USA.* 111:11299–11304.

67. Duval, M., A. Simonetti, …, S. Marzi. 2015. Multiple ways to regulate translation initiation in bacteria: mechanisms, regulatory circuits, dynamics. *Biochimie.* 114:18–29.

68. Zadegan, R. M., M. D. Jepsen, …, J. Kjems. 2015. Construction of a fuzzy and Boolean logic gates based on DNA. *Small.* 11:1811–1817.

69. Valiant, L. G. 2009. Evolvability. *J. Assoc. Comput. Mach.* 56:3.

70. Nghe, P., W. Hordijk, …, N. Lehman. 2015. Prebiotic network evolution: six key parameters. *Mol. Biosyst.* 11:3206–3217.

71. Robertson, M. R., and G. F. Joyce. 2013. Highly efficient self-replicating RNA enzymes. *Chem. Biol.* 21 (2):238–245.

72. Rodrigo, G., T. E. Landrain, and A. Jaramillo. 2012. De novo automated design of small RNA circuits for engineering synthetic riboregulation in living cells. *Proc. Natl. Acad. Sci. USA.* 109:15271–15276.

73. Mee, M. T., J. J. Collins, …, H. H. Wang. 2014. Syntrophic exchange in synthetic microbial communities. *Proc. Natl. Acad. Sci. USA.* 111:E2149–E2156.

74. Karr, J. R., J. C. Sanghvi, …, M. W. Covert. 2012. A whole-cell computational model predicts phenotype from genotype. *Cell.* 150:389–401.

75. Medley, J. K., A. P. Goldberg, and J. R. Karr. 2016. Guidelines for reproducibly building and simulating systems biology models. *IEEE Trans. Biomed. Eng.* 63:2015–2020.

Chapter 2

DESIGN

2.1 Introduction

On the theme of design, this chapter consists of excerpts from two publications. Section 1 is a manuscript entitled *Genetic Design Automation with PySBOL* which is targeted for publication to the journal ACS Synthetic Biology. The article is formatted as a tutorial (peer-reviewed). Also related to the topic of design, Section 2 is a paper previously published in ACS Synthetic Biology called *DNAplotlib: Programmable Visualization of Genetic Designs and Associated Data*.

The first manuscript is about computer-aided design for synthetic biology (more specifically, genetic design automation). A significant part of my efforts as a graduate student has involved the development of open-source software libraries in the Python and C++ programming languages (PySBOL and LibSBOL, respectively). These libraries are used in the tutorial to walk the reader through a common task for biological design, *i.e.*,a genetic construct is assembled from standardized DNA parts. The libraries are based on the Synthetic Biology Open Language and support reading and writing data in the SBOL file format and exchanging data with online parts repositories. People inside and out of the SBOL community have long been asking for an easy-to-understand introduction that illustrates the purpose of the SBOL standard. Consequently, I have written this tutorial with a broad, interdisciplinary audience in mind. Although the article includes practical examples written in Python code, the discussion is intended to be accessible to biologists with no programming skills. This tutorial also targets undergraduates, who comprise a significant part of

the synthetic biology community through their participation in the massive intercollegiate competition known as iGEM.

The second publication presents a visualization tool called DNAplotlib (short for DNA plotting library) that can be used to illustrate parts-based genetic designs. This tool uses an early version of the PySBOL library I developed. It is able to read an SBOL file and then display a diagrammatic representation of the genetic construct contained in the SBOL file.

The SBOL standard is actually two standards in one–a data standard (described in detail in Chapters 1 and 5 of this dissertation) and a standardized set of schematic icons that represent genetic parts (analogous to the schematic icons used by electrical engineers in circuit diagrams). The SBOL Visual standard [23] is closely related to the SBOL Data standard as shown in Figure 2.1.

The importance of these standardized approaches has been recognized by the journal ACS Synthetic Biology which has adopted the use of SBOLv symbols in publication figures as well as recommending that authors include genetic design information in the form of SBOL files with submitted manuscripts [17].

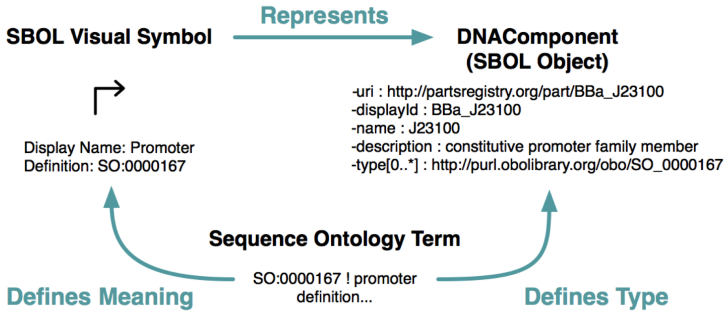


Figure 2.1: A genetic part in an SBOL file is described by a term taken from the Sequence Ontology [8], a standardized vocabulary for describing regions of a DNA sequence used by many software applications. Visualization tools which read the SBOL file know which SBOL Visual icon to render in the display based on this ontology term.

2.2 Genetic Design Automation

Genetic Design Automation with PySBOL

Bryan A. Bartley,* Kiri Choi, and Herbert M. Sauro

Bioengineering Department, University of Washington, Seattle, WA

E-mail: bartleyba@sbolstandard.org

Abstract

This tutorial is a beginner's introduction to genetic design automation using object-oriented programming in the Python scripting language. The tutorial presents a common use case that students in the International Genetically Engineered Machines (iGEM) competition encounter. A design for a feed-forward circuit is assembled using parts from the iGEM registry and the target sequence is compiled. Along the way, the foundational principles of synthetic biology are emphasized as they are put into practice through use of the library.

Introduction

This tutorial about computer-aided design for synthetic biology (more specifically, genetic design automation) serves several purposes. First, it hopes to provide an informative and educational demonstration of foundational principles of engineering, as defined by Endy,[1] as they pertain to synthetic biology, through the exercise of scientific scripting in the Python programming language. Secondly, this tutorial guides the user through designing the target sequence of a biological device using parts-based assembly, a synthetic biology workflow that students in the International Genetically Engineered Machines (igem.org) competition commonly encounter. This scenario includes stages such as checking out biological parts from the

iGEM registry, composing these into an abstraction hierarchy, coupling a circuit to a plasmid backbone, compiling a target sequence, and submitting the design back to a repository. Finally, this tutorial highlights the use of the Synthetic Biology Open Language (SBOL) data standard[2] as a foundation for open-source, collaborative, and scalable engineering efforts in synthetic biology.

The intended audience of this tutorial includes both undergraduates and professionals alike. This may include students who are developing software tools for the iGEM competition, biologists who are new to programming, and experienced developers who are developing advanced applications for synthetic biology. For readers who are not programmers, the code examples are intended to be illustrative and easily related to the narrative. Compared to other programming languages, Python is easy to learn with many scientific packages available, making it a good language for beginners. Readers who are more interested in synthetic biology rather than programming, may wish instead to try one of the many software tools now available which support SBOL.[3] However, the synthetic biologists who will contribute most to the field might be those who can devise programs to do exactly what they want, rather than relying on preconceived software.[4]

Throughout this tutorial we also emphasize the engineering principles which distinguish synthetic biology from related fields of biology, namely standardization, abstraction, and modularity. The PySBOL application programming interface (API) leverages these principles in order to help the user focus on the big picture of biological design by automating design task and tracking the implementation of this design through various stages in the laboratory. The library reads and writes SBOL files so that designs can be exchanged with other software tools and databases which support the file format.

Results and discussion

Getting Started

PySBOL provides an application programming interface (API) for synthetic biological design. Python is both a good introductory programming language for newcomers and versatile for scientific purposes. For purposes of this tutorial, users are expected to know some basics such as how to open a Python interpreter, import modules, and declare variables. For an introduction to Python programming we refer the reader to Google's Python Class[5] or Codecademy's Learn Python[6] interactive lessons. The pySBOL library is based on an approach to programming called object-oriented programming (see Methods for further introduction) which will be demonstrated in practice through the tutorial.

There are many options for installing the pySBOL library. Beginner programmers may want to install Tellurium, an integrated development environment (IDE) for biosystems analysis which includes pySBOL and other biosystems analysis packages. Tellurium offers a one-click install either as a Windows setup file[7] or Apple disk image.[8] Full installation instructions can be found at `https://github.com/sys-bio/tellurium#installation-instructions`.

PySBOL may also be installed via the Python Package Index, using the pip installer, a repository of open-source Python software that more experienced users may already be familiar with. Currently PyPI packages are supported for Mac, Windows, and Ubuntu. Additionally, packages for the Anaconda scientific computing environment are available as well. For further installation options and comprehensive instructions, visit `http://pysbol2.readthedocs.io/en/latest/installation.html`

Once the module is installed, it may be imported into the user's Python environment as follows. To confirm everything is working, check the version and / or run the included test scripts.

```
>>> from sbol import *
>>> sbol.__version__
'2.3.0'
>>> testSBOL()
```

It may take a couple of minutes for all tests to run. It will print out number of failed, skipped, and passed tests.

```
Total failed tests: 0
Total passed tests: 350
```

SBOL is a Semantic Web Standard

The SBOL standard is a data standard for the semantic web. The semantic web is a special part of the world wide web in which databases, software tools, and other networked information resources communicate with one another using computational ontologies. A computational ontology classifies concepts or things within a particular domain of knowledge in a machine-readable format that enables software to automate reasoning. Ontologies are used extensively by SBOL. For example, terms taken from the BioPAX ontology[9] and Sequence Ontology (SO)[10] indicate the molecular `type` (DNA, RNA, protein, etc.) and the functional `role` (promoter, CDS, etc.) of each `ComponentDefinition`, respectively. Each term in an ontology includes a natural language description associated with a standardized, machine-readable identifier. The SBOL standard unifies many different ontologies into a coherent model for biological design, combining perspectives from synthetic and systems biology and demonstrating interoperability between many different standards.[11]

When a new SBOL object is created, it must be assigned a uniform resource identifier (URI). The URI is a string which serves to uniquely identify and locate the data object in the semantic web. In fact, most readers are probably already familiar with a uniform resource locator (URL), which is a specific kind of URI used to look up webpages on the internet.

Although URIs may take many forms, those used to identify SBOL objects are typically composed of a scheme, a namespace, a local identifier, and a version, as shown in Figure 1.

Figure 1: SBOL objects are uniquely identified by a Uniform Resource Identifier (URI) that consists of a scheme, namespace, identifier, and version

PySBOL makes URI construction easy by allowing the user to configure the default scheme and namespace for new object creation, so the user need only specify a local identifier when calling a constructor. This identifier will be automatically appended to the default namespace. The user sets the default namespace by calling the `setHomespace` method. Although there are no perfect guarantees in life, the user can assure uniqueness of URIs in the semantic web by setting the default namespace to a network domain which they control, such as their lab or company domain. The following code snippet sets the homespace and constructs an object called "ffl" (for feed-forward loop). Printing the object displays its full URI. Later this object will be used to represent the design for a feed-forward loop.

```
>>> setHomespace('http://sys-bio.org')
>>> ffl = ComponentDefinition('ffl')
>>> print(ffl)
http://sys-bio.org/ComponentDefinition/ffl/1.0.0
```

"Drafting" a Document

In a previous era, engineers might sit at a drafting table and draft a design by hand. These days an engineer is more likely to use a computer. The engineer's drafting sheet in PySBOL is called a `Document`. The `Document` serves as a container, initially empty, for SBOL data

objects. A new instance of a `Document` may be created by calling a `Document` constructor. In the example below, the created instance is referenced by the variable `doc`. Optionally, a filename may be passed as an argument to the constructor, thus reading an input SBOL file, such as one of the supplementary files. A `Document` may contain only a few different types of SBOL objects, including `ComponentDefinitions, ModuleDefinitions, Sequences,` and `Models`. Before performing any advanced design tasks with PySBOL, these objects must first be added to a `Document`, either singly or as a list, as follows:

```
>>> doc = Document()
>>> doc.addComponentDefinition(ffl)
>>> for cd in doc.componentDefinitions:
        print(cd)
http://sys-bio.org/ffl/1.0.0
```

Only certain SBOL objects may be added directly to a Document, and these in particular are referred to as "top level" objects. The objects are stored inside a property that corresponds to the type of object. For example, to obtain a count of `ComponentDefinitions` use `len(doc.componentDefinitions)` (note that the property name starts with a lower-case character). To check which ones are contained in a `Document`, simply iterate through the corresponding property of the `Document` as shown in the code block above. The total count of all types of objects contained in a `Document` is determined using the `len(doc)` function.

Pulling Parts from Biological Parts Repositories

In today's modern technological society, a variety of interesting technologies can be assembled from off-the-shelf or interchangeable components, including cars, computers, and airplanes. Synthetic biology is inspired by a similar idea. Synthetic biologists aim to program new biological functions into organisms by assembling genetic code from standardized,

interchangeable biological parts. A biological part is an abstract representation of a DNA sequence that encodes or modulates some biological behavior inside a cell.

Standard biological parts with known functions are cataloged in biological parts repositories. Biological parts repositories serve as a common resource where synthetic biologists can go to obtain physical samples of DNA associated with descriptive data about the biological function that is encode. Perhaps the best example of a biological parts repository is the iGEM Registry of Standard Biological Parts (igem.org), which grows in contributions significantly because it serves as a collaborative platform for teams competing in the International Genetically Engineered Machine Competition (iGEM). Since 2008, the Registry has grown from approximately 2,000 parts to 32,000 parts, as determined by a recent programmatic search. Biological parts can be selected from the catalog and assembled in different combinations to construct a system or pathway in a "chassis" microbe such as *E. coli*.

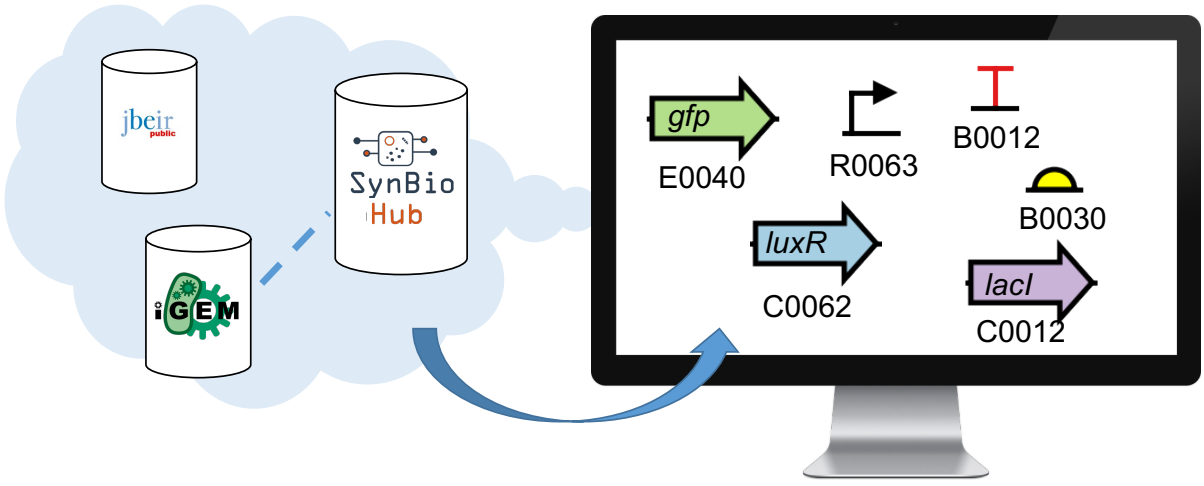


Figure 2: Biological parts repositories like SynBioHub and the Inventory of Composable Elements can be accessed programmatically using PySBOL. In this tutorial, parts are used to assemble a feed-forward circuit.

The PySBOL library puts an inventory of biological parts at the user's fingertips. PySBOL interfaces with two different types of biological parts repositories, those based on either SynBioHub[12] or the Inventory of Composable Elements (ICE)[13] database architectures.

Instances of these repositories may be installed and deployed by separate institutions or laboratories, creating a "web of registries". The pySBOL library communicates with these repositories using the SBOL data exchange standard. Although the iGEM parts registry does not support programmatic querying directly, a SynBioHub repository hosted at the University of Newcastle (https://synbiohub.org) hosts a mirror of the iGEM parts repository. These parts can be browsed online or searched programmatically using pySBOL.

In PySBOL a user interfaces with parts repositories through a `PartShop` object. The `PartShop` is initialized with the URL of the repository and, more specifically, the URL of the iGEM parts collection. In the following example, the user "pulls" parts from the repository and imports them into the user's local `Document`. These parts will be used to assemble the feed-forward circuit. Most of these parts are elementary genetic units, including promoters, ribosome binding sites (RBS), coding sequences (CDS), and transcriptional terminators. The promoters are regulated by either the LuxR activator or the LacI repressor proteins. However, part BBa_I751101 is a composite part consisting of multiple simple parts, one of which is a dual input promoter which may be regulated by both luxR and lacI. This dual input promoter will be a crucial part of the feed-forward loop and will later be extracted from the composite part.

```
>>> igem = PartShop('https://synbiohub.org/public/igem')
>>> igem.pull(['BBa_R0040', 'BBa_B0032', 'BBa_C0062', 'BBa_J61048',
 ↪  'BBa_R0063', 'BBa_B0034', 'BBa_C0012', 'BBa_B0010', 'BBa_B0012',
 ↪  'BBa_B0030', 'BBa_E0040', 'BBa_I751101'], doc)
```

When pulling from a SynBioHub `PartShop`, a common error is to mistype the URL (note the scheme `https`), which results in `RuntimeError: Couldn't resolve host name`. The user may also accidentally mistype the part designation (for example, omitting the "BBa_" prefix). If this occurs, the user may receive an error indicating that the part is not found.

A part may be retrieved from the `Document` and assigned to a local variable name in Python interpreter as follows:

```
>>> b0032 = doc.componentDefinitions['BBa_B0032']
>>> i751101 = doc.componentDefinitions['BBa_I751101']
```

This will be useful for inspecting a part's properties, as demonstrated in the next section.

Getting and Setting Object Properties

In the pySBOL library, a user manipulates property values using `get()`, and `set()` methods which generally follow a consistent pattern for different properties (see Methods for a description of the different kinds of properties). By default, all SBOL objects have some basic properties, like `name` and `description`. To view a property's value, call its `get()` method. For example, to view a component's `description`:

```
>>> print(b0032.description.get())
RBS.3 (medium) -- derivative of BBa_0030
```

The `description` contains a natural language description of the component, in this case alluding to the fact that the component is an RBS of medium strength. As mentioned in the previous section, software cannot easily interpret free text. Therefore the `roles` property contains machine-readable identifiers which unambiguously define the function of this part with ontology terms. Note that the property is plural, which indicates that the property may have more than one value. In this case, use the `getAll()` method to get a list of these roles.

```
>>> print(b0032.roles.getAll())
['http://identifiers.org/so/SO:0000139',
 ↪  'http://wiki.synbiohub.org/wiki/Terms/igem#partType/RBS']
```

These URIs may be used as web addresses in a browser which will take the user to a page with a human-readable definition of the term, though this is not always the case. Nor will every `ComponentDefinition` have `roles` defined, as suitable ontology terms may not be available.

Part BBa_I751101 is composed of multiple subparts, which are themselves represented by child objects of the parent `ComponentDefinition`. One of these subparts is a dual input promoter which can be positively regulated by the LuxR transcription factor and negatively regulated by LacI. The `ComponentDefinition.sequenceAnnotations` property contains a list of objects which describes features of the part's sequence. Its composition can be inspected as follows:

```
>>> for ann in i751101.sequenceAnnotations:
...     print(ann.displayId.get(), ann.name.get())
...
annotation1954818 R0063
annotation1954819 B0034
annotation1954820 C0062
annotation1954821 B0010
annotation1954822 B0012
annotation1954823 J54033
annotation1954824 SalI
annotation1954825 BamHI
annotation1954400 plux-lac hybrid promoter
annotation1954826 BglII
annotation1954828 J54025
annotation1954827 MluI
annotation1954829 B0030
annotation1954830 E0040
annotation1954831 B0010
annotation1954832 B0012
```

In order to use this promoter as a part for design and assembly, the `SequenceAnnotation` must be converted to a `ComponentDefinition` as follows:

```
>>> dual_input_promoter =
↪  i751101.sequenceAnnotations['annotation1954400'].extract()
```

Abstraction Hierarchies

Genetic "code" exhibits hierarchical organization, starting from the genome at a high level to progressively lower level units like operons, genes, and even lower level operators. In fact, hierarchical organization is a property of biological structures at every scale of life.

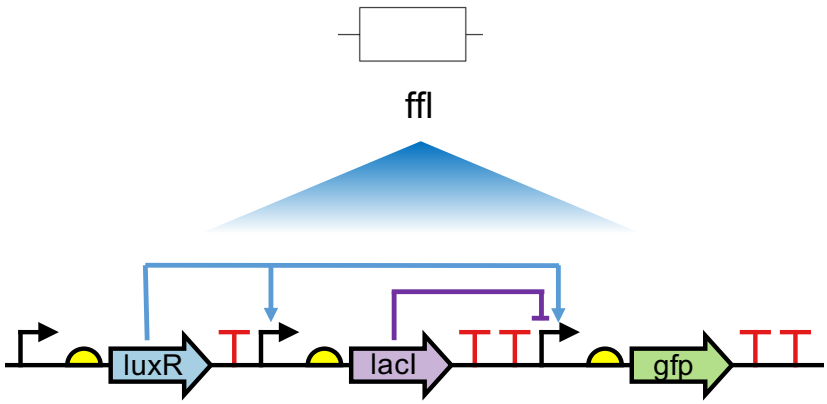
A goal of synthetic biology is to construct increasingly complex biological systems from hierarchies of biological parts. In contrast GenBank, which was originally developed to store results from DNA sequencing, contains a "flat" representation of genetic composition as a list of feature annotations. In the code below, a hierarchical representation of the feed-forward circuit is assembled:

```
>>> ffl.assemble(['BBa_R0040', 'BBa_B0032', 'BBa_C0062', 'BBa_J61048',
 ↪  'BBa_R0063', 'BBa_B0034', 'BBa_C0012', 'BBa_B0010', 'BBa_B0012',
 ↪  dual_input_promoter.displayId.get(), 'BBa_B0030', 'BBa_E0040',
 ↪  'BBa_B0010', 'BBa_B0012'])
```

In engineering terms, the `assemble` method creates an abstraction hierarchy. The purpose of an abstraction hierarchy is to hide details and manage complexity. Abstraction hierarchies help engineers work at one level of complexity without being distracted by details that define another level. This feed-forward circuit is at would describe as a biological device. Devices may then be assembled into more complex systems.

Biological parts also represent a distinct level of abstraction. When working at the "parts" level of abstraction, the engineer may ignore details about the exact DNA sequence of a part. Building designs from abstract biological parts is another reason why synthetic biologists might want to use SBOL rather than GenBank. Abstraction allows the synthetic biologist to design the high-level, functional characteristics of a biological system independently of its low-level, structural (or sequence) implementation. This enables a computer-aided design (CAD) approach similar to electronics, in which the functional design (*e.g.*,, schematic rep-

resentation) of an electronic circuit can be decoupled from the structural representation of electronic components, as shown in Figure 3.




```
ffl.assemble([ 'BBa_R0010', 'BBa_B0032', 'BBa_C0012', … ])
```

Figure 3: Abstraction hierarchies of biological parts can be assembled with the pySBOL library. In this tutorial, a feed-forward device is assemble from iGEM parts.

Sequence CAD

In the previous section, an abstract design for the feed-forward circuit was assembled into a `ComponentDefinition`. This kind of SBOL object represents a biological part as an abstract functional unit. By calling the `assemble` method, one arrives at an abstract design. By calling the `compile` method on a `Sequence`, one arrives at a realized design, with the full target DNA sequence specified. In this section, a target DNA sequence for the feed-forward circuit is compiled.

The abstraction hierarchy assembled in the previous section does not yet specify any kind of sequence to the individual parts. In order to arrange the parts into a primary sequence, the `linearize` method must be called. Afterwards, the primary structure of the feed-forward construct can be inspected as follows:

```
>>> ffl.linearize(['BBa_R0040', 'BBa_B0032', 'BBa_C0062', 'BBa_J61048',
 ↪  'BBa_R0063', 'BBa_B0034', 'BBa_C0012', 'BBa_B0010', 'BBa_B0012',
 ↪  dual_input_promoter.displayId.get(), 'BBa_B0030', 'BBa_E0040',
 ↪  'BBa_B0012'])
>>> for part in ffl.getPrimaryStructure():
...                 print(part.name.get())
...
p(tetR)
BBa_B0032
luxr
BBa_J61048
lux pL
BBa_B0034
lacI
BBa_B0010
BBa_B0012
plux-lac hybrid promoter
BBa_B0030
GFP
BBa_B0010
BBa_B0012
```

In order to generate the target sequence a `Sequence` object must be constructed and linked to the abstract design, as follows:

```
>>> ffl_seq = Sequence('ffl')
>>> doc.addSequence(ffl_seq)
>>> ffl.sequences.set(ffl_seq)
>>> target = ffl_seq.compile()
```

The `compile` method returns the target sequence as a string of nucleotides with a length of 3328 nucleotides, as shown in Figure 4.

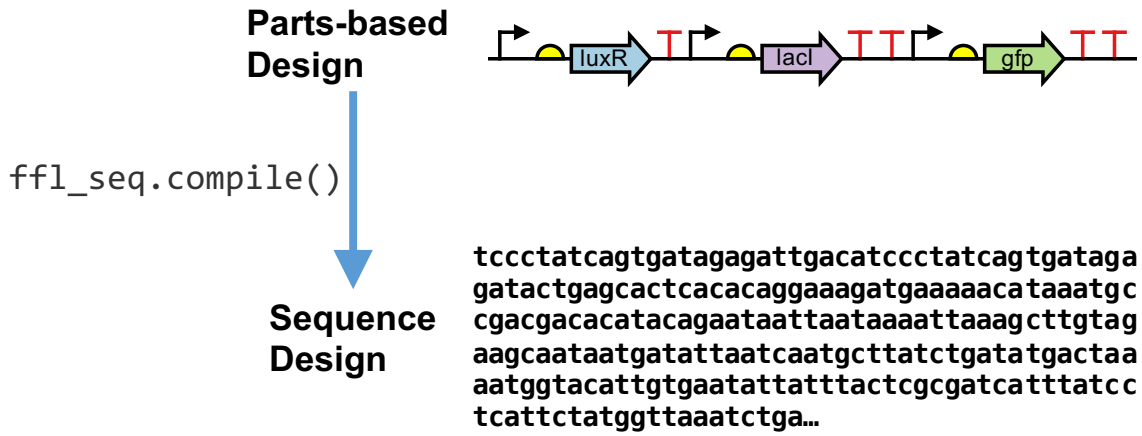


Figure 4: An abstract parts-based design is compiled into a target sequence.

Submitting Designs to a Repository

Submitting designs to a repository is important so that others, including the user, may re-use or reproduce existing designs. The synthetic biology journal ACS Synthetic Biology now encourages authors to submit SBOL data about their genetic designs to a repository like SynBioHub or JBEI-ICE. In order to submit to a `PartShop` remotely, the user must first vist the appropriate website and register. Once the user has established an account, they can then log in remotely using PySBOL.

Before submitting a design, the `Document` should be annotated and validated.

```
>>> doc.displayId.set('ffl')
>>> doc.name.set('feed-forward circuit')
>>> doc.description.set('An incoherent type 1 feed-forward loop')
>>> doc.validate()
'Valid. '
>>> login('bbartley@sys-bio.org', password)
>>> submit(doc)
'Successfully uploaded'
```

Methods

Object-oriented Programming

Python is an object-oriented programming language. In object-oriented programming, an object serves as a record or containers for data. The programmer uses methods (verbs) to perform actions on these data objects (nouns). Every object belongs to a class, which prescribes the form of data the object contains. (A helpful analogy might be to imagine a literal paper form with blank fields to be filled in.) A programmer creates an object by calling a special method called a constructor. Many objects of a class may be created and each may contain different data, but that data is consistently structured for all objects of the same class. Object-oriented programming can simplify and aid problem-solving for the programmer by organizing information into conceptual building blocks.

The Synthetic Biology Open Language specifies the data classes that are used by PySBOL and other software libraries which support the SBOL standard. In SBOL, objects which belong to classes such as `ComponentDefinitions` and `Sequences` represent the structural features of biological parts. Other SBOL objects, such as `ModuleDefinitions`, de-

scribe functional or behavioral aspects of a biological part. A programmer manipulates these data objects by calling methods using an imperative, noun-verb syntax, such as `component_definition.description.get()`.

Descriptive details about a data object are contained in properties. SBOL objects in particular may have several different types of properties according to the types of data they contain. Some properties contain text, which may need to conform to a specific format or in other cases may simply be a natural language description. For purposes of developing automated software tools, however, natural language descriptions are not very helpful. Therefore many properties of SBOL objects require a uniform resource identifier (URI) which can be easily and non-ambiguously interpreted by a software application. Some properties contain a link which points to another object. Finally, some data objects are themselves composed of other objects in what are called parent-child relationships. The child object is accessed through a property of the parent object. A composite object can be manipulated by the programmer as a single entity, and when the parent object is destroyed, so are its child objects.

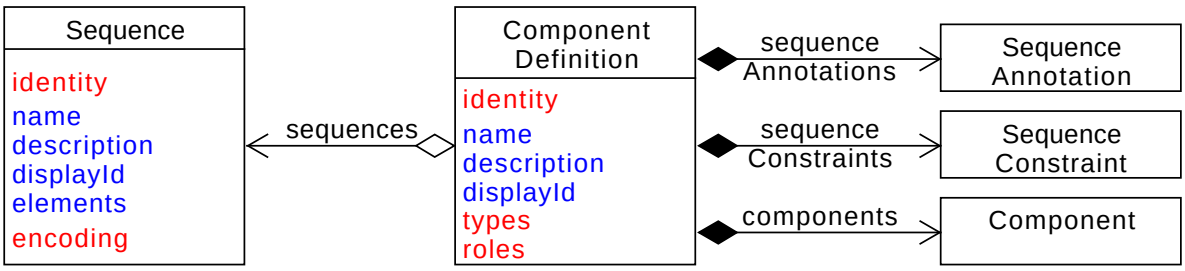


Figure 5: Diagram of ComponentDefinition and Sequence classes showing their properties. Text properties are in blue and properties which require a machine-readable URI are in red. Arrows with black diamonds indicate composite data structures, while arrows with white diamonds indicate a link.

In this tutorial, the objects used most often are `ComponentDefinitions` and `Sequences` as shown in Figure 5 The library API documentation and the official SBOL specification document describe all the data classes and their properties in detail.

Library Implementation

The PySBOL library is implemented in both Python 2.7 and 3.6 (32 and 64 bit). The library is distributed via the PyPI package service which allows users with an internet connection to install the package directly to their machine. Alternatively, installer applications are available. Comprehensive installation, documentation, and tutorials are available at https://pysbol2.readthedocs.io.

The PySBOL library was implemented starting from C++ source using code generation. A tool called the Simplified Wrapper Interface Generator (SWIG, swig.org) takes the C++ source code as an input and generates wrapper code in Python. Many scripting languages commonly used in the scientific community, including Python, Matlab, Javascript, and R, are based on C/C++. An advantage of implementing LibSBOL in C++ is that the library can now be used to generate libraries in many of these other common languages. There are several advantages of this code generation approach. First, it reduces the work required to implement a library in a new language. Second, the API is consistent across different languages so programmers may find it easier to switch between languages. Finally, the SBOL data produced by the libraries will be well-validated and predictable. All of these outcomes promote use and adoption of the SBOL data exchange standard by a wider community of programmers and software developers.

LibSBOL is an open-source, cross-platform C++ library that supports biological design tasks and customized engineering workflows for the synthetic biologist. LibSBOL is part of a growing family of open-source software libraries which communicate through the Synthetic Biology Open Language (SBOL) data exchange standard. LibSBOL, together with the Java[14] and Javascript libraries, libsbolj, and sboljs, support development efforts in a broad space of applications for synthetic biology, including databases and repositories,[12,13] web-based visualization tools,[15] and high-performance scientific computing applications.[16] As a community effort, these open source development efforts by both academic and in-

dustry contributors are intended to increase collaborative information-sharing in the highly interdisciplinary field that is synthetic biology.

Written in the C++ programming language, the libSBOL library provides high performance for scientific application developers. LibSBOL is written in standard ANSI C++11 and cross-compiles on Windows, Macintosh OSX, and Linux platforms with Clang, g++, MSVC++ compilers. Version 2.3 of libSBOL is currently in beta-release, providing an API to construct SBOL 2.2 designs and to read and write SBOL version 2.2 RDF/XML files. LibSBOL is free and open under the Apache 2.0 license. The source code is maintained under version control on Github and may be downloaded at https://github.com/SynBioDex/libSBOL. Pre-compiled binaries are distributed via binary installers or the Homebrew package service (https://brew.sh/). Comprehensive installation instructions, API documentation, and tutorial materials are available at http://synbiodex.github.io/libSBOL/.

Both PySBOL and LibSBOL support querying of online biological parts repositories. In addition the libraries interface with the online SBOL validator/converter tool hosted by the University of Utah.[17] This important feature ensures that users are creating and distributing valid SBOL files so that other software applications can read them.

The stability of the LibSBOL source code is enforced using the Travis continuous integration platform, which automatically builds and tests code changes as soon as the source code is updated. As another layer of validation, the library includes a test suite of SBOL test files which it shares in common with the Java and Javascript libraries. These built-in tests on these files verify that no data is lost during file exchange.

Acknowledgement

The authors thank Chris Myers, Zach Zundel, Meher Samineni, and Michael Zhang at the University of Utah for discussions and troubleshooting during the development of this library. Thanks also to the SBOL development community.

References

(1) Endy, D. *Nature* **2005**, *438*, 449–453.

(2) Bartley, B.; Beal, J.; Clancy, K.; Misirli, G.; Roehner, N.; Oberortner, E.; Pocock, M.; Bissell, M.; Madsen, C.; Nguyen, T. *Journal of Integrative Bioinformatics (JIB)* **2015**, *12*, 902–991.

(3) Group, S. D. Synthetic Biology Open Language Applications. http://sbolstandard.org/software/tools/, [Online; accessed 02-Dec-2017].

(4) Felton, M. J. *Modern Drug Discovery* **2002**, *5*, 28–33.

(5) Google, Python Class. https://developers.google.com/edu/python/, [Online; accessed 02-Dec-2017].

(6) Codecademy, Learn Python. https://www.codecademy.com/learn/learn-python, [Online; accessed 02-Dec-2017].

(7) Medley, K.; Choi, K. Tellurium. https://sourceforge.net/projects/pytellurium/files/Tellurium-1.3/1.3.6/Tellurium-1.3.6-Python-2.7-win32-portable-setup.exe/download, [Online; accessed 02-Dec-2017].

(8) Medley, K.; Choi, K. Tellurium. https://github.com/sys-bio/tellurium/releases/download/1.3.5-rc3/Tellurium-1.3.5-Spyder-2.3.8-OSX.dmg, [Online; accessed 02-Dec-2017].

(9) Demir, E.; Cary, M. P.; Paley, S.; Fukuda, K.; Lemer, C.; Vastrik, I.; Wu, G.; D'eustachio, P.; Schaefer, C.; Luciano, J. *Nature biotechnology* **2010**, *28*, 935–942.

(10) Eilbeck, K.; Lewis, S. E.; Mungall, C. J.; Yandell, M.; Stein, L.; Durbin, R.; Ashburner, M. *Genome biology* **2005**, *6*, R44.

(11) Roehner, N.; Beal, J.; Clancy, K.; Bartley, B.; Misirli, G.; Grunberg, R.; Oberortner, E.; Pocock, M.; Bissell, M.; Madsen, C. *ACS synthetic biology* **2016**, *5*, 498–506.

(12) Madsen, C.; McLaughlin, J. A.; Mısırlı, G.; Pocock, M.; Flanagan, K.; Hallinan, J.; Wipat, A. *ACS synthetic biology* **2016**, *5*, 487–497.

(13) Ham, T. S.; Dmytriv, Z.; Plahar, H.; Chen, J.; Hillson, N. J.; Keasling, J. D. *Nucleic acids research* **2012**, gks531.

(14) Zhang, Z.; Nguyen, T.; Roehner, N.; Misirli, G.; Pocock, M.; Oberortner, E.; Samineni, M.; Zundel, Z.; Beal, J.; Clancy, K. *IEEE Life Sciences Letters* **2015**, *1*, 34–37.

(15) McLaughlin, J. A.; Pocock, M.; Mısırlı, G.; Madsen, C.; Wipat, A. *ACS synthetic biology* **2016**, *5*, 874–876.

(16) Myers, C. J.; Barker, N.; Jones, K.; Kuwahara, H.; Madsen, C.; Nguyen, N.-P. D. *Bioinformatics* **2009**, *25*, 2848–2849.

(17) Zundel, Z.; Samineni, M.; Zhang, Z.; Myers, C. J. *ACS synthetic biology* **2017**, *6*, 1161–1168.

2.3 Programmable Visualization of Genetic Designs

ACS SyntheticBiology

Letter

pubs.acs.org/synthbio

DNAplotlib: Programmable Visualization of Genetic Designs and Associated Data

Bryan S. Der,[†,#] Emerson Glassey,[†,#] Bryan A. Bartley,[‡] Casper Enghuus,[§] Daniel B. Goodman,[∥,⊥] D. Benjamin Gordon,[†] Christopher A. Voigt,[†] and Thomas E. Gorochowski*[,†]

[†]Synthetic Biology Center, Department of Biological Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, United States

[‡]Department of Bioengineering, University of Washington, Seattle, Washington 98195, United States

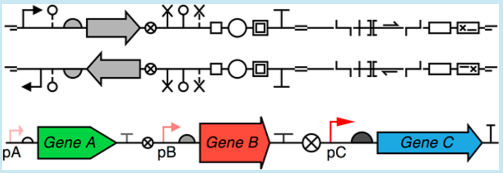
[§]MIT Microbiology Program, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, United States

[∥]Department of Genetics, Harvard Medical School, Boston, Massachusetts 02115, United States

[⊥]Wyss Institute for Biologically Inspired Engineering, Harvard Medical School, Boston, Massachusetts 02115, United States

Supporting Information

ABSTRACT: DNAplotlib (www.dnaplotlib.org) is a computational toolkit for the programmable visualization of highly customizable, standards-compliant genetic designs. Functions are provided to aid with both visualization tasks and to extract and overlay associated experimental data. High-quality output is produced in the form of vector-based PDFs, rasterized images, and animated movies. All aspects of the rendering process can be easily customized or extended by the user to cover new forms of genetic part or regulation. DNAplotlib supports improved communication of genetic design information and offers new avenues for static, interactive and dynamic visualizations that map and explore the links between the structure and function of genetic parts, devices and systems; including metabolic pathways and genetic circuits. DNAplotlib is cross-platform software developed using Python.



KEYWORDS: *visualization, standardization, SBOLv, biodesign automation, synthetic biology*

Engineering disciplines rely on standardized pictorial representations of parts and their interconnections to create schematics that clearly communicate how they are pieced together and to enable the reliable construction of large complex systems. In bioengineering, DNA sequences are often synthesized to create genetic constructs that probe or perturb the natural function of a cell or implement novel capabilities. Unlike more established engineering disciplines, the way that a genetic design is visually represented can vary significantly between laboratories and across different areas of the field. This leads to ambiguities that hinder data exchange, understanding, and the effective reuse of this research.

The Synthetic Biology Open Language Visual[1] (SBOLv) initiative was started to help alleviate this problem, defining a set of agreed symbols for commonly used genetic elements. In addition, other schemes such as the Systems Biology Graphical Notation[2] (SBGN) have been developed to more broadly standardize the graphical notation used to describe biological processes. The importance of these standardized approaches has also been recognized by publishers, with a major synthetic biology journal (ACS Synthetic Biology) adopting the use of SBOLv symbols when presenting genetic design information.[3]

Although these initiatives will help accelerate adoption of these standards, they rely on the availability of supporting tools to enable the production of compliant diagrams. Some tools do exist to generate SBOLv visualizations from genetic design information, either through graphical point-and-click interfaces (*e.g.*, VectorNTI,[4] TinkerCell,[5] GenoCAD,[6] DeviceEditor[7] and SBOL Designer) or text-based inputs (*e.g.*, Pigeon[8] and VisBOL[9]). These are effective for small numbers of constructs, but lack the ability to easily process large design libraries, are difficult to integrate into existing analyses, and offer only limited customization of the visualizations produced. An ability to tune how each genetic element is displayed (*e.g.*, the size, shape and color) based on its characterized performance[10] would enable clearer communication of key design features and enable an effective comparison of multiple designs. No tools currently support this capability.

To address these limitations, we developed DNAplotlib, a computational toolkit that enables the highly customizable visualization of standardized genetic designs in a programmable way (Figure 1). DNAplotlib is written in the Python programming language and makes extensive use of the matplotlib[11] graphics library to produce high-quality output in the form of vector-based PDFs, rasterized images and animated movies

Special Issue: IWBDA 2016

Received: September 10, 2016
Published: October 17, 2016

DOI: 10.1021/acssynbio.6b00252
ACS Synth. Biol. 2017, 6, 1115−1119

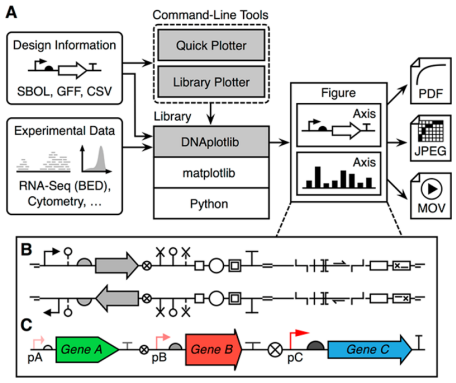


Figure 1. Overview of DNAplotlib. (A) Schematic of the visualization pipeline and supporting libraries. Genetic designs are provided as SBOL,[14] GFF or CSV files, or created through direct calls to the DNAplotlib library. Associated experimental data relating to individual parts or entire designs (*e.g.*, RNA-seq transcription profiles in the BED format[24]) can also be provided to influence properties of the visualization. Shaded boxes denote elements included as part of the library. (B) DNAplotlib supports the complete set of standardized SBOLv[1] parts in both forward and reverse orientations. (C) The size, color, shape and labeling of all genetic parts can be customized to convey associated part information, *e.g.*, promoter strength.

(Figure 2; Movie S1). Python was chosen due to its broad and growing use in the analysis of biological data, its ability to effectively "glue" together many different computational tools to create complex workflows,[12,13] and for being highly portable across all major operating systems. To simplify the process of generating visualizations in code, numerous helper functions are included to load genetic design information from files in the Synthetic Biology Open Language (SBOL),[14,15] General Feature Format (GFF), and Comma Separated Values (CSV) formats (Supporting Information). The full set of SBOLv parts[1] are available (Figure 1B) and users can easily extend existing functionality to cover new types of part or regulation and apply their own visual annotations at precise points within a genetic construct (Figures 2 and 3). Built-in parts also offer a broad range of customization options, enabling the visual communication of other characteristics such as measured performance (*e.g.*, promoter or terminator strength) that go beyond the part type alone (Figure 1C).

At the core of DNAplotlib is the main rendering pipeline (Figure 4A). This is implemented within the DNARenderer object and executed using the renderDNA(...) function. To tailor the rendering of each type of part or regulation arc, rendering functions are provided to the DNARenderer as dictionaries (part_renderers and reg_renderers in Figure 2A) with the part or regulation type mapping to the associated rendering function. Standard built-in functions can be chosen that cover the full range of SBOLv parts (Figure 1B; Supporting Information), or the user can specify their own, which may include new types of part or regulation not currently available (*e.g.*, recombinases, see Figure 3). To create a visualization, designs are provided in the form of a list where each element is a dictionary defining the part at that position in the design as well as other design information such as orientation, length and styling options. The use of a dictionary data type to store this information allows for varying numbers and types of option to be easily accommodated. Visualizations are automatically generated by scanning this list and, for each element, calling the associated function for the part type encountered. If an unrecognized part type or attribute is met, this element is ignored to ensure that multiple rendering functions with differing levels of functionality do not break the entire pipeline. Regulation is handled in a similar way with start and end points provided, in addition to styling options. Regulation links are automatically routed to minimize overlapping regions. All rendering is performed using a matplotlib axis object, which enables genetic designs to be directly incorporated into existing plotting routines (*e.g.*, bar charts and scatter plots, see Figure 2).

All built-in part and regulation renderers can be customized through the use of predefined options (Figures 4B and 5). To customize part appearance, a dictionary called opts is added to the specific part or regulation element that needs customizing. The opts dictionary defines a mapping between a customization option and the value it should take. These options are automatically sent to the relevant rendering function by the DNARenderer object when the part or regulation arc is drawn. Options not used by the renderer are ignored. A full table of all options, their format, and the elements that are compatible are shown in Figure 5.

The programmable nature of DNAplotlib opens up many unique capabilities not possible with other tools. For example, genetic circuits are composed of many parts whose regulation leads to numerous internal states of gene expression. Illustrating these and the strengths of regulation present is a challenge as the complexity of a circuit grows. Similarly, the construction of large variant libraries that explore a potential genetic design space has become commonplace as DNA synthesis costs have fallen and assembly methods have improved.[16−19] Visualizing a large number of internal circuit states or design variants manually is both time-consuming and error-prone. However, a simple computer program can be written to rapidly and accurately enumerate these, and DNAplotlib used to automate the visualization of key design features, part attributes and the internal regulatory links that are present (Figure 2A,B). This is made possible by direct programmable access, which also allows for tight integration into existing analysis workflows with minimal effort. DNAplotlib is already used within the genetic circuit design automation program Cello[20] to visualize candidate designs. Furthermore, the ability to automate the generation of large numbers of visualizations containing small variations in regulation opens up new opportunities to produce animated visualizations that convey the dynamics of a system (Movie S1). This is useful for genetic devices such as oscillators[21] whose output naturally varies in time, having no single steady state.

Another feature differentiating DNAplotlib is the inclusion of "trace-based" symbols for promoters, ribosome binding sites (RBSs), genes, terminators, and user-defined regions that goes beyond the SBOLv standard. These symbols take inspiration from genome browsers,[22,23] aiming to display not only functional information about the part type encoded at a particular point in a design, but also to provide a physically accurate representation of its position and extent within the DNA. This allows for a direct comparison to experimental data (Figure 2C) or other designs (Figure 2D) at a base pair resolution. This is achieved by either extending the length of gene and user-defined element symbols, or having filled rectangular regions cover the backbone of the construct for the length of a promoter, RBS or terminator, and using standard symbols extending from these to denote the part type (see Figure 2C,D). With sequencing seeing increased use across biology and allowing for the collection of large amounts of data at this

DOI: 10.1021/acssynbio.6b00252
ACS Synth. Biol. 2017, 6, 1115−1119

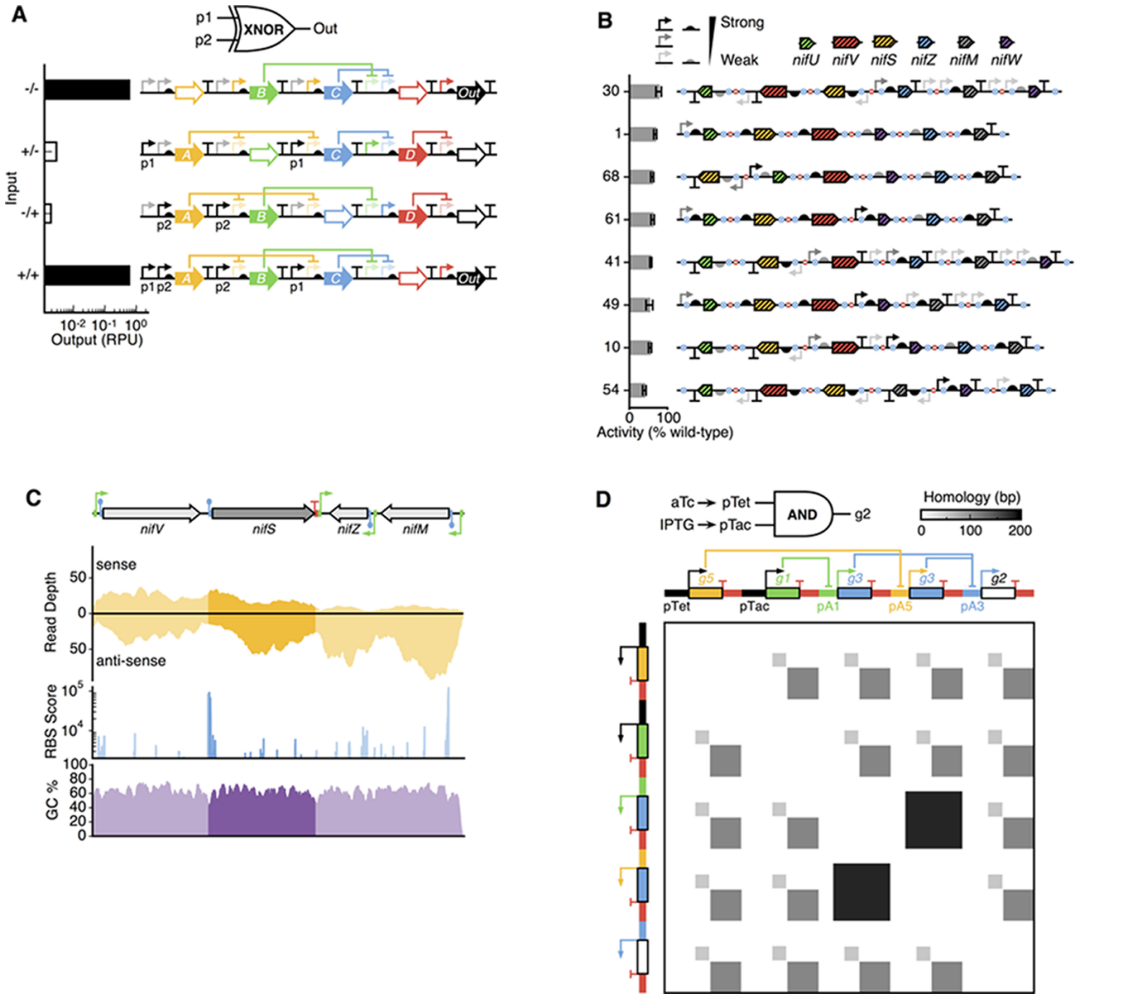


Figure 2. Examples of DNAplotlib visualizations. All are available from the project Web site. (A) Bar graph shows predicted output in relative promoter units (RPUs) for a hypothetical repressor-based XNOR genetic device designed by Cello.[20] The corresponding construct and expected state of all promoters and genes for each combination of inputs is shown to the right. Input promoters are active if black and labeled, repressible promoters are active if strongly colored, and genes are expressed if filled. Regulatory links that are present for a given set of inputs are included. (B) Selection of refactored *nif USVWZM* gene cluster designs.[25] Bar graphs represent the relative activity of the encoded synthetic nitrogen fixation pathway with error bars showing ±1 standard deviation. Numbers correspond to the variant number in the original study. In the genetic designs, promoter and RBS strengths are shown ranging from strong (black) to weak (light gray), spacer elements are blue and cloning scars are red. (C) Zoomed section of variant 75 from the *nif USVWZM* library[25] drawn using trace-based renderers to enable direct comparison of nucleotide data. Three data tracks are show: strand-specific RNA-seq read depths,[25] scores from an RBS prediction software, and GC percentage for a 50 bp centered moving window. Data for the *nif S* region has been highlighted. (D) Homology analysis of a CRISPRi circuit implementing a 2-input, 1-output AND gate.[26] The promoters pTet and pTac act as inputs and the g2 guide-RNA is the output. The same construct is plotted vertically and horizontally using trace-based renderers. Heat map shows the internal homology present ignoring homology that would be present between identical positions in each copy of the circuit. Highlighted regions show that part reuse and similarity of several regions within the guide-RNA sequences leads to potential hot-spots for recombination.

level of detail, the demand for this capability is likely to grow. To support the use of associated data at a base pair resolution, DNAplotlib includes functions to load trace files in the commonly used Browser Extensible Data (BED) format[24] (Figure 2C; Supporting Information).

Direct access to DNAplotlib from Python gives greatest flexibility when generating visualizations. However, in some cases it may be simpler for nonprogrammers to specify designs and part customizations in text-based files. These can be shared more easily, allow for better reuse of design or styling information among members of a lab, and support the wider adoption of standardized genetic designs. For this purpose, we provide two command-line interfaces. The first called "Quick Plotter" (quick.py) mimics the idea of Pigeon[8] and uses a simple syntax to define basic constructs as a single line of text. This is useful for the quick creation of small constructs with limited customization. The second called "Library Plotter" (plot_SBOL_designs.py) is designed for the visualization of

DOI: 10.1021/acssynbio.6b00252
ACS Synth. Biol. 2017, 6, 1115−1119

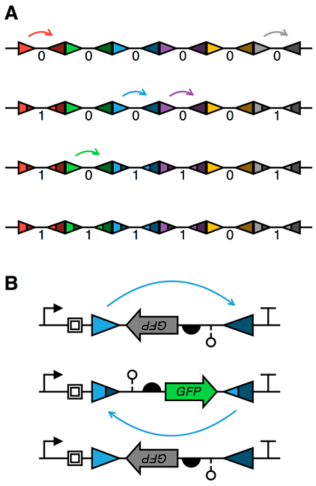


Figure 3. Extending functionality to cover new types of genetic part and regulation. Recombinase sites do not form part of the current SBOLv standard. Even so, they can be easily incorporated into DNAplotlib plots by providing custom renderers for these specific elements. (A) Array of recombinase sites implementing a 64-bit genetic memory device.[27] Current binary state is shown below each pair of recombination sites. Arrows indicate manipulations of the array at each step by integrases associated with each pair of recombination sites. (B) Illustration of a reversible recombinase NOT-gate device. In these examples, regulation arcs are used to indicate the flipping of DNA between the recombinase sites.

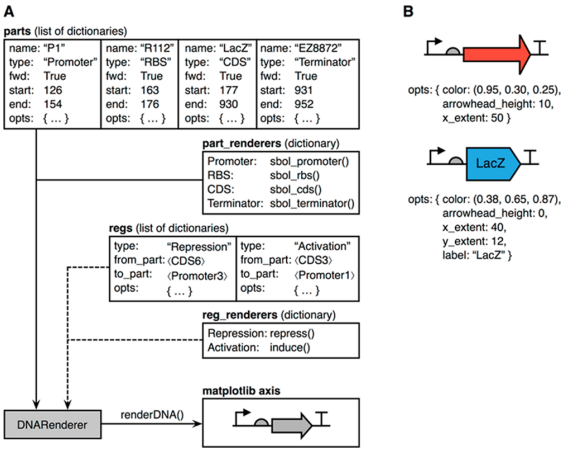


Figure 4. Data structures controlling the visualization of a genetic design. (A) DNAplotlib provides the DNARenderer object that takes design and regulatory information (parts and regs) with associated rendering functions for each element (part_renderers and reg_renderers), and then coordinates the creation of a visualization through the renderDNA(...) function. Dotted lines denote optional elements and chevrons denote part objects. All rendering is performed using a matplotlib axis to allow for the easy incorporation of other standard plotting routines. (B) The opts dictionary can be included with any part or regulatory definition to tailor the styling of the component (see Figure 5 for a full list of options). Options are shown for the coding region parts.

large design libraries. Users are required to provide several text files defining their set of parts, styling information, and the library of designs (*e.g.*, part ordering, orientation and regulatory links). These are then processed and a visualization of the full library of designs generated and saved to file.

There are also several broader functions that DNAplotlib supports. First, the visualization of genetic designs at present is a predominately manual process. Illustration tools are commonly used to draw the individual components and these are then added to existing plots of underlying data to create a final diagram. Errors in the design are time-consuming to fix and similar forms of diagram cannot be easily generated by others. While the development of a visualization using DNAplotlib might initially take a comparable amount of time, once a script is produced, it can immediately act as a template for others. For example, the novel plot shown in Figure 2D, which displays a comparison of the homology present within a genetic circuit, could immediately be used by others with minimal change, greatly simplifying the distribution of useful visual analyzes. Second, the maintenance of standards-compliant designs over time can be a challenge as standards often change and evolve with their field. Because DNAplotlib internally captures the visual standard, existing scripts merely need to be rerun to generate up-to-date diagrams. This provides a powerful means of ensuring the long-term applicability and relevance of visualizations developed.

DNAplotlib is released as open-source software. The project welcomes contributions from others within the community and all source code and a gallery of examples is available at the project Web site (www.dnaplotlib.org).

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge on the ACS Publications website at DOI: 10.1021/acssynbio.6b00252.

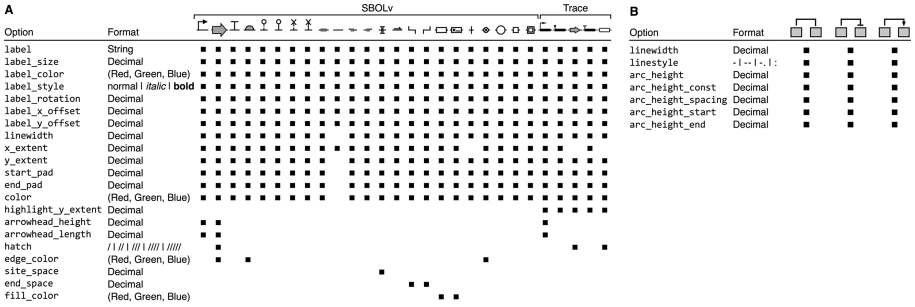


Figure 5. Customization options supported by each part and regulation type. (A) Options for all part types covering both SBOLv and trace part renderers. Black squares denote a supported option. For options with a color format, values are given as fractions of 1.0 for red, green and blue components. (B) Options for all regulation renderers.

DOI: 10.1021/acssynbio.6b00252
ACS Synth. Biol. 2017, 6, 1115−1119

Supplementary Text (PDF)
Movie S1: Repressilator dynamics (MOV)

AUTHOR INFORMATION

Corresponding Author

*E-mail: thomas.gorochowski@bristol.ac.uk.

Author Contributions

#B.S.D. and E.G. contributed equally to this work.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported by US Defense Advanced Research Projects Agency (DARPA) Living Foundries awards HR0011-13-1-0001 and HR0011-15-C-0084 (C.A.V., T.E.G. and D.B.G.); Office of Naval Research, Multidisciplinary University Research Initiative grant N00014-13-1-0074 (C.A.V.); NSF Synthetic Biology Engineering Research Center grant SynBERC EEC0540879 (C.A.V.); and National Science Foundation grant DBI-1355909 (B.A.B.).

REFERENCES

(1) Quinn, J. Y., Cox, R. S., 3rd, Adler, A., Beal, J., Bhatia, S., Cai, Y., Chen, J., Clancy, K., Galdzicki, M., Hillson, N. J., Le Novere, N., Maheshwari, A. J., McLaughlin, J. A., Myers, C. J., P, U., Pocock, M., Rodriguez, C., Soldatova, L., Stan, G. B., Swainston, N., Wipat, A., and Sauro, H. M. (2015) SBOL Visual: A Graphical Language for Genetic Designs. *PLoS Biol. 13*, e1002310.

(2) Le Novère, N., et al. (2009) The Systems Biology Graphical Notation. *Nat. Biotechnol. 27*, 735−741.

(3) Hillson, N. J., Plahar, H. A., Beal, J., and Prithviraj, R. (2016) Improving Synthetic Biology Communication: Recommended Practices for Visual Depiction and Digital Submission of Genetic Designs. *ACS Synth. Biol. 5*, 449.

(4) Lu, G., and Moriyama, E. N. (2004) Vector NTI, a balanced all-in-one sequence analysis suite. *Briefings Bioinf. 5*, 378−388.

(5) Chandran, D., Bergmann, F. T., and Sauro, H. M. (2009) TinkerCell: modular CAD tool for synthetic biology. *J. Biol. Eng. 3*, 19.

(6) Czar, M. J., Cai, Y., and Peccoud, J. (2009) Writing DNA with GenoCAD. *Nucleic Acids Res. 37*, W40−47.

(7) Chen, J., Densmore, D., Ham, T. S., Keasling, J. D., and Hillson, N. J. (2012) DeviceEditor visual biological CAD canvas. *J. Biol. Eng. 6*, 1−12.

(8) Bhatia, S., and Densmore, D. (2013) Pigeon: a design visualizer for synthetic biology. *ACS Synth. Biol. 2*, 348−350.

(9) McLaughlin, J. A., Pocock, M., Misirli, G., Madsen, C., and Wipat, A. (2016) VisBOL: Web-Based Tools for Synthetic Biology Design Visualization. *ACS Synth. Biol. 5*, 874−876.

(10) Canton, B., Labno, A., and Endy, D. (2008) Refinement and standardization of synthetic biological parts and devices. *Nat. Biotechnol. 26*, 787−793.

(11) Hunter, J. D. (2007) Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng. 9*, 90−95.

(12) Cock, P. J., Antao, T., Chang, J. T., Chapman, B. A., Cox, C. J., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B., and de Hoon, M. J. (2009) Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics 25*, 1422−1423.

(13) Sanner, M. F. (1999) Python: a programming language for software integration and development. *J. Mol. Graphics Modell. 17*, 57−61.

(14) Galdzicki, M., Clancy, K. P., Oberortner, E., Pocock, M., Quinn, J. Y., Rodriguez, C. A., Roehner, N., Wilson, M. L., Adam, L., Anderson, J. C., Bartley, B. A., Beal, J., Chandran, D., Chen, J., Densmore, D., Endy, D., Grunberg, R., Hallinan, J., Hillson, N. J., Johnson, J. D., Kuchinsky, A., Lux, M., Misirli, G., Peccoud, J., Plahar, H. A., Sirin, E., Stan, G. B., Villalobos, A., Wipat, A., Gennari, J. H., Myers, C. J., and Sauro, H. M. (2014) The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nat. Biotechnol. 32*, 545−550.

(15) Roehner, N., Beal, J., Clancy, K., Bartley, B., Misirli, G., Grunberg, R., Oberortner, E., Pocock, M., Bissell, M., Madsen, C., Nguyen, T., Zhang, M., Zhang, Z., Zundel, Z., Densmore, D., Gennari, J. H., Wipat, A., Sauro, H. M., and Myers, C. J. (2016) Sharing Structure and Function in Biological Design with SBOL 2.0. *ACS Synth. Biol. 5*, 498−506.

(16) Casini, A., Storch, M., Baldwin, G. S., and Ellis, T. (2015) Bricks and blueprints: methods and standards for DNA assembly. *Nat. Rev. Mol. Cell Biol. 16*, 568−576.

(17) Engler, C., Gruetzner, R., Kandzia, R., and Marillonnet, S. (2009) Golden gate shuffling: a one-pot DNA shuffling method based on type IIs restriction enzymes. *PLoS One 4*, e5553.

(18) Appleton, E., Tao, J., Haddock, T., and Densmore, D. (2014) Interactive assembly algorithms for molecular cloning. *Nat. Methods 11*, 657−662.

(19) Weber, E., Engler, C., Gruetzner, R., Werner, S., and Marillonnet, S. (2011) A Modular Cloning System for Standardized Assembly of Multigene Constructs. *PLoS One 6*, e16765.

(20) Nielsen, A. A. K., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., and Voigt, C. A. (2016) Genetic circuit design automation. *Science 352*, aac7341.

(21) Elowitz, M. B., and Leibler, S. (2000) A synthetic oscillatory network of transcriptional regulators. *Nature 403*, 335−338.

(22) Kent, W. J., Sugnet, C. W., Furey, T. S., Moskin, K. M., Pringle, T. H., Zahler, A. M., and Haussler, D. (2002) The Human Genome Browser at UCSC. *Genome Res. 12*, 996−1006.

(23) Robinson, J. T., Thorvaldsdóttir, H., Winckler, W., Guttman, M., Lander, E. S., Getz, G., and Mesirov, J. P. (2011) Integrative genomics viewer. *Nat. Biotechnol. 29*, 24−26.

(24) Quinlan, A. R., and Hall, I. M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics 26*, 841−842.

(25) Smanski, M. J., Bhatia, S., Zhao, D., Park, Y., L, B. A. W., Giannoukos, G., Ciulla, D., Busby, M., Calderon, J., Nicol, R., Gordon, D. B., Densmore, D., and Voigt, C. A. (2014) Functional optimization of gene clusters by combinatorial design and assembly. *Nat. Biotechnol. 32*, 1241−1249.

(26) Nielsen, A. A., and Voigt, C. A. (2014) Multi-input CRISPR/Cas genetic circuits that interface host regulatory networks. *Mol. Syst. Biol. 10*, 763.

(27) Yang, L., Nielsen, A. A., Fernandez-Rodriguez, J., McClune, C. J., Laub, M. T., Lu, T. K., and Voigt, C. A. (2014) Permanent genetic memory with > 1-byte capacity. *Nat. Methods 11*, 1261−1266.

DOI: 10.1021/acssynbio.6b00252
ACS Synth. Biol. 2017, 6, 1115−1119

Chapter 3

BUILD

3.1 Introduction

One of the pioneer standards for genetic engineering was a DNA assembly method called "BioBrick standard assembly". A BioBrick is a DNA molecule which contains some unit of genetic function, such as a protein coding sequence, a binding sequence for a regulatory protein, or some other type of genetic function. BioBricks are propagated in and harvested from host bacteria as plasmid molecules with standardized DNA sequences. These standardized DNA sequences define regions where the DNA can be restricted (cut) and ligated (joined). Standard assembly thus allows BioBricks to be composed into complex genetic assemblages. The BioBrick assembly method was popularized by the iGEM competition and proved simple enough that teams of undergraduates quickly generated thousands of unique DNA constructs [27]. However, our experience on UW's iGEM team in 2008 revealed shortcomings in the standard. For one thing, BioBricks assembly leaves artifacts at assembly junctions ("scars") which in many cases unpredictably affect the biological function of the DNA, thus violating the assumption that standard biological parts are interchangeable. The other problem is that it is slow, taking approximately 5 days to design, build, and test a simple construct in *E. coli*. As a result we began to investigate new DNA assembly methods that did not use standard joiners and published an optimized protocol for BioBrick assembly [28].

Our methods were faster, had fewer steps where things could go wrong, and required less DNA substrate, but they relied on customized (rather than standardized) DNA assembly

procedures in order to join BioBricks together. Specifically, custom DNA oligonucleotides had to be synthesized and ordered for each assembly. Since then, BioBrick standard assembly has fallen out of favor. Many labs now use Gibson assembly [14], another technique which also uses customized DNA oligonucleotides for joining blocks of DNA. The lesson from this experience was that biological parts do not quite fit the ideal definition of interchangeable components. A sequence assembled from DNA parts often requires customization and optimization in order to achieve the desired biological behavior.

Synthetic biologists are now engineering DNA constructs on the scale of tens of thousands of nucleotide bases. This process often requires troubleshooting DNA assembly errors or mutations in the target construct. In the manuscript Quality Control for Large-scale DNA Assembly, I discuss the pros and cons of current quality control practices, and propose three new best practices for quality control that will help synthetic biologists build bigger and better.

3.2 Improved Methods for DNA Assembly

2624–2636 Nucleic Acids Research, 2010, Vol. 38, No. 8 Published online 12 April 2010
doi:10.1093/nar/gkq179

In-Fusion BioBrick assembly and re-engineering

Sean C. Sleight*, Bryan A. Bartley, Jane A. Lieviant and Herbert M. Sauro

Department of Bioengineering, University of Washington, Seattle, WA 98195, USA

Received November 30, 2009; Revised March 1, 2010; Accepted March 2, 2010

ABSTRACT

Genetic circuits can be assembled from standardized biological parts called BioBricks. Examples of BioBricks include promoters, ribosome-binding sites, coding sequences and transcriptional terminators. Standard BioBrick assembly normally involves restriction enzyme digestion and ligation of two BioBricks at a time. The method described here is an alternative assembly strategy that allows for two or more PCR-amplified BioBricks to be quickly assembled and re-engineered using the Clontech In-Fusion PCR Cloning Kit. This method allows for a large number of parallel assemblies to be performed and is a flexible way to mix and match BioBricks. In-Fusion assembly can be semi-standardized by the use of simple primer design rules that minimize the time involved in planning assembly reactions. We describe the success rate and mutation rate of In-Fusion assembled genetic circuits using various homology and primer lengths. We also demonstrate the success and flexibility of this method with six specific examples of BioBrick assembly and re-engineering. These examples include assembly of two basic parts, part swapping, a deletion, an insertion, and three-way In-Fusion assemblies.

INTRODUCTION

Synthetic biology is an emerging discipline that aims to design and construct novel biological organisms programmed by genetic circuits. Many synthetic biologists assemble genetic circuits from standardized biological parts called BioBricks. Examples of BioBricks include promoters, ribosome-binding sites (RBS), protein or RNA-coding sequences, and transcriptional terminators. Currently every BioBrick is a physical DNA sequence on a circular plasmid that is stored and distributed by the Registry of Biological Parts (http://www.partsregistry.org) as lyophilized DNA in 384-well plate format. Standardized sequences on BioBricks enable Standard Assembly of two BioBricks via restriction enzyme digestion and ligation in an idempotent fashion (Figure 1a) (1–5). Standard Assembly involves digestion of two plasmids with different restriction enzymes that leave compatible sticky ends which can be ligated together into a new plasmid. This effectively replaces the restriction sites between the assembled parts with a 'scar' sequence, allowing for the new BioBrick to be later combined with other BioBricks. This standardized procedure takes much of the planning out of DNA fragment assembly since the same restriction enzymes can be used for every assembly reaction.

Currently several BioBrick assembly standards (http://openwetware.org/wiki/The_BioBricks_Foundation:RFC) have been proposed to improve upon the original BioBrick standard, largely due to the fact that this original standard produces an 8-bp scar between assembled BioBricks and hence does not allow for the creation of fusion proteins. Nearly all of these current assembly standards involve assembly by restriction enzyme digestion and ligation. There are also several PCR-based methods currently being used for DNA assembly that have the potential for standardization. These methods include In-Fusion (6,7), SLIC (8), T5 exonuclease recombination (9), USER (10), oligonucleotide assembly (11) and SOEing (12). The former four methods generally involve converting overlapping, blunt-end PCR products into fragments with sticky overhangs that can anneal to form circular plasmids, but the method for generating the overhangs differs. For example, the SLIC method (8) uses T4 DNA polymerase while the USER method (10) uses a uracil exonuclease. Unlike restriction digestion, the site at which the overhangs are created is generally not constrained by a specific sequence. The latter two methods involve overlapping oligonucleotides with a PCR-amplified vector (11) or extending overlapping PCR products (12) and do not use subsequent enzymatic treatment.

Here, we describe an alternative BioBrick assembly method that allows for BioBricks to be quickly assembled and re-engineered using the Clontech In-Fusion PCR Cloning Kit (6,7). The proprietary In-Fusion enzyme with exonuclease activity fuses together any PCR

*To whom correspondence should be addressed. Tel: +1 206 616 1928; Fax: +1 206 685 3300; Email: sleight@u.washington.edu

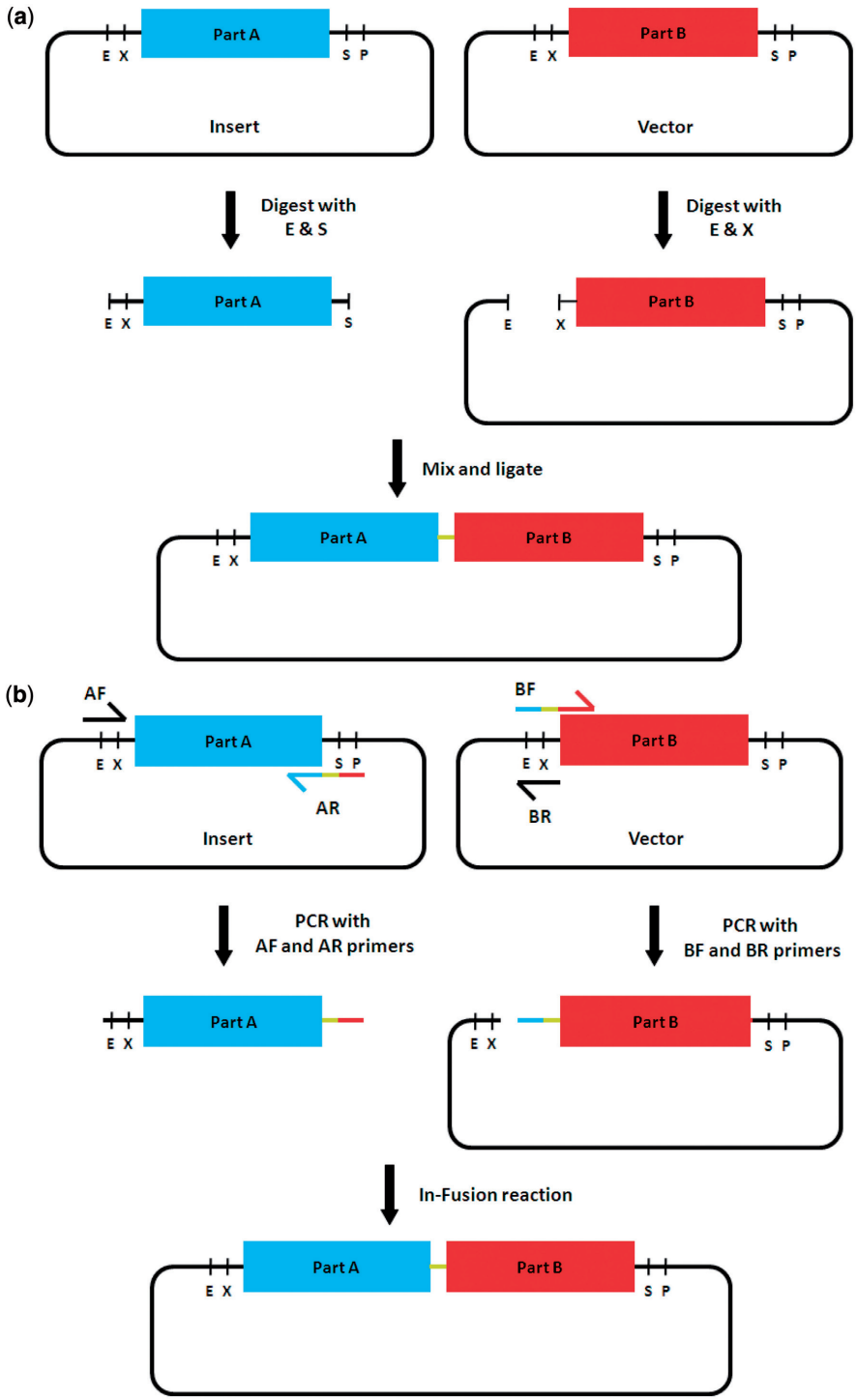


Figure 1. Standard assembly versus In-Fusion assembly. (**a**) Standard Assembly of two BioBricks (Parts A and B) involves restriction digestion and ligation. Both parts are on pSB1A2 vectors encoding ampicillin resistance. The Part A plasmid is digested with EcoRI (E) and SpeI (S), while the second plasmid is digested with EcoRI (E) and XbaI (X). SpeI and XbaI restricted fragments have compatible sticky ends for ligation. The desired digested fragments are gel purified and ligated together to create the assembled plasmid with both parts. A scar sequence is left between both parts that does not have the original restriction site and the restriction sites flanking the parts are maintained. (**b**) In-Fusion assembly of two BioBricks involves PCR, purification, and a subsequent In-Fusion reaction. Parts A and B are PCR-amplified (in this example the vector is amplified with Part B) and purified without gel extraction. Each assembly requires four primers, where two are specific to the junction (parts to assemble) and two are general vector primers. BioBrick Part A (blue) and Part B (red) are on pSB1A2 plasmids encoding ampicillin resistance. Primers described in 'Materials and Methods' section are color-coded to show their homology. The thick black line indicates BioBrick prefix or suffix homology on the pSB1A2 vector. The yellow sequence is the scar that is normally between parts after standard BioBrick assembly, if this is desired, or can be a linker sequence for a fusion protein. The purified PCR products are mixed together in the In-Fusion reaction to create a circular plasmid. Restriction sites flanking the parts maintain the standard BioBrick format.

product with a linearized vector into a circular plasmid when the fragments share at least 15 bp of homology on both ends. To assemble two BioBricks, one PCR-amplified BioBrick needs to have homology on each end with the second PCR-amplified BioBrick (vector amplified with the BioBrick) to allow for the fragments to be fused together in the In-Fusion reaction (Figure 1b). More than two BioBricks can be fused together as shown in the examples below and four fragments have been successfully fused (6). This method can also be used to re-engineer BioBricks as shown in the examples below and we have also successfully used this method for vector replacement, site-directed mutagenesis and cumbersome transcriptional terminator re-engineering (results not shown).

In-Fusion Assembly greatly expands the possible circuits to construct since in our experience, the individual parts we want to construct into circuits often do not exist as BioBricks on their own or in the correct order. For this reason In-Fusion provides the flexibility to PCR-amplify any sequence from any part, then assemble these fragments together in one step. In our experience, the advantages of In-Fusion BioBrick assembly over Standard Assembly are that it is faster, more flexible, and has a high success rate (see Supplementary Figure S1 for a timeline and schematic overview for Standard and In-Fusion BioBrick Assembly). In-Fusion Assembly is fast once primers are available since BioBricks can be immediately PCR-amplified from parts extracted from the BioBrick Parts Distribution plate. Standard assembly requires an initial amplification of the BioBrick through transformation, overnight growth, and plasmid extraction. Standard assembly also requires tedious extraction of restricted DNA fragments from a gel, more intermediate enzymatic reactions, and more time to quantitate and optimize these reactions. In-Fusion assembly is more flexible in the sense that there is more control over the exact engineered sequence and mutations can be easily introduced with mutagenic primers (6). We have found this method to have a high success rate in that nearly every assembly reaction yields the desired construct. Standard assembly we have found to be less reliable.

The disadvantages of In-Fusion BioBrick assembly are that the specific assembly supplies are more expensive (~$15/assembly versus ~$5/assembly for Standard Assembly), custom primers are required, and occasionally there are mutations in assembled plasmids. Mutations in the construct may depend on a number of factors, including primer quality and the error rate of the polymerase. However, using reagents specified in the 'Materials and methods' section, we demonstrate here that such mutations are sufficiently rare that typically only a single putative construct needs to be sequenced. In general, In-Fusion assembly is ideal when re-engineering an existing BioBrick with many parts since there are less assembly reactions to perform, or when assembling a circuit with specific parts distributed among several BioBricks. On the other hand, if the BioBricks you want to assemble already exist as digested fragments in the freezer, then Standard Assembly may be ideal for this particular situation. The motivation for optimizing and adapting the In-Fusion method for BioBrick assembly is to improve the success rate, flexibility, and speed for constructing genetic circuits and create a new BioBrick assembly standard (13). Small synthetic biology labs that do not perform high throughput restriction enzyme digests and ligations will especially benefit from using this method.

MATERIALS AND METHODS

Primer design rules

For the In-Fusion reaction to work, the forward primer of the first PCR-amplified fragment must have at least 15-bp homology to the reverse primer of the second fragment, and vice versa. A longer homology length is used in this protocol because in our experience it works better than 15 bp. There are two simple primer design rules that allow for semi-standardized assembly, in the sense that even though the same components cannot be used repeatedly as in a standardized assembly method, following these rules will remove much of the planning required for performing In-Fusion Assembly reactions. The two rules are: (i) the reverse primer for Part A (AR) should be the last 20 bases Part A + scar (if wanted) + first 20 bases Part B (reverse complement of this entire sequence), and (ii) the forward primer for Part B (BF) should be the reverse complement of the AR primer. Since the primers are exactly complementary, this gives at least 40 bp of homology at the junction of Parts A and B in total. In contrast, the forward primer of Part A (AF) and the reverse primer of Part B (BR) do not need customization. These primers are specific to the standard Biobrick 'prefix' (immediately upstream of each part) or 'suffix' (immediately downstream of each part) and may be re-used when assembling different constructs. Of course optimization of the primers may improve chances of a clean PCR product. In the assembly examples described here using these primer design rules, we found that every PCR product amplified with a single amplicon (10/10 PCR products). One particular PCR reaction required a gradient on the primer annealing step during the PCR reaction to generate a single PCR product, but in general a 55°C annealing temperature worked well for all reactions. In general we found these primer design rules to be robust and other examples not described here have used these rules with success.

Primer design software tool

We have designed a software tool for In-Fusion assembly primer design (located at http://sys-bio.org/primerdesign/). This tool allows you to input the DNA sequences for each part to assemble, the scar sequence between the parts, and the length of overlap with the adjoining part (the default length is 20). When 20 bases of homology are added to the 5′-end of each of the primers, along with the 8-base scar, this gives a total junction homology of 48 bp. The program outputs the forward and reverse primers for the Part A and Part B junction (AR and BF primers). The primer sequences generated are based on the rule described above. A future version of this

tool will check if there are repeated sequences (such as a scar sequence) located at the 3′-end of the primer and if so, will extend the primer to two bases past the repeated sequence to ensure specificity. Other future additions will include T_m calculations, GC content, and other relevant primer design specifications. The vector-specific primers (AF and BR) are already standardized and do not need to be designed with this tool.

Maximizing success rates

There are a few additional guidelines to follow to maximize success with this method. First, do not use repeated part sequences (e.g. scar sequences, transcriptional terminators, etc.) on the 3′ end of your primer to avoid multiple PCR products. The primer design tool described above will eventually check for this. Second, it is better to PCR-amplify the vector (plasmid backbone) with the smaller part to assemble especially if the part is less than 100 bp since the molar ratio between both fragments should be as close as possible. Third, dilute the template DNA as much as possible to avoid 'background' plasmids from being transformed later (between 100 and 500 pmol DNA works well for miniprepped plasmids, while plasmids from the 2009 Parts Registry plate differ in concentration).

Primers designed and used in this study

All primers were ordered from Integrated DNA Technologies (IDT) unmodified with standard desalting at the 25 nmol scale. Four primers specific to the pSB1A2 vector (http://partsregistry.org/Part:pSB1A2) were designed and used in this study. In Figure 2a, the Part A forward primer (AF) is: 5′-TTCTGGAATTCGCG GCCGCTTCTAG-3′ (specific to the pSB1A2 non-coding sequence prefix + 5 bases upstream of the prefix). The Part B reverse primer (BR) is: 5′-CTAGAAGCGGCCGCGA ATTCCAGAA-3′ (reverse complement of the AF primer). In Figure 2b, the Part A forward (AF) primer is: 5′-TACT AGTAGCGGCCGCTGCAGGCTTC-3′ (specific to the pSB1A2 suffix + 5 bases downstream of the suffix). The Part B reverse primer (BR) is: 5′-GAAGCCTGCAGCGG CCGCTACTAGTA-3′ (reverse complement of Part A forward primer). AR and BF primers for both figures were customized depending on the parts to assemble or re-engineer. We used 48 bp of homology between parts for assembly reactions as described above except in the example shown in Figure 6 where we used 15 bp.

PCR and template DNA for PCR

Phusion High Fidelity PCR Mastermix was used for PCR in a 25 or 50 µl reaction volume. Reaction steps were used following the Phusion protocol. An amount of 100 pmol forward and reverse primers were added to each reaction. A volume of 1 µl of a 1:1000 diluted miniprepped plasmid or plasmid extracted from the Registry plate (~100–500 pmol total) was added to the PCR reaction for template DNA. PCR was performed in an Eppendorf Mastercycler EP S Gradient thermocycler according to the manufacturer's instructions with the annealing step of 55°C for 30 s. Eppendorf PCR tubes (0.2 ml) were used for all reactions. Digestion with DpnI after the PCR reaction may reduce background plasmids from being transformed later, but in our experience diluting the PCR template reduces background sufficiently that DpnI digestion is unnecessary. In most cases, this eliminates an extra step in the assembly process.

PCR product purification and quantitation

PCR products were purified with the Qiagen PCR Purification Kit and eluted with 30 µl of molecular grade water. PCR products were quantitated with a Nanodrop (Thermo Scientific).

In-Fusion reaction and transformation

In-Fusion reactions were performed using the recommended protocol with some extra details and exceptions noted here. A 2:1 insert:vector molar ratio was normally used with 100 ng of vector for two-way reactions and a 2:2:1 insert:insert:vector molar ratio was normally used with 100 ng of vector for three-way assemblies (exceptions noted below). The 10 µl volume consisting of insert, vector, and molecular grade water was transferred into the In-Fusion dry-down reaction tube and mixed by pipetting up and down several times. This reaction was transferred into a 0.2 ml PCR tube and incubated in a thermocycler for 15 min at 37°C followed by 15 min at 50°C. A volume of 30 µl TE Buffer (pH 8.0) was added to the tube and mixed by pipetting up and down several times. A volume of 2.5 µl of this mixture was added to one 50-µl tube of Fusion-Blue chemically competent cells thawed on ice. Cells were incubated on ice for 30 min, heat shocked at 42°C for 45 s, and put back on ice for 1 min. A volume of 200 µl SOC media was added to the cells and mixed by pipetting up and down a few times. The one hour incubation time was not necessary when transforming plasmids conferring ampicillin resistance, but was required with kanamycin resistance conferring plasmids. A volume of 200 µl of the transformant culture was spread on an LB plate supplemented with 100 µg/ml of ampicillin (without the centrifugation steps in the protocol). This procedure normally gives 30–1000 colonies after overnight incubation at 37°C.

Quantitating success and mutation rates with different homology and primer lengths

The assay described in Figure 3 and results in Tables 1 and 2 quantitated the effect of homology and primer lengths on success and mutation rates. Success rate for each assembly was measured by simply counting the number of fluorescing versus non-fluorescing transformant colonies on LB plates supplemented with 100 µg/ml of ampicillin and 0.1 mM IPTG. Mutation rate for each assembly was determined by sequencing eight plasmids extracted from fluorescing cultures grown from individual clones.

Colony PCR

Screening for the desired construct was generally performed by colony PCR, with the exception of functional

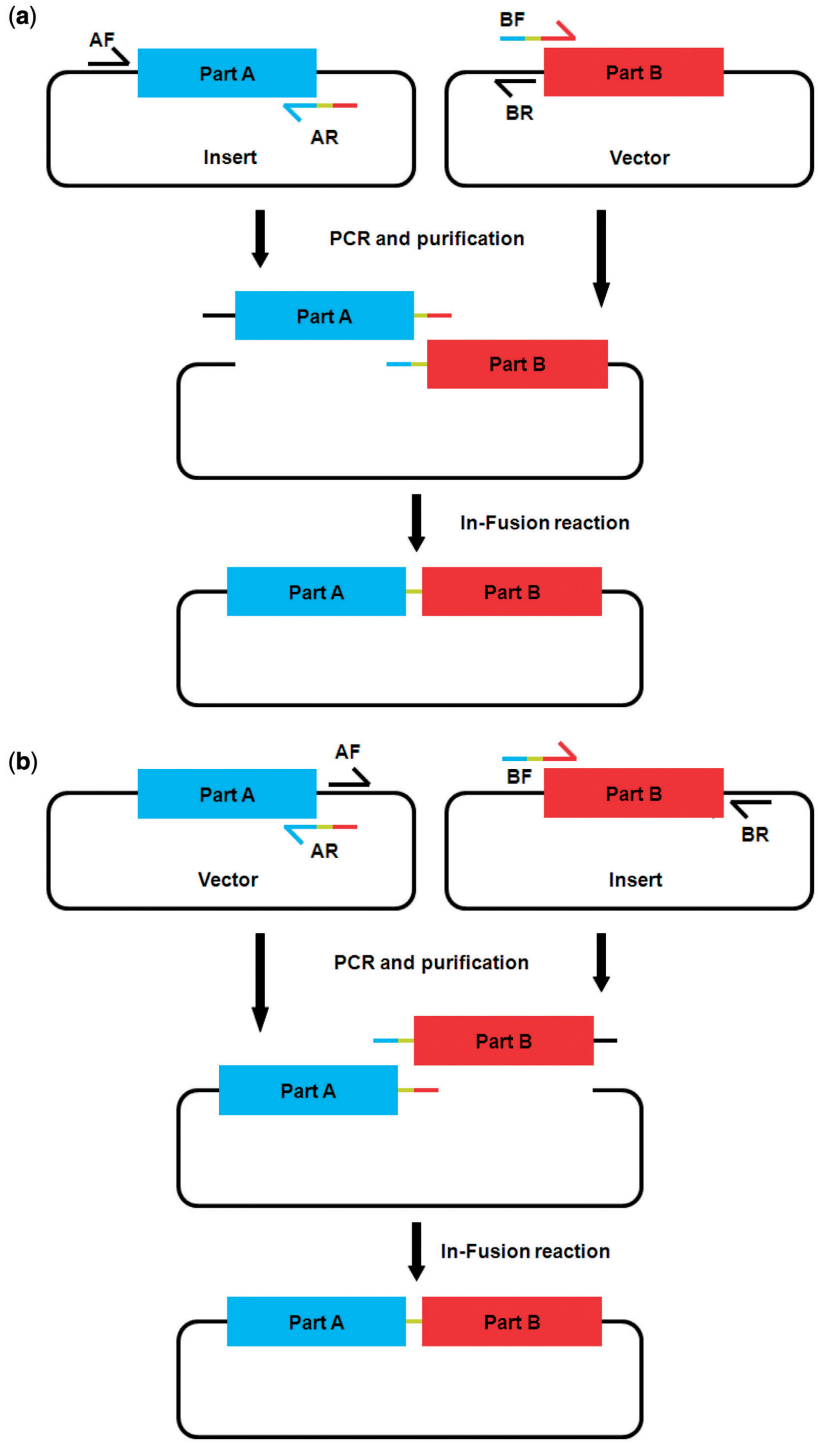


Figure 2. General strategies for In-Fusion BioBrick assembly. There are two general strategies for In-Fusion assembly on pSB1A2 plasmids depending on which BioBrick is PCR-amplified with the vector. The first strategy PCR-amplifies the upstream part by itself and the downstream part with the vector (**a**). The second strategy PCR-amplifies the upstream part with the vector and the downstream part by itself (**b**). Each strategy requires four primers, where two are specific to the parts to assemble and two are general vector primers. The vector primers are described in the 'Materials and Methods' section and can either be specific to the prefix or suffix. When Part B is amplified with the vector, prefix primers are used. When Part A is amplified with the vector, suffix primers are used. All other details are described in Figure 1b.

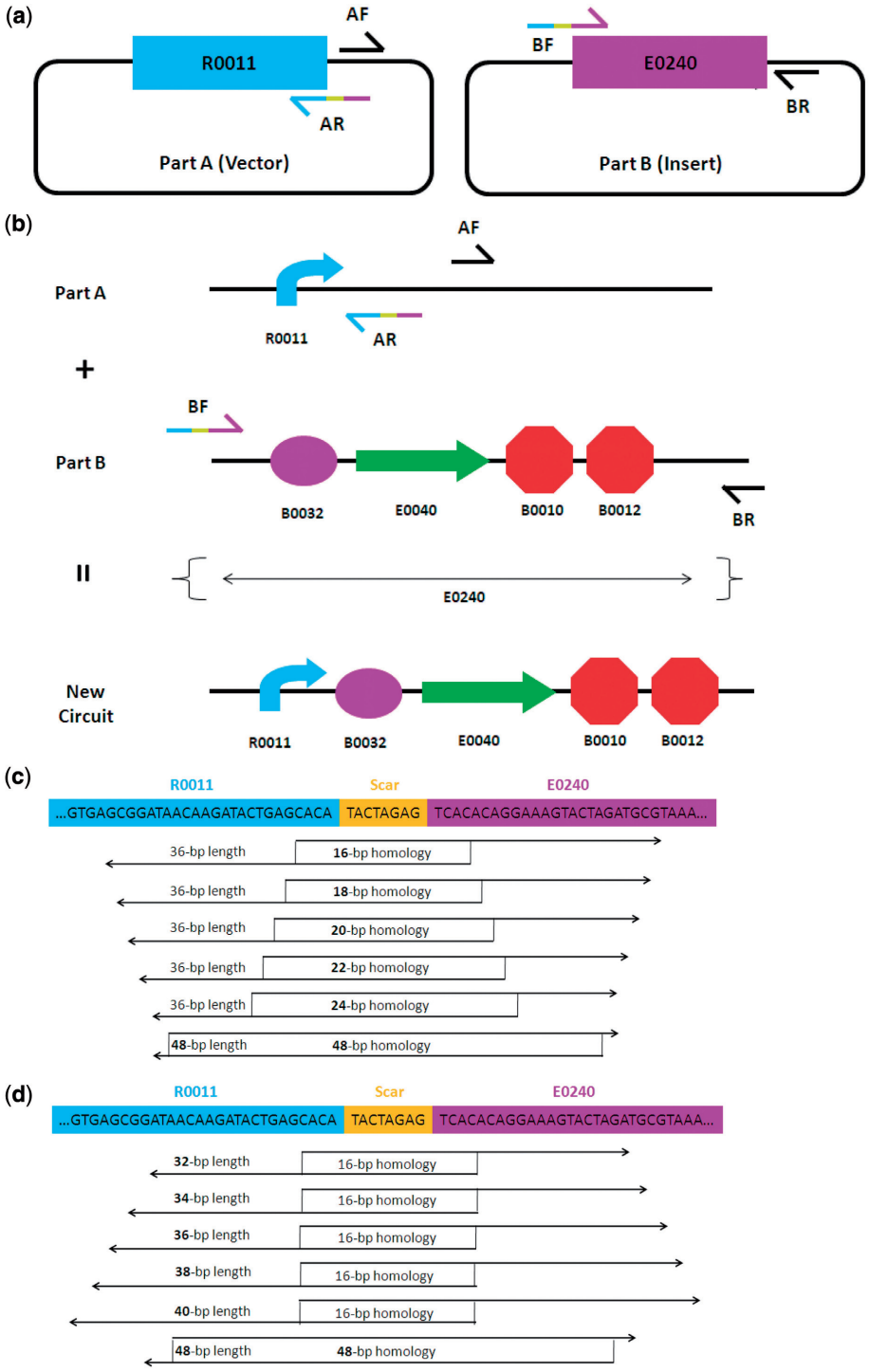


Figure 3. Assembly of two basic parts. (**a**) R0011 and E0240 plasmids are shown with the forward and reverse primers for PCR. R0011 is amplified with the vector and E0240 is amplified as the insert. (**b**) Detailed schematic of the assembly strategy with the forward and reverse primers. The R0011-amplified PCR product has an additional scar sequence and RBS that has homology to the E0240-amplified PCR product. (**c**) Primer design schematic to test the effect of homology length on success and mutation rates. The desired sequence for R0011, scar sequence, and E0240 is shown. For each primer pair, forward primers are shown as right arrows, reverse primers as left arrows, and the region of homology between primers is boxed. Homology and primer lengths are shown where bolded numbers highlight differences between primer pairs. (**d**) Primer design schematic to test the effect of primer length on success and mutation rates. Details are similar to that described in (c).

Table 1. Success and mutation rates of In-Fusion assembly using different homology lengths

Homology level	Success rate	Junction mutation rate
16	61.8% (222/359)	0% (0/7)
18	64.6% (197/305)	0% (0/8)
20	73.1% (196/268)	0% (0/8)
22	74.2% (291/392)	0% (0/6)
24	74.4% (314/422)	0% (0/8)
48[a]	84.8% (495/584)	0% (0/7)

Homology length is indicated with the success and mutation rates. For success rates, numbers in parentheses after the rates indicate the number of successful clones out of the number of total clones tested. For mutation rates, numbers in parentheses after the rates indicate the number of junction mutations out of the number successful clones sequenced.
[a]Primer length is 48 bases (all others are 36 bases).

screening described in the previous section. In these cases, success rate is defined as the percentage of correct constructs out of the total number of colonies screened by colony PCR. Colony PCR allows us to discriminate between the desired construct and construction background (PCR template plasmid that co-transforms with the newly assembled plasmid).

Colony PCR was performed on at least six colonies using the VF2 (5′-TGCCACCTGACGTCTAAGAA-3′) and VR (5′-ATTACCGCCTTTGAGTGAGC-3′) primers specific to the pSB1A2 vector (~100 bp on either side of the part) or primers specific to the desired construct. Negative control reactions were also performed using VF2/VR primers on the plasmid template DNA for the original PCR reaction. Thus, a colony hosting the desired construct will exhibit a mobility shift when compared with the negative control reactions. Colony PCR reactions were performed using 10 µl Fermentas PCR Master Mix per reaction in 0.2 ml PCR tubes. All 10 µl of the colony PCR products were run out on a 1% agarose gel stained with SYBR Safe (Invitrogen) with 1-kb ladder (NEB).

Plasmid extraction and sequencing

At least three correct colonies as identified by colony PCR were grown in test tubes with 5 ml LB supplemented with ampicillin (100 µg/ml) shaking overnight at 250 rpm. We chose three to ensure that one plasmid was correctly constructed without mutations which tend to occur most often at the junctions (regions of homology between fragments). Minipreps were performed with the Qiagen Miniprep Kit using 3 ml culture volume and plasmids were eluted with 30 µl of molecular grade water. Plasmids were submitted for sequencing with VF2 and VR or custom primers to sequence the entire genetic circuit.

General strategies for In-Fusion assembly

We devised two general strategies for assembly depending on which BioBrick is PCR-amplified with the vector (Figure 2). Since both PCR products need to have homology to each other on both ends for the In-Fusion reaction to work, the simplest strategy was amplify each PCR product to have homology to the vector at one end and to have homology at the junction between parts at the other end. As shown in Figure 2a, to amplify the upstream part (Part A) as the insert and the downstream part (Part B) as the vector, the vector specific primers need to be upstream of each part (i.e. in the vector prefix).

Since the forward primer for Part A (AF) and reverse primer for Part B (BR) are complementary sequences, the resulting PCR products share 25 bp of homology at the site where Parts A and B will be joined in the In-Fusion reaction. Likewise, as shown in Figure 2b, to amplify the upstream part (Part A) as the vector and the downstream part (Part B) as the insert, the vector specific primers need to be downstream of each part (i.e. in the vector suffix). Use of these vector-specific primers allows for BioBrick standard formats to be maintained so that new In-Fusion assembled constructs can be submitted to the Registry. These four primers described in the 'Materials and Methods' section can be re-used for every In-Fusion Assembly.

However, custom primers need to be designed in order to provide homology at the junction between parts. The AR and BF primers shown in both Figure 2a and b are complementary, resulting in PCR products with homologous ends that will be fused in the In-Fusion reaction. These primers were designed with simple rules of having a sequence specific to the template DNA of one part at the 3′-end and a 5′ overhang that is homologous to the other part (see 'Materials and Methods' section for details). Following these simple rules will allow for primers to be designed quickly and in a semi-standardized fashion.

RESULTS

We first describe the success rate and mutation rate of In-Fusion assembly using different homology and primer lengths. We then describe additional examples of In-Fusion assembly and re-engineering reactions to demonstrate the versatility and success of this method. These examples include assembly of two basic parts, part swapping (simultaneous promoter and RBS re-engineering), a deletion (conversion of a polycistronic transcribed sequence into a fusion protein-coding sequence), an insertion (of a degradation tag), and three-way assemblies (one to insert an antibiotic resistance gene and swap out a terminator, and another to construct a fusion protein). The insertion and deletion examples are illustrated in Supplementary Figures S2 and S3.

Effect of homology and primer length on success rate and mutation rate of In-Fusion assembly

To understand how homology and primer length affects the success rate and mutation rate of assembly reactions, we devised an assembly assay that would allow for these rates to be determined for a large number of individual clones. The two basic BioBrick parts used for the assembly assay were R0011 (lacI-regulated promoter) and E0240

(consisting of an RBS, GFP-coding sequence and double transcriptional terminator) (Figure 3a). By amplifying the vector with R0011, the PCR product obtained (Part A) starts with the suffix and ends with R0011 plus the E0240 junction sequence (Figure 3b). Part B starts with the R0011 junction sequence and ends with the suffix (Figure 3b). Fusion of these two PCR products occurs at each end and creates a circular plasmid. Transformation of the assembly reaction into competent cells allows for the success rate to be easily determined since colonies can only fluoresce if they have the correctly assembled R0011 and E0240 together in the same plasmid.

We designed primers to vary the homology level from 16–24 bp in increments of 2 bp while keeping the primer length constant at 36 bases (Figure 3c). We chose 16–24 bp because the minimum homology length is 15 bp and this would allow us to determine the relationship between success rates around this minimum at a fine scale. We predicted higher homology levels would increase success rate, but also possibly increase mutation rate at the same time. The 36-base primer length was chosen since it is known that longer primer lengths increases the chances of mutations occurring in the primer. We also used primers that maximized the homology and primer length with 48 bp of homology and a primer length of 48 bases. We predicted that higher homology would increase success rate since another study observed this relationship (8), but at a cost of having a higher mutation rate. We also designed primers to vary the primer length from 32 to 40 bases in increments of 2 bases since we predicted longer primers would increase mutation rate (Figure 3d). Therefore, there were a total of 10 assembly reactions performed in parallel: five to test homology lengths, five to test primer lengths (the 36-base length primers overlapped with the 16 base homology length assembly), and one extreme assembly of 48 bp of homology and a primer length of 48 bases.

These 10 assembly reactions required 20 PCR products, one insert and one vector for each. Remarkably, all 20 PCR products amplified using an annealing temperature of 55°C. Table 1 shows the results of homology length on the success rate and mutation rate of In-Fusion assembly. As homology level increases, the assembly success rate also increases. There appears to be small difference between low homology (16–18 bp) and high homology (20–24 bp) with high homology increasing success by about 10% on average. The extreme case of 48 bp of homology had the highest success rate at nearly 85%. We therefore decided that 48 bp of homology should be used for our primer design rules to maximize success rates. Forty-eight base pairs should not be considered the optimal length, but is a conservative choice since no homology lengths were tested between 24 and 48 bp.

Remarkably, there were no mutations in any of the junctions when we sequenced several successful clones. For all 10 assemblies, we only found one mutation among all of the clones sequenced (73 clean sequences out of 80 attempts) for the entire ~1-kb genetic circuit. Table 2 shows the effect of primer length on the success and mutation rates of In-Fusion assembly. There do not appear to be any remarkable differences between primer lengths of 32–40 bases with respect to either success or mutation rate. Therefore, the optimal homology and primer length is 48 bp because this achieves the highest success rate without the cost of high mutation rate since mutations are rare. The rare mutations in the circuit itself are because Phusion is a very high fidelity polymerase (4.4×10^{-7} according to the product spec sheet), but we can not rule out the possibility that the mutation existed on the template DNA at a low level.

Part swapping: simultaneous promoter and RBS replacement

Next we wanted to construct the same R0011 + E0240 circuit, but with RFP (E1010) instead of GFP (E0040). This would have taken three Standard Assembly steps: to first construct R0011 with B0032, E1010 with B0010/12, then to assemble all these parts together. Instead we could simultaneously re-engineer the existing J04450 circuit with a new promoter and RBS in a single step using In-Fusion (Figure 4a). To do this, for Part A, R0011 and B0032 were first amplified with the E1010 junction sequence and vector (Figure 4b). E1010 and B0010/12 were then amplified from J04450 with the B0032 junction sequence to make Part B (Figure 4b). Colony PCR results remarkably showed that all six colonies had the correct construct, but we were able to screen out negative colonies by the colony color for this particular assembly. Two separate colony PCR experiments were performed on the same six slightly red glowing colonies, one to determine the size of the insert (Figure 4c, left) and one to identify colonies that had the correct RBS (Figure 4c, right). Since the negative control J04450 plasmid (#7 in the right gel photo) did not amplify with the B0032-specific primer and the six colonies did amplify, it was assumed that all six colonies (100%) had the correct construct and three of these were verified with sequencing.

Table 2. Success and mutation rates of In-Fusion assembly using different primer lengths

Primer length	Success rate	Junction mutation rate
32	64.3% (198/308)	0% (0/8)
34	66.7% (246/369)	0% (0/8)
36	61.8% (222/359)	0% (0/7)
38	65.9% (245/372)	0% (0/7)
40	58.5% (220/376)	0% (0/6)
48[a]	84.8% (495/584)	0% (0/7)

Primer length is indicated with the success and mutation rates. For success rates, numbers in parentheses after the rates indicate the number of successful clones out of the number of total clones tested. For mutation rates, numbers in parentheses after the rates indicate the number of junction mutations out of the number successful clones sequenced.
[a]Homology level is 48 bases (all others are 16 bases).

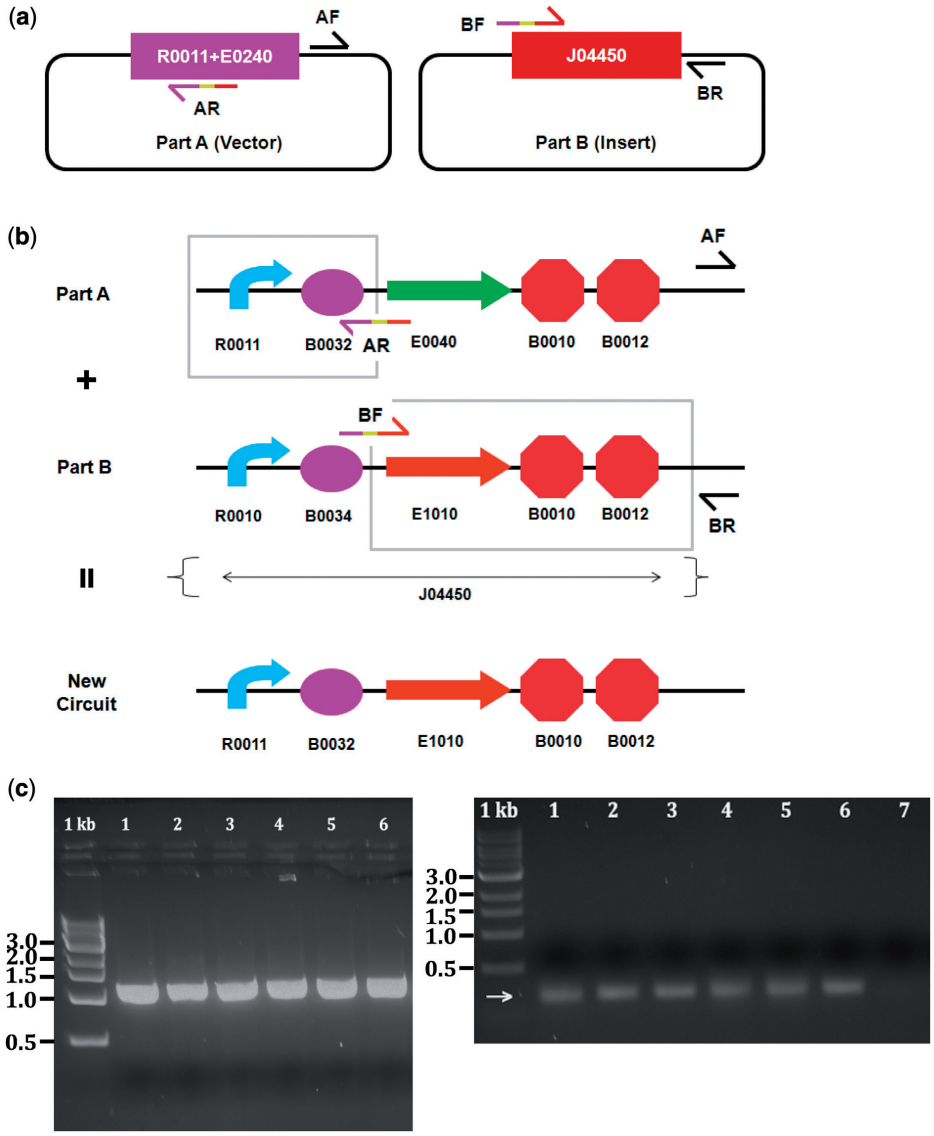


Figure 4. Part swapping: simultaneous promoter and RBS re-engineering. (**a**) R0011+E0240 and J04450 plasmids are shown with the forward and reverse primers for PCR. R0011+E0240 is amplified with the vector and J04450 is amplified as the insert. (**b**) Detailed schematic of the assembly strategy with the forward and reverse primers. Only the promoter (R0011) and RBS (B0032) are PCR-amplified from the R0011+E0240 plasmid. Only E1010 and B0010/12 are PCR-amplified from the J04450 circuit in order to change its promoter and RBS in one assembly step. (**c**) Since both plasmids used as template DNA in the PCR reaction were approximately the same size as the desired construct, two colony PCR reactions were performed on the same six colonies. The gel on the left shows six colonies amplified with VF2/VR primers and the gel on the right shows the same six colonies (#1–6) and negative control J04450 plasmid (#7) amplified with the VF2 and R0011+E0240 AR primer. Correct colonies show a PCR product of about 1.1 kb for the left gel and a PCR product of about 300 bp for the right gel (correct size indicated by arrow).

Three-way assembly: insertion of an antibiotic resistance gene and terminator swapping

Since insertion of DNA into the middle of a multi-part construct takes several Standard Assembly reactions, we demonstrate the efficiency of this construction in a three-way In-Fusion reaction. We wanted to insert the B0032 RBS and kanamycin resistance gene into the T9002 (Lux receiver) circuit, while at the same time change the second transcriptional terminator of T9002 to avoid repeated sequences (3) (Figure 5a). To do this, we first amplified Part A from the prefix of T9002 to upstream of the GFP-coding sequence with the B0032 junction sequence (Figure 5b). Next the B0032 and kanR-coding sequence was amplified for Part B, having the Part A and C junction sequences on either side (Figure 5b). The B0011 terminator was then amplified with the vector back to the prefix as Part C, having Part A and B junction sequences on either side (Figure 5b).

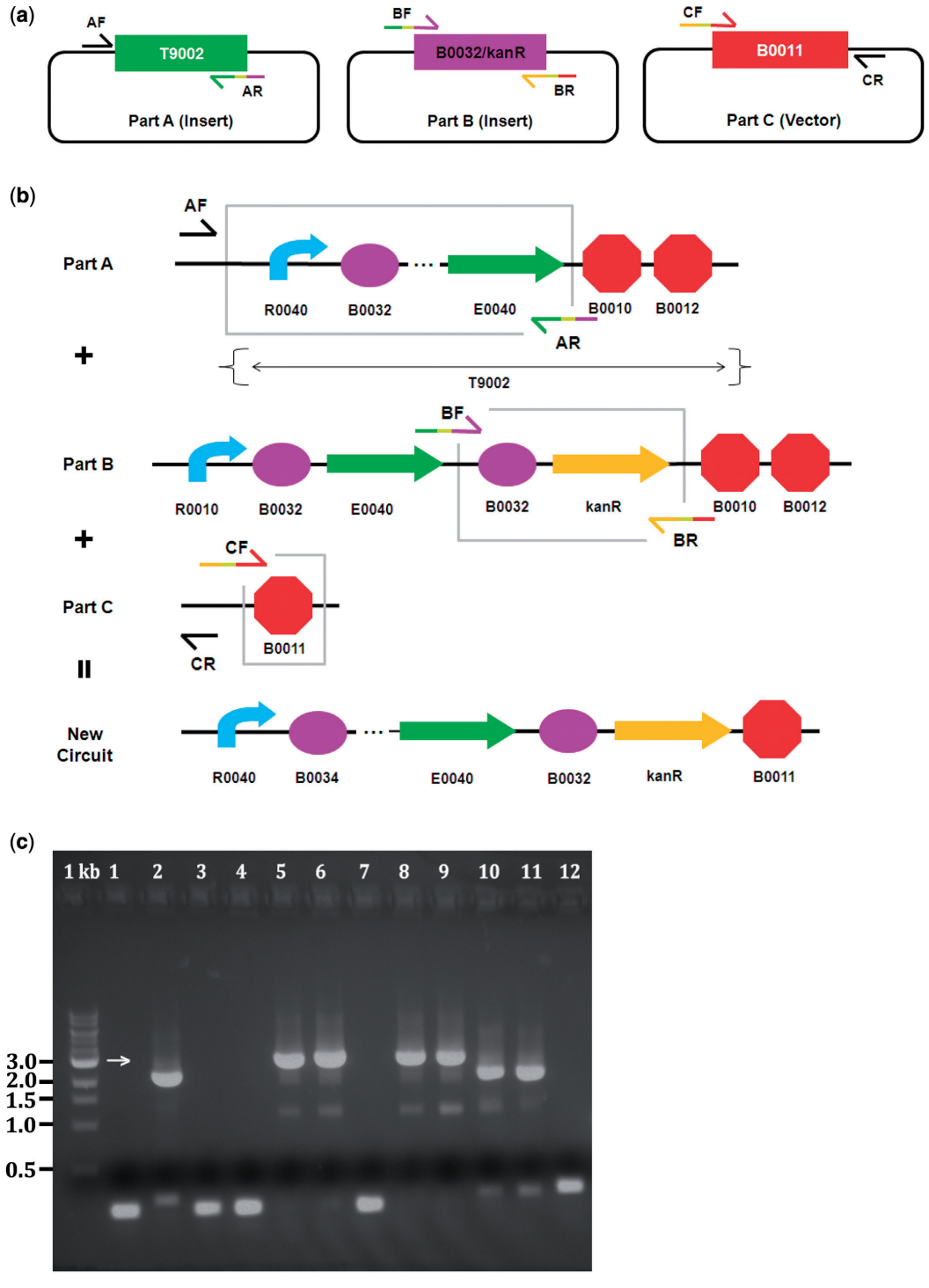


Figure 5. Insertion of an antibiotic resistance gene and terminator swapping through a three-way In-Fusion Assembly. (**a**) T9002, B0032/kanR, and B0011 plasmids are shown with the forward and reverse primers for PCR. B0011 is amplified with the vector and both T9002 and B0032/kanR are amplified as inserts. (**b**) Detailed schematic of the assembly strategy with the forward and reverse primers. The entire T9002 circuit is amplified upstream of B0010 (Part A) with homology to connect an RBS and kanR gene downstream (Part B). To avoid the use of repeated transcriptional terminators (as in T9002 but not shown in the figure), a B0011 terminator (Part C) was placed downstream of kanR. (**c**) Colony PCR results show that 4 out of 12 colonies were positive for the correctly assembled construct (correct size indicated by arrow). A negative control was not necessary since only a ~3-kb fragment would indicate a successful colony.

Colony PCR results showed that 4 out of 12 (33%) colonies had the correct construct and three were verified by sequencing (Figure 5c).

Three-way assembly: creation of a fusion protein

In this assembly, a luxR-GFP fusion protein is incorporated in the middle of the I731014 transcriptional cascade (Figure 6a and b). As in the previous example, an otherwise complex construction using Standard Assembly is efficiently streamlined using the In-Fusion cloning method. The I731014 circuit comprises the backbone assembly component. The GFP-coding sequence and a terminator comprise one of the insert components. The final component consists of a linker sequence amplified from a non-Biobrick vector (pBAD/HisA, Invitrogen).

To determine the effect of the vector:inserts ratio for a three-way assembly, we tested 1:2:2, 1:5:5 and 1:10:10 ratios. A 1:2:2 ratio resulted in 0/24 correct constructs, a 1:5:5 ratio resulted in 4 out of 24 (16.7%) successes, and 1:10:10 ratio resulted in 9 out of 24 (37.5%) correct constructs (Figure 6c). While three-way assemblies generally exhibit a lower success rate compared with two-way assemblies, optimizing the vector:inserts ratio improves the odds of success. Furthermore, this example demonstrates that assembly components representing a wide range of sizes (3.8 kb, 0.8 kb and 0.2 kb) may be simultaneously assembled in a single assembly step. In this example, the minimum 15 bp of homology was used. It is likely that the success rate of three-way assemblies could be further improved by increasing the length of homology at the junctions.

DISCUSSION

There are currently several assembly methods used to construct plasmids for synthetic biology research or other applications. In the synthetic biology community, Standard Assembly is the most widely used method despite competing standards. Custom DNA synthesis is still too expensive for most synthetic biology labs to perform routine constructs, but is ideal when constructing DNA from scratch when there is no template DNA available for PCR or when there are many assembly reactions to perform (14). Overlap extension PCR methods (11,12) are also useful to construct novel DNA sequences without a template, but can be somewhat expensive depending on the length of the construct. For many synthetic biologists, the enormous number of parts available in the Registry allows for diverse circuits to be constructed without the use of DNA synthesis or overlap extension PCR methods.

In our experience, there are three major advantages to In-Fusion assembly over Standard Assembly. First, In-Fusion provides a flexible method to perform large-scale assemblies by mixing and matching parts from the Registry. Second, this method is faster since gel extraction is unnecessary, there are fewer experimental steps, and fewer reactions to optimize, allowing for many reactions to be performed in parallel. Third, we find the success rate for In-Fusion to be high and consistently are able to engineer the desired construct. Our results indicate that the In-Fusion success rate increases with the homology length without the cost of frequent mutations in assembled plasmids which we found to be rare. More homology between assembly components requires longer primer lengths, increasing the primer cost slightly, but the corresponding increase in success rate may justify the extra cost when performing difficult assemblies. We found that in general two-way In-Fusion assemblies have >60% success rate, so in this case using the minimal homology of 15 bp may be sufficient. However, we found that three-way assemblies have lower and more variable success rates (lower than 40% in the examples shown here) depending on the construct. For these more difficult assembly reactions, having more homology will maximize the chances for success and outweigh the extra primer costs. We also did not find it necessary to purchase expensive purified primers because the mutation rate is low enough for it to not be an issue. Although we didn't perform a systematic analysis, there's no obvious relationship between success rate and PCR product size, consistent with (6), but in some special cases optimizing the insert:vector ratio may be necessary.

In regard to assembly standards, ideally an assembly standard will use the same laboratory components (e.g. restriction enzymes and ligase) so that the same components can be used to assemble any two (or more) BioBricks together. This In-Fusion BioBrick assembly method in its current state cannot be completely standardized because custom primers need to be ordered for each individual assembly and hence different components are required. However, the assembly method described here can be semi-standardized by two simple primer design rules (described in the 'Materials and Methods' section) that allow for much of the planning to be removed from primer design. The primer design tool we built will also decrease the time it takes to order primers and maximize success with In-Fusion assemblies. It would be possible to expand this method to use the same standard vector primers for every assembly reaction, but doing so would introduce large scar sequences between parts due to the minimum amount of homology required between PCR-amplified BioBricks. These large scar sequences would most definitely cause problems for spacing between different parts [e.g. the RBS and coding sequences (15,16), unless these sequences were already together on one part and properly spaced].

In conclusion, we have optimized the In-Fusion assembly protocol and adapted this method for BioBrick Assembly and re-engineering. We used this method to make several diverse constructs and simplified the number of experimental steps as much as possible, as illustrated by the six examples above. The cost per reaction of In-Fusion assemblies is relatively high compared with Standard Assembly, but the consistent success of our diverse assemblies and the elimination of intermediate assembly steps in complex constructions make the cost worthwhile in our experience. We hope this method can be expanded upon in the future to fuse a large number of fragments together (17,18) and be standardized to use the same laboratory components.

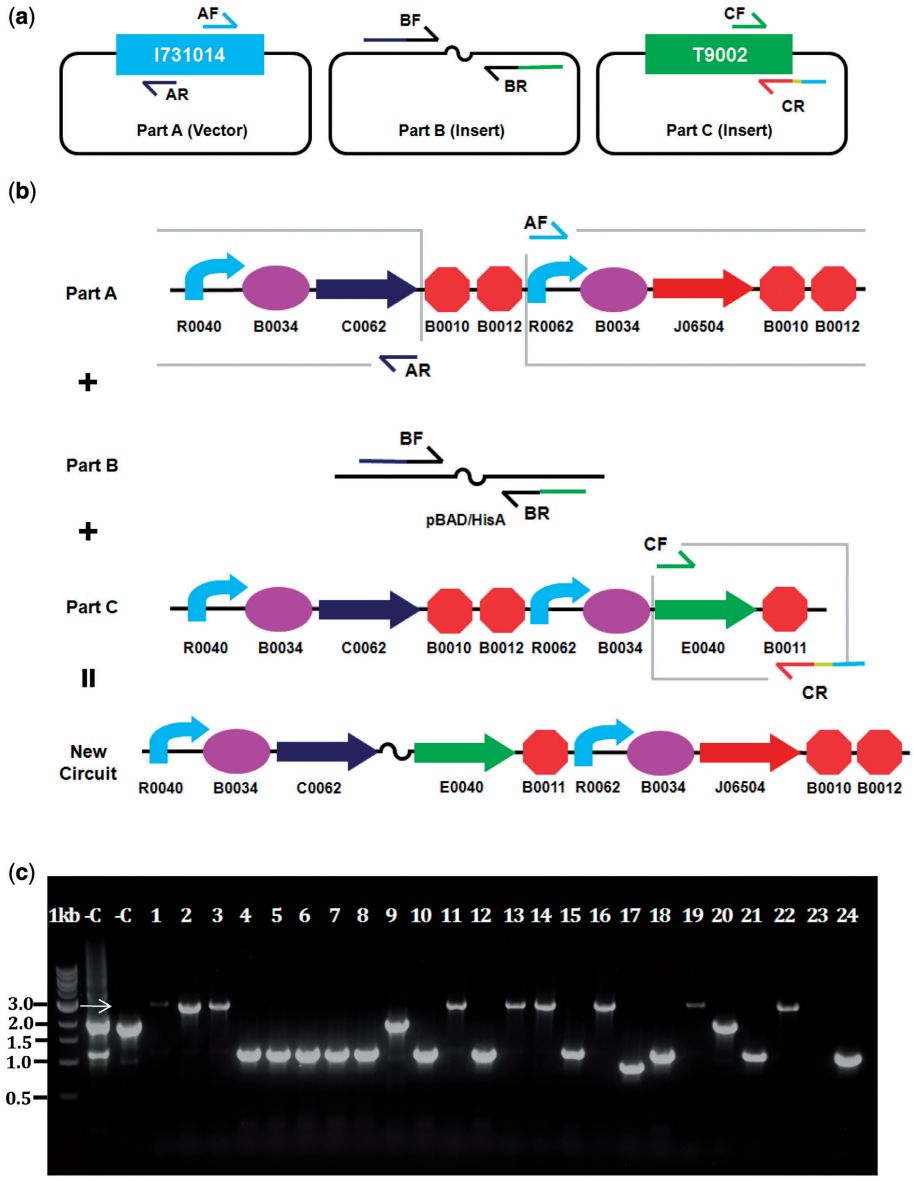


Figure 6. Simultaneous construction of a fusion protein and insertion into a transcriptional cascade via three-way assembly. (**a**) I731014 is analogous to the T9002 transcriptional cascade, but uses the mCherry reporter rather than GFP. GFP was inserted downstream of luxR with a long linker component incorporated between the two coding sequences. (**b**) Detailed schematic showing regions of homology incorporated on 5′ tails of primers. The linker component was amplified from a commercial cloning vector and included several epitope tags. (**c**) A 1:10:10 vector:insert:insert ratio was optimal for this three-way assembly, resulting in a 9 out of 24 success rate. The success rate was evaluated by colony PCR using VF2/VR primers. Negative control reactions were performed on the original I731014 and T9002-F templates. Background from the pBAD/HisA vector was not a factor because this vector has a different selection marker than the final construct. The desired construct exhibits a VF2/VR amplicon of 2.9 kb and a noticeable mobility shift compared to the negative controls (correct size indicated by arrow).

SUPPLEMENTARY DATA

Supplementary Data are available at NAR Online.

ACKNOWLEDGEMENTS

The authors thank Deepak Chandran for creating the primer design software tool and to Frank Bergmann for putting this tool online. We also would like to thank members of the Sauro and Klavins synthetic biology groups at the University of Washington for materials and useful discussions.

FUNDING

Sauro Lab start-up budget (University of Washington) and National Science Foundation (NSF) Grant in Theoretical

Biology (0827592). Funding for open access charge: Sauro Lab start-up funds.

Conflict of interest statement. None declared.

REFERENCES

1. Knight,T.F.Jr (2003) Idempotent Vector Design for Standard Assembly of Biobricks. DSpace@MIT.
2. Shetty,R.P., Endy,D. and Knight,T.F. Jr (2008) Engineering BioBrick vectors from BioBrick parts. *J. Biol. Eng*., **2**, 5.
3. Canton,B., Labno,A. and Endy,D. (2008) Refinement and standardization of synthetic biological parts and devices. *Nat. Biotechnol*., **26**, 787–793.
4. Kelly,J.R., Rubin,A.J., Davis,J.H., Ajo-Franklin,C.M., Cumbers,J., Czar,M.J., de Mora,K., Glieberman,A.L., Monie,D.D. and Endy,D. (2009) Measuring the activity of BioBrick promoters using an in vivo reference standard. *J. Biol. Eng*., **3**, 4.
5. Anderson,J.C., Dueber,J.E., Leguia,M., Wu,G.C., Goler,J.A., Arkin,A.P. and Keasling,J.D. (2010) BglBricks: a flexible standard for biological part assembly. *J. Biol. Eng*., **4**, 1.
6. Zhu,B., Cai,G., Hall,E.O. and Freeman,G.J. (2007) In-Fusion Assembly: seamless engineering of multidomain fusion proteins, modular vectors, and mutations. *Biotechniques*, **43**, 354–359.
7. Benoit,R.M., Wilhelm,R.N., Scherer-Becker,D. and Ostermeier,C. (2006) An improved method for fast, robust, and seamless integration of DNA fragments into multiple plasmids. *Protein Expr. Purif*., **45**, 66–71.
8. Li,M.Z. and Elledge,S.J. (2007) Harnessing homologous recombination in vitro to generate recombinant DNA via SLIC. *Nat. Methods*, **4**, 251–256.
9. Gibson,D.G., Young,L., Chuang,R., Venter,J.C., Hutchison,C.A. III and Smith,H.O. (2009) Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nat. Methods*, **5**, 343–345.
10. Geu-Flores,F., Nour-Eldin,H.H., Nielsen,M.T. and Halkier,B.A. (2007) USER fusion: a rapid and efficient method for simultaneous fusion and cloning of multiple PCR products. *Nucleic Acids Res*., **35**, e55.
11. Gibson,D.G. (2009) Synthesis of DNA fragments in yeast by one-step assembly of overlapping oligonucleotides. *Nucleic Acids Res*., **37**, 6984–6990.
12. Horton,R.M., Cai,Z.L., Ho,S.N. and Pease,L.R. (1990) Gene splicing by overlap extension: tailor-made genes using the polymerase chain reaction. *Biotechniques*, **8**, 528–535.
13. Sleight,S.C. (2009) BBF RFC 26: In-Fusion BioBrick Assembly. DSpace@MIT.
14. Bayer,T.S., Widmaier,D.M., Temme,K., Mirsky,E.A., Santi,D.V. and Voigt,C.A. (2009) Synthesis of methyl halides from biomass using engineered microbes. *J. Am. Chem. Soc*., **131**, 6508–6515.
15. de Smit,M.H. and van Duin,J. (1990) Secondary structure of the ribosome binding site determines translational efficiency: a quantitative analysis. *Proc. Natl Acad. Sci. USA*, **87**, 7668–7672.
16. Pfleger,B.F., Fawzi,N.J. and Keasling,J.D. (2005) Optimization of DsRed production in Escherichia coli: effect of ribosome binding site sequestration on translation efficiency. *Biotechnol. Bioeng*., **92**, 553–558.
17. Engler,C., Gruetzner,R., Kandzia,R. and Marillonnet,S. (2009) Golden gate shuffling: a one-pot DNA shuffling method based on type IIs restriction enzymes. *PLoS ONE*, **4**, e5553.
18. Villiers,B.R., Stein,V. and Hollfelder,F. (2009) USER friendly DNA recombination (USERec): a simple and flexible near homology-independent method for gene library construction. *Protein Eng. Des. Sel*., **23**, 1–8.

3.3 Quality Control for Large-scale DNA Assembly

Three Best Practices for Quality Control of Large-scale DNA Assembly

Bryan A. Bartley[1], Michal Galdzicki[2], John H. Gennari[3], Herbert M. Sauro[1]
[1]University of Washington, Department of Bioengineering, Seattle, WA 98195
[2]Arzeda Corp. 2715 W Fort St Seattle, WA 98199
[3]University of Washington, Biomedical & Health Informatics, Seattle, WA 9819

ABSTRACT

An aspirational goal of synthetic biology is the creation of new living systems by large-scale, combinatorial assembly of DNA. We claim that new ways of thinking about quality control (QC) can help synthetic biologists achieve this vision. In this paper we discuss the pros and cons of current QC practices, and propose three new best practices for QC that will help synthetic biologists build bigger and better. We extend the paradigm of bio-design automation for synthetic biology to include Failure Analysis, an engineering practice with historical roots in the development of military, aerospace, automotive, and electronic technologies. We leverage an openly exchanged data standard for annotating defects in a fabricated DNA construct and analyzing failure modes. Standardized representation of QC data is critical for synthetic biologists to trust each other's work and build off of it. Standardized failure analysis closes the loop in an automated design cycle, as lessons from failure feedback to inform new design strategies. We are prototyping automated software tools that demonstrate new ways of visualizing and thinking about QC for large-scale DNA assembly.

DISCUSSION

Synthetic biology is an interdisciplinary endeavor that requires collaboration between experts in different fields and multiple iterations through the "design, build, and test" cycle. Many synthetic biology projects are complicated engineering projects dogged by frequent failure and obstacles to reproducibility. Although many synthetic biology groups have highlighted these challenges [1]–[3], the field still lacks sufficient methods, tools, and standards for solving problems collaboratively over great distances, managing failure, and reproducing scientific results. We claim that new ways of thinking about QC are needed to help synthetic biologists engineer more robust, more sophisticated, and more interesting genetic systems. In this paper we discuss the pros and cons of current QC practices, propose three new best practices, and demonstrate the use of these best practices to debug failures in genetic circuits.

One QC best practice we demonstrate here is Failure Analysis. Failure Analysis is a systematic approach to documenting patterns of failure in system components. Originally developed by the military, it was soon adopted by NASA to manage failure in the Apollo missions and missions thereafter. From NASA it

spread to civil aviation, automotive, and electronic industries, and is now established engineering practice[4]–[7]. Though actual practices vary from field to field, Failure Analysis typically involves standardized qualitative descriptions as well as quantitative metrics, such as probability of failure.

Previously, we tested the long-term performance of several genetic constructs made from well-characterized DNA parts, documenting certain patterns of failure, or failure modes as they are called in formal Failure Analysis[8]. One quite general failure mode occurs when recombinant processes delete bits of nucleotides from in between homologous sequences. From this observation a simple design rule may be formulated, *i.e.*, to avoid homologous sequences in a design. Unfortunately, failure analysis is currently too labor-intensive to become standard operating procedure in laboratories, making it difficult for synthetic biologists to study failure modes and formulate design rules. This experience led us to develop an automated approach, so that the synthetic biologist can collect, document, and analyze data about failure modes on a large number of constructs as a matter of routine.

Failure analysis is now one of many standardized, computer-aided engineering tasks applied to great success in conventional engineering fields[6], [7], [9]. The bio-design community takes its inspiration from these successes and aims to apply similar paradigms to biological engineering. As far as we know, ours is the first example of computer-aided failure analysis applied to synthetic biology.

In order to standardize and automate Failure Analysis we leveraged the Synthetic Biology Open Language (SBOL)[10], an open data exchange standard. Previously the SBOL Development Group demonstrated design and manufacture of a synthetic gene circuit following a standardized, automated workflow. Our contribution extends the standardized, automated workflow for synthetic biology from design and manufacturing to quality control. Conclusions about the reliability of design components can be easily shared with downstream builders who must rely on properly verified upstream constructs. Moreover, Failure Analysis helps builders learn from other builders' prior experience. Thus QC information feeds back into the design cycle, allowing synthetic biologists to learn from failure and improve their designs.

Failure in DNA assembly is inevitable. The recombinant DNA methods used in synthetic biology (e*g*, PCR, Gibson assembly) are derived from the very same, fundamental biochemistry that drives evolution and biological variation. Recombinant engineering of DNA is inherently difficult to control and predict precisely, as most anyone who has attempted it will attest. In addition human error is also a factor in failed assemblies, but even with a fully automated construction pipeline, unknown and unexpected mutations are bound to generate biological variation in the assembly process. For this reason, it is

customary to screen multiple instances of a DNA construct in order to isolate a clone that matches the target design, a QC procedure called sequence verification[11].

Sequence verification traditionally requires the expert eye of a biologist but nowadays verification is often automated for efficiency. Several automated platforms for sequence verification are available[11], [12], and many commercial DNA synthesis services have in-house platforms for quality control. Still, by one estimate 50% of gene synthesis costs are spent trying to isolate perfect clones[13], and based on our own experience sequence verification is still a significant bottleneck in the synthetic biology workflow, especially for small research organizations which lack specialized support for bioinformatics.

Although DNA assembly and DNA synthesis are often confused terms, there is an important difference. Assembly differs from synthesis in terms of both scale and complexity of the target constructs that may be achieved. The upper limit in size for *de novo* gene synthesis is about 10^5 bases with a best-case error frequency of $1/10^5$ bases [14]. Constructs larger than this must be assembled from DNA fragments via recombinant methods, although the fragments themselves may be purely synthetic, of course. Thus large-scale constructs above 10^5 bp exceed a critical limit at which new quality control issues related to recombinant assembly may begin to dominate. Even if the synthetic components are individually sequence-verified, assembly may introduce new errors. Recombinant assembly implies a parts-based, hierarchical approach to assembling units of biological function, but current QC methods provide neither functional nor hierarchical reasoning about failure. These issues have not yet received much attention, perhaps because most genetic circuits featured in literature are still relatively primitive in terms of biological complexity and do not exceed this critical limit. Recently, however, Temme, et al. refactored a 23.5 kb nitrogen fixation gene cluster in *Klebsiella oxytoca* using only synthetic, well-characterized parts, concluding that better ways to diagnose sequencing errors in genetic circuits are needed[15]. As a practical matter, failures often occur for relatively simple recombinant constructs even at typical assembly scales of less than 10 kb, which only strengthens the case for better QC methods.

Current QC approaches for DNA synthesis provide little information to the synthetic biologist to help debug genetic circuits or guide higher-order steps of large-scale assembly. For examples of current QC practices, refer to Box 1. The tools featured each have their advantages but also demonstrate certain limitations which hinder reasoning over large-scale genetic designs.

For example, automated sequence verification platforms typically takes a pass or fail approach. Illustrating this is CloneQC [12] and GenoREAD[11] (see Box 1). The pass or fail approach makes sense for commercial DNA construction services whose objective is to sell DNA, but *caveat emptor*! The synthetic circuit may not function as you originally designed, or it may mutate once introduced into its

production host. In either case, the builder may need to diagnose the failure, adjust the assembly strategy, or completely redesign the construct. Pass or fail criteria provide no hint to guide this decision-making. We hypothesize there may be important clues in the sequencing data that simple pass or fail criteria do not capture.

Pass or fail criteria used by programs like CloneQC and GenoREAD reflect a very goal-oriented approach to synthetic biology, perhaps reflecting very real economic factors driving both commercial DNA synthesis and academic research. Given that failures of gene circuit construction and function are so common, even ubiquitous, there is a danger that valuable information is being discarded to great detriment of both scientific inquiry and reproducibility.

Our previous work documented failures in a T9002 biobrick from the iGEM parts registry[8]. Even in small sample sizes of a few failed clones, distinct and interesting patterns began to emerge. We suspect that other synthetic biologists may intuit similar patterns in their own sequencing data, but lack quantitative tools to investigate and describe them. These failures should not simply be ignored, because there is potentially useful information to be gleaned in them. Ignoring these lessons from failure simply amounts to a waste of valuable research dollars. Rather, failure data should be collected and routinely analyzed during the screening process. However, conducting failure analysis by manual sequence verification is currently too labor-intensive to become standard practice. This led us to automate the process using Python scripts and encode the data in the SBOL data exchange format (see Methods).

Automated failure analysis can help the synthetic biologist rapidly accumulate statistically meaningful datasets about patterns of failure. The failure metrics we propose are objective criteria which synthetic biologists may use to evaluate design rules for DNA assembly. As others have proposed, design rules may formalize manufacturing expertise and in the long-run reduce manufacturing costs[16].

Biological parts repositories, such as the iGEM (parts.igem.org) registry and Joint BioEnergy Institute's Inventory of Composable Elements (JBEI-ICE)[17], serve as important community resources where synthetic biologists can obtain physical samples of DNA and important data about those samples. Sharing and re-use of DNA parts through repositories critically depends on good QC data. See Box 1 for examples of QC tools provided by the iGEM registry and JBEI-ICE.

The iGEM and JBEI-ICE repositories have excellent tools for visualizing and interpreting DNA sequence alignments for typical constructs, but these tools may prove troublesome for interpreting large-scale, contiguous assemblies of sequencing reads (contigs). One problem is scalability. For example when one tries to visualize a large plasmid in ICE's VectorEditor, or any other sequence editor for that matter, sequence features and alignments are rendered too small to interpret comfortably by eye. Long designs

aligned with large contigs are rare currently. A recent search of the JBEI-ICE repository (accessed 10/16/15) found 44 constructs larger than 10 kb with aligned sequencing data. But edge cases like these will inevitably become more common as synthetic biology grows more ambitious. Another problem is that none of the QC tools described in Box 1 provide an objective, quantifiable measure of confidence in any single unit of biological function. Consequently, the biologist cannot explain observations about phenotype in terms of genotype except in the fuzziest of terms. In other words it is impossible to debug the genetic program.

The example in Box 1 of a sequence verification report for biobrick T9002 hides a more insidious QC problem. The sequencing reads were not trimmed to remove noisy reads. Therefore noisy base calls appear as errors in the sequence of the physical sample, although there is insufficient evidence to draw this conclusion. In fact what the long part needs is another sequence read to verify the noisy base calls. Contrary to what the QC report indicates, the T9002 construct is one of the most trusted in the iGEM parts distributions, indicated by its one star rating.

The QC best practices we discuss here are intended to support scientific discovery as much as bio-design automation. The QC approaches we discuss here have applications for experimental evolution, because they allow the biologist to explore correlations between genotype and phenotype. Experimental evolution is an important trend in synthetic biology research and many groups are experimenting with directed evolution [18], [19] as a design strategy. In these cases the paradigm of bio-design automation is turned upside down. Designs are not forward engineered, rather evolution selects the design. Such studies could benefit from standardized tools that allow the synthetic biologist to easily reconstruct fitness landscapes. Visualization tools that support correlated views of construction and function (see Best Practice 1) may be useful to the biologist studying evolutionary trajectories, epistatic interactions, and biological modularity.

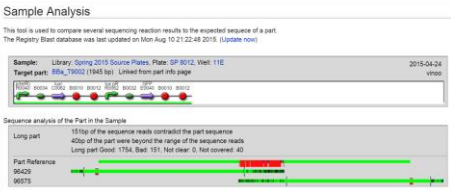
The QC data we discuss here is based on Sanger sequencing reads. However much of this sequence verification workflow applies to Next Generation Sequencing (NGS) methods as well. One aspect of NGS data that is not efficiently addressed by our annotation methodology is the representation of sequencing depth. The annotation approach we describe here would be inefficient for capturing base by base measurements like sequencing depth. Sequencing depth and Phred QC scores both require a unique value for each nucleotide base and are perhaps more easily described using numerical arrays rather than annotations. Unfortunately, the SBOL 2.0 data model does not currently support representation of numerical arrays.

BOX 1: PROS & CONS OF CURRENT QC PRACTICE

iGEM Registry

Pro: Shows the functional composition of the design

Con: Can't diagnose failure. Which part is broken? What type of error occurred?



CloneQC

Pro: Easy pass/fail assessment for DNA construction services.

Con: Can't diagnose reasons for failure. Effort and information are wasted.

STATISTICS

Target	# Passes	# Checks	# Fixables	# Fails	Total
9L3_32.Y3.07	1	0	0	17	18
9L3_32.Y3.08	1	0	0	3	4
9L3_32.Y3.09	1	0	0	1	2
9L3_32.X3.02	1	1	0	15	17
9L3_32.Y3.01	0	1	0	20	21
9L3_23.C1.09	0	0	19	1	20
9L3_32.Y2.06	2	2	0	8	12

9L3_32.Y3.07

Passing Clones	Check Clones	Fixable Clones
BAG2008F_3_17_F12		

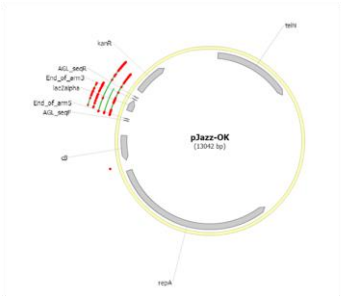
9L3_32.Y3.08

Passing Clones	Check Clones	Fixable Clones
BAG2008F_3_17_E07		

JBEI-ICE

Pro: Sequencing coverage and depth are easy to assess at a glance

Con: Large-scale constructs are difficult to interpret



RESULTS

In order to demonstrate and illustrate the QC best practices we are advocating, we conducted retrospective QC analyses on failed assemblies from previous projects described elsewhere[8], [20]. This extends our work in which we tested the evolutionary stability of circuit variants over time and cataloged mutations as they appeared in a population. In this work, we identify modes of failure before propagation. Our results show that interesting patterns of failure may be detected as early as the screening process, before a DNA assembly is propagated in its host, from which hypotheses about the cause of failure can be deduced and mitigating actions can be planned.

Best Practice 1: Support Correlated Views of Construction & Function

We recommend reporting QC metrics in a correlated view with a functional diagram of the construct (Figure 1). In essence, we propose a novel way of visualizing sequence alignment data. Inspecting an alignment base by base is comparable to examining machine code bit by bit, while our schema allows one to examine QC data from a comfortable level of abstraction. A correlated view of construction and

function allows the builder to quickly diagnose functional failures due to construction errors and mutations. A correlated view of construction and function may lead to the discovery of new genetic design rules, because it helps the synthetic biologist relate observations about phenotype back to genotype[16].

To illustrate this, we use SBOL Visual (Accepted, PLOS Biology) symbols to represent the biological function associated with each genetic part of the T9002 biobrick from the iGEM registry. This construct is derived from well-understood components, the cell-to-cell communication receiver F2620[21] and green fluorescent protein generator E0240. Symbols include promoters, ribosome binding sites, coding sequences, transcriptional terminators, and assembly scars. We also use some nonstandard symbols. The arc represents a regulatory link from the LuxR activator protein to its target promoter. The red X's indicate disabled parts. The QC statistics, based on alignment against the target sequence, are displayed below each part.

The specific sequence alignment metrics we recommend are percentages of identity, error, ambiguity, and unsequenced. These four categories represent different degrees of sequence verification and together they add up to a hundred percent. Identity quantifies the proportion of verified bases in a construct compared to its design sequence, while error quantifies the proportion of a clone's sequence which does not match. Ambiguity is important because noisy sequencing data results in uncertainty about the clone's true sequence. Unsequenced is an important metric for any construct of non-trivial length, because the construct may be incompletely sequenced. Cost is often a factor in deciding if a construct is completely sequenced. For example, sometimes it makes economic sense to spot-check a clone's sequence before spending money to fully sequence the entire construct.

Using a correlated view of construction and function, the engineer can predict how the circuit will behave given sequencing errors. In this example, the promoter and several adjacent parts have 100% error and a regulatory link has been broken. The circuit should produce the LuxR regulator but there will be no fluorescence exhibited by the cells. In this example, the predicted behavior is rather trivial, but the performance of large-scale, logic circuits could exhibit very perplexing behavior given the failure of a particular part. A correlated view of construction and function provides one way to debug a genetic program. Our example contrasts with the T9002 example from the iGEM registry (Box 1) which provides no insight into which parts are affected.

This example shows failure of specific components that comprise a mutant of T9002. As previously reported[8], the T9002 construct and variants derived from it exhibit a variety of mutational failures, including point mutations, recombination errors, and deletions affecting certain genetic parts more frequently than others. One frequently observed mutation involved disabling mutations of the R0062 promoter. Although the mutant construct in Fig 1 was isolated in a different experiment, it exhibits a similar failure of the promoter and adjacent components as described in our previous failure analysis. This recurring pattern suggests that certain units of genetic function, such as the R0062 promoter, may fail more frequently owing to strong negative selection pressure. We suggest that correlated views of construction and function can make it easier for the engineer to reason about genetic function and recognize possible patterns of failure.

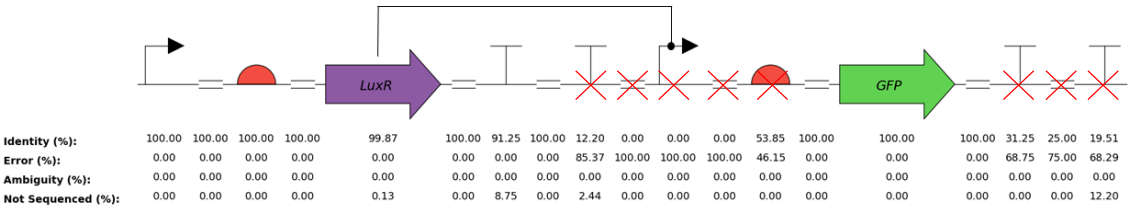


Fig 1. A mutant of T9002. Mutations have disabled the promoter bound by LuxR, breaking the regulatory link. Correlated views of construction and function help the biologist to relate genotype to phenotype.

Patterns of failure imply a non-random distribution of errors. In order to show patterns of failure, a statistical analysis of failures across many clones must be conducted. Our approach to automated classification and annotation of sequence variants, described in the Methods section, enables statistical analysis of assembly errors and mutations. See Best Practice 3 for a demonstration of failure analysis.

Best Practice 2: Support Hierarchical Reasoning About QC

As the synthesis of minimal genomes becomes more routine in synthetic biology, it will be necessary for synthetic biologists to navigate QC information at different levels of genetic hierarchies. The functional organization of large-scale DNA constructs, from plasmids to genomes, cannot be easily understood using sequence editors. Our approach of encoding QC data in SBOL enables reasoning over hierarchies of genetic organization and hierarchical navigation through the user-interface.

Unfortunately the SBOL Visual standard set of genetic glyphs (version 1.0) lags behind the data model (version 2.0) and does not express all of the design principles contained in the new data model. For example, currently SBOL Visual lacks an explicit symbol for representing high-levels of modular abstractions. We would like to introduce the convention of using the generic, user-defined symbol underscored by a triangle to indicate a hierarchical abstraction of lower-level components. As a corollary, a triangle underscoring a typed SBOL visual symbol indicates a typed module. For example, a double

terminator may be represented by a single terminator glyph underscored with a triangle that abstracts two single terminators at a lower level of composition.

The T9002 construct is a basic example of hierarchical DNA construct. At the highest level of organization, the T9002 construct is joined with a pSB1A2 vector backbone which encodes replication and selection functions (not shown). Vector backbones are often assumed to be a relatively inert structural feature of a design and are not a priority for QC. In this example the backbone is unsequenced. The assumption that backbones are inert may not be justified, however, as a variety of mutations, such as copy number variations may affect function. Inside the T9002 construct are two devices, the cell-to-cell communication receiver F2620[21] and green fluorescent protein generator E0240. Further down the hierarchy are the atomic components of the design, promoters, ribosome binding sites, coding sequences, transcriptional terminators, and assembly scars. If we wished, we could annotate even lower levels of biological detail, like operators within a promoter or codons translating to the chromophore in a fluorescent protein in our design.

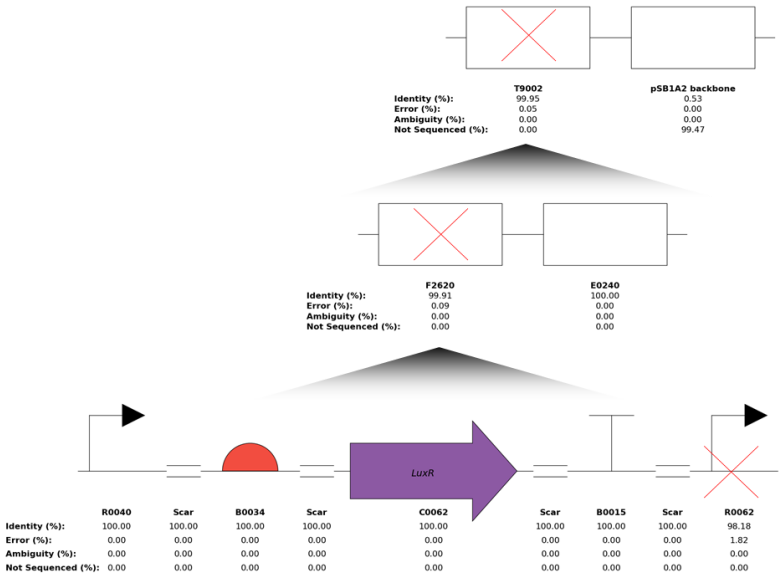


Fig 2. A hierarchical representation of a genetic design can be easily navigated to localize mutations in specific modules. As genetic designs become more complex, abstractions such as this may be helpful for navigating large DNA sequences.

This real example of a T9002 failure was disabled by a point mutation. Using our visualization scheme the failure can quickly be localized starting at a high level of abstraction to a point mutation in the R0062 (pluxR) promoter. In principle, this approach should be useful for navigating QC data at any scale of genetic organization, starting from the genome level and drilling down through systems, operons, genes, etc. As DNA assemblies scale in both structural and functional complexity, tools that support this QC best practice will become increasingly important.

Best Practice 3: Learn from Failure

Recently, we constructed a variant library of inducible reporter constructs that express yellow fluorescent protein (YFP) at different levels. The RBS variants used in the library were previously shown to modulate protein expression in a graded manner using simple sequence repeats (SSRs) embedded between the RBS and the start codon of the downstream gene. At the time, clones were verified using conventional tools for alignment and sequence editing. The general conclusions from each sequence analysis were written in a notebook, including informal descriptions of the sequence errors observed. Anecdotal evidence of patterns of failure began to emerge through this process. So we retrospectively analyzed 28 failed constructs which had been screened out from further study.

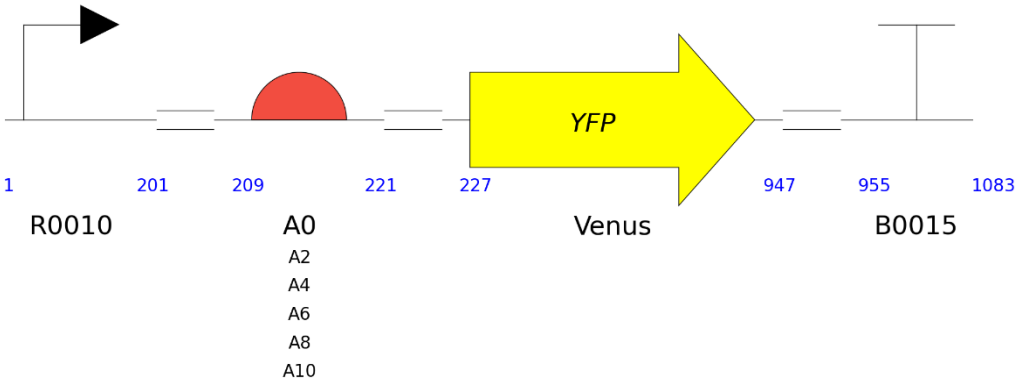


Fig. 3 These designs that comprise our library consist of biobrick R0010 (the canonical plac promoter), several RBS variants, the Venus variant of yellow fluorescent protein (YFP), and the B0015 terminator biobrick. Variants of the RBS differed by repeats of an adenine base. For example, A0 is the original RBS, while A10 has 10 repeating adenines. Start and end coordinates of the components are indicated in blue. The library was assembled by two-way Gibson assembly and transformed into DH5α Turbo cells (New England BioLabs, Ipswich, MA)

As explained in Methods, our QC approach involves automated classification of discrete mutations. From these data we created a failure report that shows the distribution of errors in each component of the target design. This failure report is collectively pooled from 5 assemblies conducted at separate times. The main factor that varied in each assembly was the substrate for the RBS variant. Otherwise, the assemblies followed the same protocol.

Table 1. PART-WISE FAILURE DISTRIBUTION (n = 28)

	R0010	Scar	RBS Variant	Scar	YFP Venus	Scar	B0015
INSERTIONS	0	0	1	0	90	3	4
DELETIONS	4	1	1	1	153	2	19
TOTAL	4	1	2	1	243	5	23

Based on this distribution, sequence errors do not appear randomly distributed among each unit of genetic function. There are a striking number of mutations in both the Venus YFP coding sequence and the B0015 terminator. In this example, we will focus on the patterns of failure in the coding sequence, since failure patterns in terminators were previously described (REF).

Errors in a DNA construct may be introduced at different steps in the assembly process. For example, errors may already be present in the input substrates, such as primers or PCR products. Often assembly errors occur at junctions where substrates are joined, due to the imperfect nature of recombinant biochemistry. However, even assuming the assembly process were perfect, errors may occur after introduction into a host due to unavoidable, evolutionary processes. A high frequency of errors in a certain genetic component may indicate strong, negative selection for that unit of genetic function. While failures are usually simply screened out and ignored, in this case they may in fact provide a valuable warning to the engineer. Even if "successful" constructs can be identified, they may quickly fail or mutate once deployed for their intended purpose, thus confounding any downstream applications of the circuit. It would be helpful to have this information up front, before expending significant energy to troubleshoot experimental results.

The synthetic biologist can distinguish between these different types of failure using a number of clues, not the least of which is detailed inspection of the nucleotide sequence. Assembly substrates, such as PCR products, may be sequence verified before assembly, as was the case here. Other types of failure, such as those caused by negative selection, might only be identified by looking at statistical patterns in failures in multiple clones. Moreover, as we will demonstrate, the synthetic biologist should sequence background constructs to serve as an experimental control. Background controls distinguish between errors which are inherent to the assembly process versus errors which are inherent to the host, environmental, and other contextual conditions in which a DNA circuit is deployed.

Although background constructs are usually considered a "failure" of assembly, and in many cases are not even sequenced, they can serve as a useful control. Construction background results from carry-over of

template DNA into subsequent steps of an assembly process. Since biochemical reactions are not entirely efficient, some of the DNA assembly substrate does not react. Therefore errors in a background construct are more likely mutations resulting from evolutionary selection rather than errors introduced by the assembly reactions.

Table 2. PART-WISE DISTRIBUTION OF DELETIONS (n = 28)

	R0010	**Scar**	**RBS Variant**	**Scar**	**YFP Venus**	**Scar**	**B0015**
BACKGROUND (n=16)	4	1	1	1	43	1	12
RECOMBINANTS (n=12)	0	0	0	0	110	1	7
TOTAL	4	1	2	1	243	5	23

Failure frequency of the component parts in a background construct provides one statistical measure of selection pressure. Table 2 presents an analysis of the deletion frequency in the same 28 failures presented in Table 1, but this time dividing the failures into background versus recombination errors. This perspective implies that the YFP Venus sequence is selected against by evolution, because even the unassembled background constructs exhibit high deletion rates.

In addition to failure frequency, the distribution of size and location of mutations is also informative. This allows the builder to deduce whether mutations are clustered around a specific base coordinate and how large the area of selective pressure may be. Table 3 shows the distribution of deletions in three background clones we selected for closer study. Given the average size of these mutations, they appear to be point mutations or small deletions. Moreover, they seem to be tightly clustered around approximately the same base coordinate, a mutational "hotspot". This suggests localized selection pressure, and possibly indicates some unknown biological interaction occurring at the corresponding site on the reporter protein.

Table 3. DISTRIBUTION OF DELETIONS IN BACKGROUND CONSTRUCTS

Clone		R0010	Scar	A0	Scar	YFP-Venus	Scar	B0015
1	Frequency	0	0	0	0	3	0	0
	Size (bp)	0	0	0	0	1.3	0	0
	Site (base #)	0	0	0	0	864.5	0	0
2	Frequency	0	0	0	0	2.0	0	0

	Size (bp)	0	0	0	0	4.5	0	0
	Site (base #)	0	0	0	0	870.3	0	0
3	Frequency	0	0	0	0	1.0	0	0
	Size (bp)	0	0	0	0	1.0	0	0
	Site (base #)	0	0	0	0	865.0	0	0

One possibility we considered is that mutations around this site disrupt the chromophore region of the fluorescent protein which is the essential structural motif that provides fluorescence. Fluorescence has a nasty side effect called phototoxicity[22], [23] which damages cells. Consequently, evolution will favor mutants with a broken chromophore, and mutations might be correlated somehow with phototoxicity. To investigate this further, we reviewed known disabling mutations of green fluorescent protein (from which Venus yellow fluorescent protein is derived). The Gly222 amino acid, which corresponds roughly to base coordinate 893 in our design, has previously been shown to participate in chromophore maturation and mutations to this residue can attenuate maturation time[24]. Although the evidence is not conclusive, the mutations we observed in this vicinity could cause frameshifts which disrupt chromophore maturation. However, when the clones described in Table 3 were screened, they had only been propagated under ambient light conditions. This further implies that phototoxicity from Venus may occur in cultures grown under ambient light conditions. Phototoxicity may be compromising experimental results more often than realized, as argued in a recent *Nature Methods* blog[25].

Another failure mode which was *not* detected by the present analysis were frequent mutations of glutamate codons (GAG) to premature stop codons (TAG) at different sites in the Venus mutation. In order to discover this failure mode, we had to refer to the nucleotide level alignments, because we currently do not annotate the effects of codon errors. Our QC practices at present time do not entirely replace the need for sequence level inspection, rather they complement them. However, our automated approach for annotation and standardized representation using Sequence Ontology terms (see Methods) should still work in principle for describing patterns of failure in the translated sequence. The Sequence Ontology includes standardized terms for describing the effect that a DNA mutation will have on its translated sequence, such as *polypeptide_truncation* (SO:0001617). Our next step will be to add this classification ability to SBOL-QC, our Python package for automated QC.

METHODS

The RBS variants described in Fig 2 were constructed using a 2-way Gibson assembly as previously described[20].

Sanger Sequencing was performed by a commercial sequencing service, GENEWIZ (Seattle, WA). Sequencing data consists of chromatograms annotated with Phred quality scores. Noisy sequences were trimmed from the ends of sequence reads using the Geneious sequence editor tool from Biomatters (Auckland, New Zealand). Base calls with greater than 5% chance of miscall are considered too noisy and were trimmed. Sequence files were then exported as text (.seq) files.

The remaining steps of sequence verification were automated using a custom Python (v2.7.9) package we call SBOL-QC (https://github.com/SynBioDex/SBOL-QC). This package provides API methods for aligning multiple sequence reads against a genetic design, provided as an SBOL file. These design files were created programmatically using SBOL-QC. SBOL-QC depends on the Clustal Omega[26] command-line tool (v.1.2.0) and the EMBOSS consensus tool (v6.5.0.0)[27] . The QC methods in SBOL-QC pipe the sequencing data through the command-line tools for multiple sequence alignment and produce a consensus call. The consensus call is the best prediction for the actual sequence of an assembly based on provided data. This consensus sequence is aligned and verified against the target design sequence. Up to now, the process has used only conventional bioinformatics tools. Now the alignment is translated into SBOL.

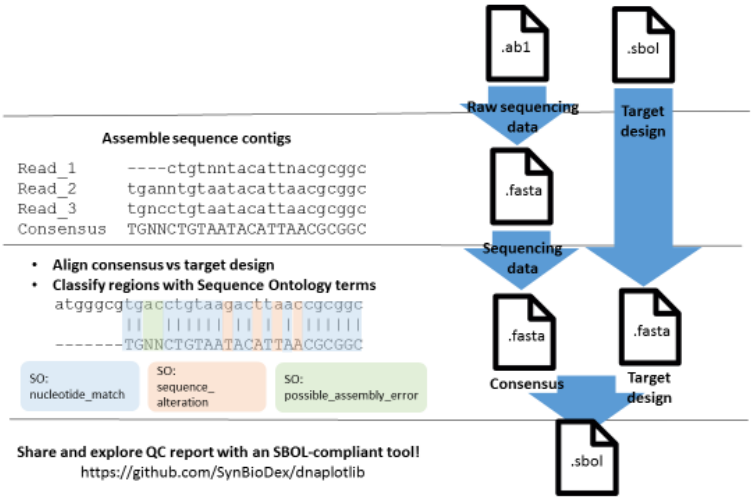


Figure 4. Workflow for automated sequence verification and failure analysis with SBOL-QC

The alignment is translated into SBOL by annotating regions of disagreement between the assembly and target sequences. The usual convention for representing alignments shows nucleotide by nucleotide comparisons. In our SBOL representation, errors are classified into SequenceAnnotation objects indicating the site ranges where insertions, deletions, and substitutions have occurred (see Fig). These annotation objects are classified using a unique term from the Sequence Ontology that identifies the type of error that occurred. Ontologies are standardized vocabularies, and the Sequence Ontology (SO) is one

of many leveraged by SBOL to standardize biological description. Although originally developed for genome annotation, SO provides a rich vocabulary useful for synthetic biology, including annotations for assembly, artifacts from sequencing data, and mutations. Translating an alignment into an SBOL data structure enables the QC best practices discussed previously (see Discussion).

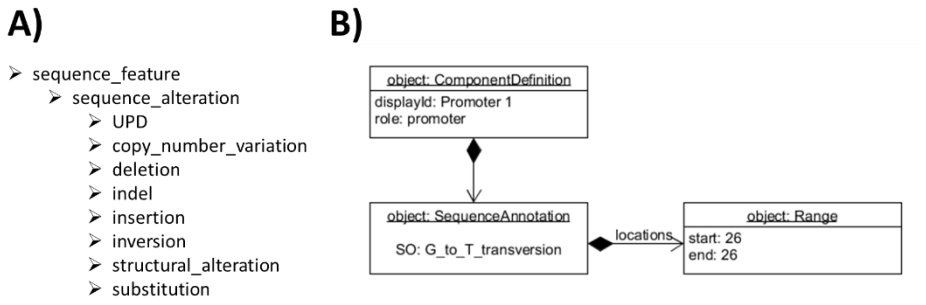


Fig. 5 A) The Sequence Ontology is a standardized vocabulary for annotating DNA sequences with rich terms for mutational variants, assembly junctions, and DNA sequencing artifacts. Additional terms may be subclassified under the each sequence_alteration. For example, transitions and transversions are types of substitutions. B) An example representation of a point mutation encoded using the Synthetic Biology Open Language 2.0 data model.

The API methods in SBOL-QC calculate a variety of QC statistics that describe failure modes on a part by part basis. These include alignment statistics, like percentage identity, error, ambiguity, and unsequenced. They also include failure frequencies, the average size of mutations, and the average location of mutations, metrics which may be helpful for identifying regions that are genetically unstable or biologically interesting.

The SBOL-QC library employs an internal metastandard to make an important semantic distinction between a genetic design versus an instantiated assembly. A design is indicated with the SO term "designed_sequence" (SO:0000546). Once the design is annotated with QC terms, the type is changed to "engineered" (SO: 0000783) and the design's original unique identifier (URI) is changed to indicate an a unique instance of the DNA construct has been manufactured. Finally a new SBOL file with QC is output. This allows us to directly associate QC data with a biological design, which is not possible with current alignment formats and yet another way the SBOL standard supports management of DNA repositories.

The SBOL-QC library also integrates DNAplotlib, a Python package for visualization of SBOL Visual schematics (manuscript in preparation). The QC reports featured here were created with the help of this tool, which can read an SBOL file and render an SBOL Visual schematic from the design.

ACKNOWLEDGMENTS

Current support for the development of the SBOL standard is provided by the NSF award #1355909 for ABI Development of the Synthetic Biology Open Language Resource

The Python visualization tool DNAplotlib is leverage by our SBOL-QC package to make QC reports. DNAplotlib was developed by Thomas Gorochowski (MIT).

Thanks to Ernst Oberortner, Hector Plahar, and Nathan Hillson for assistance with the JBEI-ICE repository.

REFERENCES

[1] J. Peccoud et al., "Essential information for synthetic DNA sequences," *Nat Biotech*, vol. 29, no. 1, p. 22, Jan. 2011.

[2] S. Cardinale and A. P. Arkin, "Contextualizing context for synthetic biology – identifying causes of failure of synthetic biological systems," *Biotechnol. J.*, vol. 7, no. 7, pp. 856–866, Jul. 2012.

[3] B. A. Renda et al., "Engineering reduced evolutionary potential for synthetic biology," *Mol. Biosyst.*, vol. 10, no. 7, pp. 1668–1678, 2014.

[4] MIL-P-1629, "Procedures for Performing a Failure Mode, Effects and Criticality Analysis," 1949.

[5] D. ~W. Clement, "Computer aided failure analysis," *NASA STI/Recon Tech. Rep. N*, vol. 77, p. 20348, Aug. 1976.

[6] W. V Oxford and R. H. Propst, "Efficient Computer-Aided Failure Analysis of Integrated Circuits using Scanning Electron Microscopy," *Reliab. IEEE Trans.*, vol. R-34, no. 5, pp. 410–417, Dec. 1985.

[7] M. Lehtelä, "Computer-aided failure mode and effect analysis of electronic circuits," *Microelectron. Reliab.*, vol. 30, no. 4, pp. 761–773, 1990.

[8] S. C. Sleight et al., "Designing and engineering evolutionary robust genetic circuits.," *J. Biol. Eng.*, vol. 4, no. 1, p. 12, 2010.

[9] D. J. Russomanno et al., "Viewing Computer-Aided Failure Modes and Effects Analysis from an Artificial Intelligence Perspective," IOS Press.

[10] M. Galdzicki et al., "The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology.," *Nat. Biotechnol.*, vol. 32, no. 6, pp. 545–50, 2014.

[11] M. L. Wilson et al., "Sequence verification of synthetic DNA by assembly of sequencing reads," *Nucleic Acids Res.*, vol. 41, no. 1, pp. 1–11, 2013.

[12] P. a. Lee et al., "CLONEQC: lightweight sequence verification for synthetic biology," *Nucleic Acids Res.*, vol. 38, no. 8, pp. 2617–2623, 2010.

[13] S. M. Maurer et al., “Making Commercial Biology Safer: What the Gene Synthesis Industry Has Learned About Screening Customers and Orders,” *Goldman Sch. …*, pp. 1–29, 2009.

[14] S. Kosuri and G. M. Church, “Large-scale de novo DNA synthesis: technologies and applications.,” *Nat. Methods*, vol. 11, no. 5, pp. 499–507, 2014.

[15] K. Temme et al., “Refactoring the nitrogen fixation gene cluster from Klebsiella oxytoca,” *Proc. Natl. Acad. Sci.*, vol. 109, no. 18, pp. 7085–7090, 2012.

[16] M. W. Lux et al., “Genetic design automation: engineering fantasy or scientific renewal?,” *Trends Biotechnol.*, vol. 30, no. 2, pp. 120–126, Oct. 2015.

[17] T. S. Ham et al., “Design, implementation and practice of JBEI-ICE: An open source biological part registry platform and tools,” *Nucleic Acids Res.*, vol. 40, no. 18, pp. 1–8, 2012.

[18] B. Kacar and E. Gaucher, “Towards the Recapitulation of Ancient History in the Laboratory: Combining Synthetic Biology with Experimental Evolution,” *Artif. Life*, vol. 13, p. 8, 2012.

[19] R. E. Cobb et al., “Directed Evolution as a Powerful Synthetic Biology Tool,” *Methods*, vol. 60, no. 1, pp. 81–90, Mar. 2013.

[20] K. H. Kim et al., “Controlling E . coli gene expression noise,” vol. 9, no. 3, pp. 497–504, 2015.

[21] B. Canton et al., “Refinement and standardization of synthetic biological parts and devices,” *Nat. Biotechnol.*, vol. 26, no. 7, pp. 787–793, 2008.

[22] J.-Y. Tinevez et al., “Chapter fifteen - A Quantitative Method for Measuring Phototoxicity of a Live Cell Imaging Microscope,” in *Imaging and Spectroscopic Analysis of Living CellsImaging Live Cells in Health and Disease*, vol. Volume 506, P. M. C. B. T.-M. in Enzymology, Ed. Academic Press, 2012, pp. 291–309.

[23] V. Magidson and A. Khodjakov, “Chapter 23 - Circumventing Photodamage in Live-Cell Microscopy,” in *Digital Microscopy*, vol. Volume 114, G. S. and D. E. W. B. T.-M. in C. Biology, Ed. Academic Press, 2013, pp. 545–560.

[24] J. a. Sniegowski, “Base Catalysis of Chromophore Formation in Arg96 and Glu222 Variants of Green Fluorescent Protein,” *J. Biol. Chem.*, vol. 280, no. 28, pp. 26248–26255, 2005.

[25] D. Evanko, “Is phototoxicity compromising experimental results?,” *Nature methods*, 2013. [Online]. Available: http://blogs.nature.com/methagora/2013/11/is-phototoxicity-compromising-experimental-results.html.

[26] F. Sievers et al., “Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega,” *Mol. Syst. Biol.*, vol. 7, no. 1, Oct. 2011.

[27] P. Rice et al., “EMBOSS: The European Molecular Biology Open Software Suite,” *Trends Genet.*, vol. 16, no. 6, pp. 276–277, Oct. 2015.

Chapter 4

TEST

4.1 Introduction

Experimental characterization corresponds to the "test" stage of the design-build-test cycle for synthetic biology. Synthetic biologists employ a range of experimental techniques to characterize the encoded behavior of a biological part, device, or system. These techniques very often involve the use of *in vivo* molecular reporters like fluorescent proteins and fluorescent aptamers which allow measurements to be made in living cells.

One technique, often used in our lab, relies on microplate readers which automates bacterial culturing and spectrofluorimetric measurements. This technique facilitates high-throughput analysis of up to 96 cultures at once. In the article *Designing and Engineering Evolutionary Robust Genetic Circuits* we used this technique to monitor the expression of several genetic constructs over a long time period. This study demonstrated engineered genetic circuits in synthetic microbes become unstable over evolutionary time in the absence of positive selective pressure. Some simple design principles for extending the life of genetic circuits are discussed.

Flow cytometry is another technique used in our lab. This technique enables single-cell measurements. Although high-resolution time course data cannot be acquired using flow cytometry, it has the advantage of measuring cell-to-cell variation. As shown in our manuscript *Controlling Gene Expression Noise in E. coli*, intracellular protein copy numbers show significant cell-to-cell variability within an isogenic population due to the random nature of biological reactions.

Although rational engineering and forward design of biological systems is an aspirational ideal of synthetic biologists, in reality, most genetic constructs do not work as intended. Consequently, synthetic biologists often employ combinatorial approaches to generate large libraries of genetic variants composed of randomized parts. Afterwards, they screen the library to isolate variants which exhibit the desired behavior. The conference abstract *Exploring Genetic Design Space with Phylosemantics* describes a computational algorithm for screening a variant library. The algorithm systematically classifies random, combinatorial variants according to similarities in their genetic configuration. The phylosemantic algorithm clusters common configurations together, making it easier to detect patterns in gene expression associated with particular genetic configurations. Phylosemantics might be used to discover "design rules" which can be applied to the forward-design of new genetic constructs.

4.2 Designing and Engineering Evolutionarily Robust Genetic Circuits

Sleight *et al. Journal of Biological Engineering* 2010, **4**:12
http://www.jbioleng.org/content/4/1/12

JOURNAL OF BIOLOGICAL ENGINEERING

RESEARCH Open Access

Designing and engineering evolutionary robust genetic circuits

Sean C Sleight*, Bryan A Bartley, Jane A Lieviant, Herbert M Sauro

Abstract

Background: One problem with engineered genetic circuits in synthetic microbes is their stability over evolutionary time in the absence of selective pressure. Since design of a selective environment for maintaining function of a circuit will be unique to every circuit, general design principles are needed for engineering evolutionary robust circuits that permit the long-term study or applied use of synthetic circuits.

Results: We first measured the stability of two BioBrick-assembled genetic circuits propagated in *Escherichia coli* over multiple generations and the mutations that caused their loss-of-function. The first circuit, T9002, loses function in less than 20 generations and the mutation that repeatedly causes its loss-of-function is a deletion between two homologous transcriptional terminators. To measure the effect between transcriptional terminator homology levels and evolutionary stability, we re-engineered six versions of T9002 with a different transcriptional terminator at the end of the circuit. When there is no homology between terminators, the evolutionary half-life of this circuit is significantly improved over 2-fold and is independent of the expression level. Removing homology between terminators and decreasing expression level 4-fold increases the evolutionary half-life over 17-fold. The second circuit, I7101, loses function in less than 50 generations due to a deletion between repeated operator sequences in the promoter. This circuit was re-engineered with different promoters from a promoter library and using a kanamycin resistance gene (*kanR*) within the circuit to put a selective pressure on the promoter. The evolutionary stability dynamics and loss-of-function mutations in all these circuits are described. We also found that on average, evolutionary half-life exponentially decreases with increasing expression levels.

Conclusions: A wide variety of loss-of-function mutations are observed in BioBrick-assembled genetic circuits including point mutations, small insertions and deletions, large deletions, and insertion sequence (IS) element insertions that often occur in the scar sequence between parts. Promoter mutations are selected for more than any other biological part. Genetic circuits can be re-engineered to be more evolutionary robust with a few simple design principles: high expression of genetic circuits comes with the cost of low evolutionary stability, avoid repeated sequences, and the use of inducible promoters increases stability. Inclusion of an antibiotic resistance gene within the circuit does not ensure evolutionary stability.

Background

Synthetic biology is the design and engineering of new biological functions and systems that do not occur in nature. This relatively new field has provided insight into the mechanisms of natural gene networks [1,2] and engineered multicellular pattern formations [3], bacterial photography [4], tumor-targeting bacteria [5], feed-forward network based concentration sensors [6], robust and tunable oscillators [7], and genetic networks that count [8]. On the genome level, entire metabolic pathways have been engineered to overproduce an anti-malaria compound [9], biofuels from plant biomass [10,11] lycopene through automated genome engineering and accelerated evolution [12], and a synthetic chromosome [13] transplanted into a host bacterium [14]. Despite recent efforts of engineering at the genome level, most synthetic biology constructs are engineered at the level of genetic circuits encoded on plasmids.

Genetic circuits are built bottom-up from biological parts. A biological part is a DNA sequence that encodes a basic biological function [15]. Examples of parts

* Correspondence: sleight@u.washington.edu
Department of Bioengineering, University of Washington, Seattle, WA 98195, USA

include promoters, ribosome binding sites (RBS), protein or RNA coding regions, and transcriptional terminators. Biological engineers can assemble individual parts or combination of parts together using a BioBrick assembly standard for physical composition [16] (described in [17]). Parts that conform to this BioBrick assembly standard are BioBrick standard biological parts, or BioBricks. Standard Assembly involves digestion of two BioBricks encoded on plasmids with different restriction enzymes that leave compatible sticky ends which can be ligated together into a new BioBrick. This assembly method effectively replaces the restriction sites between the assembled parts with a 'scar' sequence, allowing for the new BioBrick to be later combined with other BioBricks. Alternative assembly strategies have recently been proposed [18,19] to improve upon the original assembly standard. The MIT Registry of Standard Biological Parts (called "The Registry" from here on) maintains over 3000 BioBricks encoded on plasmids that are available to researchers with a wide variety of different functions, from bacterial photography, to quorum sensing to odor production and sensing.

BioBricks are widely available for the design of more complex systems, but in general are not well-characterized [15,17]. The most well-characterized part to date is a cell-cell communication receiver device [17], which was provided with a published prototype "biological part datasheet" containing information engineers would need to use it in their own designs. One of the figures in this datasheet describes the reliability of this circuit over evolutionary time. Connecting the receiver device to a GFP-reporter device causes this circuit to repeatedly lose function in less than 100 generations due to a deletion mutation between transcriptional terminators that are repeated in both the receiver and reporter devices. Another example of genetic circuits losing function over evolutionary time is illustrated by studies of microchemostat-evolved strains containing a cell density regulation circuit that loses function in less than 100 hours [20,21]. The evolutionary stability of whole circuits is therefore an emergent property of the context of its biological parts.

Evolutionary stability is a problem in genetic circuits if there is no selective pressure to maintain function of the circuit. The current belief is that this loss-of-function occurs because any cell in the population that acquires a mutation in the genetic circuit often has a growth advantage and can outcompete the cells in the population with all functional plasmids. As the cells divide, any cell with a larger percentage of mutant plasmids will eventually dominate the population until only cells with mutant plasmids remain. A simulation study predicted that the time for a non-functional mutant of a synthetic microbe to become the majority of the population is a function of the growth rate difference between the mutant and functional cells, circuit size, circuit architecture, and mutation rate [22]. Non-functional mutants often have a growth advantage because a mutation that inactivates a genetic circuit can reduce its metabolic load. The magnitude of metabolic load caused by expression and replication of foreign genes is dependent on many factors such as plasmid size, plasmid copy number, the foreign gene being expressed, antibiotic resistance gene, metabolic state of the cell, growth media, and amount of dissolved oxygen in the media [23]. Dekel and Alon [24] directly measured the cost associated with expression and maintenance of Lac proteins when they provided no fitness benefit and found mutations that alleviated this cost in the non-selective environment. There are also examples of chromosomal genes that have lost function over evolutionary time when not under selection [25,26] and so encoding synthetic circuits into the chromosome will only delay this problem.

The evolutionary stability of genetic circuits within synthetic microbes will be an increasingly significant issue as these circuits become more complex and need to be functional over longer periods of time. The ability to engineer evolutionary robust genetic circuits will be important for applied uses of synthetic microbes that perform long-term functions in the environment and possibly in the human body. This ability will also be important for the study of genetic circuits in microchemostats and microfluidic devices over multiple generations. Ideally, a selective regime should be used to maintain circuit function over evolutionary time. However, design of a selective regime for synthetic microbes is unique to the genetic circuit of interest, and design for maintaining function of a particular circuit is often difficult. Therefore, general design principles are needed for engineering evolutionary robust circuits that will maximize stability over time.

As a first step towards this goal, this study aimed to understand the loss-of-function mutations that occur in two genetic circuits over evolutionary time and their evolutionary stability dynamics. Next, we re-engineered these circuits in various ways to determine the predictability of mutations in replicate evolved populations and whether we could make these circuits more evolutionary robust. The results from these experiments allowed us to observe the mutations in several diverse circuits, determine their evolutionary stability dynamics, and develop simple design principles for engineering evolutionary robust circuits.

Results

Loss-of-function mutations and evolutionary stability dynamics in two genetic circuits

We first measured the evolutionary stability dynamics of two genetic circuits propagated in *Escherichia coli*

MG1655 in order to determine the loss-of-function mutations that cause their instability and which circuit is the most robust over evolutionary time. High-copy plasmids were used instead of low or medium-copy plasmids to maximize selective pressure so that evolution would occur more rapidly since replication and expression of genetic circuits encoded on high-copy plasmids will increase metabolic load and lower fitness. Cells with a low metabolic load (e.g., cells with mutant plasmids) have greater fitness than cells with a higher metabolic load (e.g., cells with functional plasmids) (unpublished results). Therefore, we expect that mutants will be able to rapidly outcompete functional cells that have a high expression level. However, other factors besides expression level will play a role in this evolutionary process such as mutation rate and the metabolic load associated with plasmid replication.

The two circuits we used to measure the evolutionary stability dynamics and determine the loss-of-function mutations were T9002 (Figure 1a) and I7101 (Figure 2a). T9002 is the Lux receiver circuit previously described [17] and expresses *luxR* that activates GFP expression when the inducer AHL is added to the media (see Figure 1 legend for details). I7101 has a *lacI*-regulated promoter and expresses GFP only when the inducer IPTG is added to the media since *lacI* is constitutively overexpressed from the chromosome in this particular strain (*Escherichia coli* MG1655 Z1). The evolutionary stability dynamics were measured by serial propagation with a dilution factor that allows for about 10 generations per day.

Figure 1b shows the evolutionary stability dynamics of the T9002 circuit propagated in high input (with AHL) and low input (without AHL) conditions. From different timepoints in the experiment, the low and high input populations were induced with AHL to measure their normalized expression (here measured by fluorescence divided by cell density) over time. The low input evolved populations slowly lose their function to about 50% of the maximum after 300 generations. The evolved populations in high input conditions rapidly lose their function in less than 30 generations (the dynamics of this evolutionary stability are described below in Figure 3). No functional clones were observed after 30 generations as determined by measurement of individual colonies. The mutation that repeatedly causes loss-of-function in the high input evolved populations is a deletion between two homologous transcriptional terminators (Figure 1c), the same mutation described in [17]. This mutation evidently occurs at such a high rate that mutants quickly take over the population. In fact, Canton *et al* (2008) [17] were unable to isolate a population derived from a single isolate that did not already carry the deletion. The mutant plasmid was transformed back into the progenitor and was shown not to fluoresce after induction with AHL. In this initial study we also tested the evolutionary stability of a BioBrick engineered version of the repressilator circuit [1]. We could not measure its function over time due to unstable GFP expression at the population level, but found that the circuit repeatedly had a deletion between homologous *tetR* promoters.

Figure 2b shows the evolutionary dynamics of the I7101 circuit propagated in high (with IPTG) and low input (without IPTG) conditions. The evolved populations in low input conditions lose about 70% of their function over 300 generations. The high input evolved populations lose about half their function in 30 generations and nearly all function after 300 generations. For this circuit, the loss-of-function is repeatedly a deletion between two homologous operator sequences in the promoter (Figure 2c). The mutant plasmid was transformed back into the progenitor and was shown not to fluoresce when induced with IPTG. This initial study suggests that the use of repeated parts in synthetic circuits should be avoided due to the high mutation rate. Also, there is a high metabolic load associated with the expression of genetic circuits on high-copy plasmids since keeping these circuits off substantially improves evolutionary stability.

Evolution experiment with re-engineered circuits

Based on the results of the previous experiments, we re-engineered the T9002 and I7101 circuits to test various predictions of evolutionary stability and mutational predictability. For the T9002 circuit, the loss-of-function mutation was repeatedly a deletion between two homologous transcriptional terminators. Mutations and genetic rearrangements can occur due to misalignment of homologous sequences during replication (termed "replication slippage") [27]. Deletion mutations between repeated sequences are known to be dependent upon repeat length, proximity, and homology level [28]. These deletions are *recA*-independent if the repeat length is less than 200 bp [27,29], as is the case with the repeated terminators in the T9002 circuit. Thus, we re-engineered the last terminator of T9002 with various terminators available in the Registry to measure the effect of terminator homology level and orientation with evolutionary stability. We predicted that we could increase evolutionary robustness by decreasing the mutation rate of this deletion. Furthermore, although there have been several studies on recombination between repeated sequences, this phenomenon has not been studied in the context of synthetic biology using genetic circuits constructed from BioBricks. For instance, we do not know the effect of using various BioBrick terminators with different homology levels in the same circuit. The

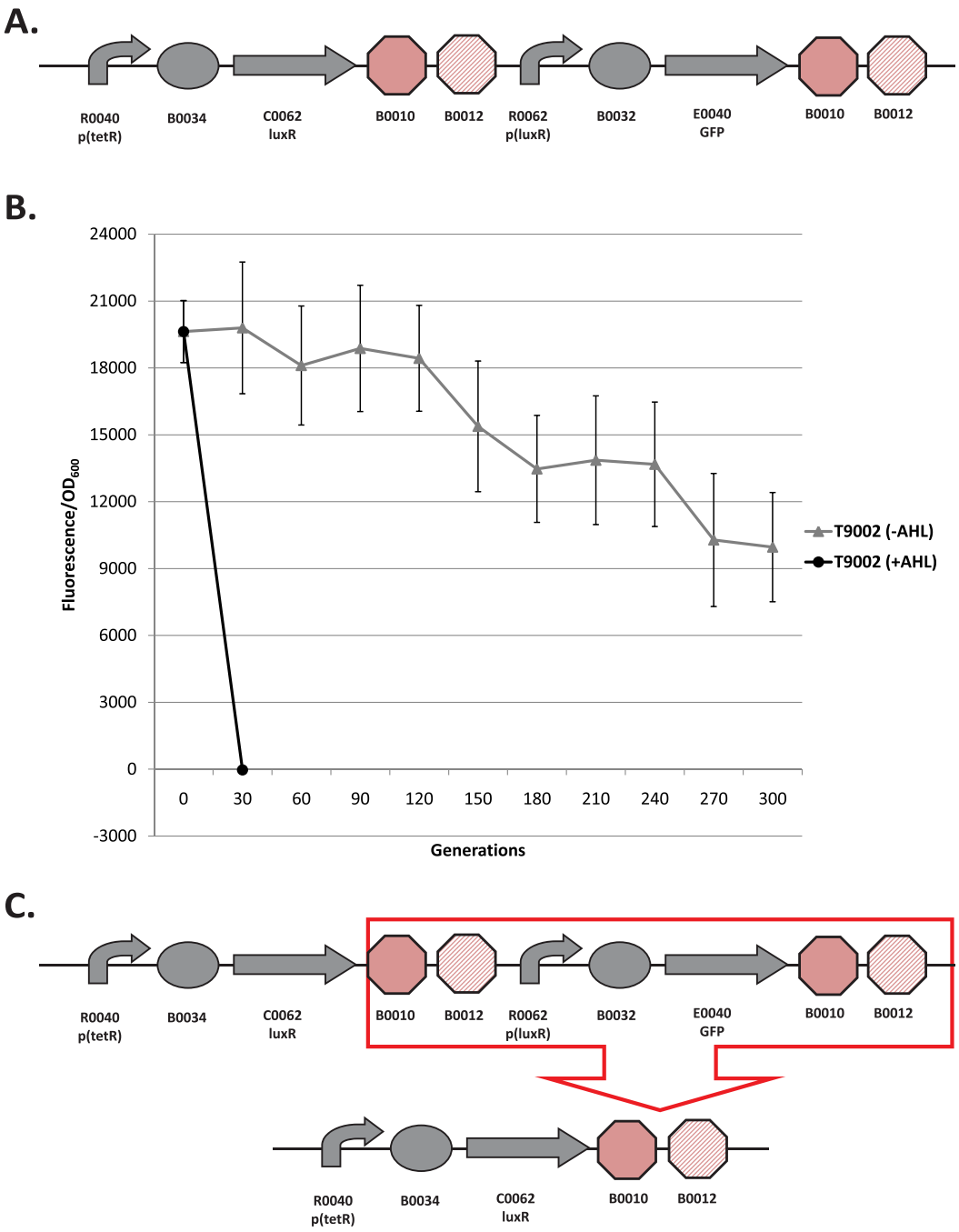


Figure 1 Loss-of-function mutations and evolutionary stability dynamics in T9002. (A) The T9002 genetic circuit. Symbols depict promoters (bent arrows), ribosome binding sites (ovals), coding sequences (arrows), and transcriptional terminators (octagons). T9002 consists of two devices, a luxR receiver device and a GFP-expressing device. The first device is composed of the *tetR*-regulated promoter R0040 that is constitutively expressed in the MG1655 strain since it does not produce TetR, B0034 RBS, C0062 luxR coding sequence, and B0010-B0012 (B0015) transcriptional terminator. The second device is composed of the R0062 luxR promoter, B0032 RBS, E0040 GFP coding sequence, and B0015 transcriptional terminator. LuxR is constitutively expressed from the *tetR* promoter. When the inducer $3OC_6HSL$ (AHL) is added to the media, it binds with LuxR to activate transcription of GFP from the *luxR* promoter. If no AHL is in the media, the circuit is off. (B) Evolutionary stability dynamics of T9002 evolved under low (-AHL) and high (+AHL) input conditions. Low and high input evolved populations are shown with solid gray triangles and solid black circles, respectively. Evolved populations at different timepoints were grown with AHL to measure relative GFP levels. Relative fluorescence normalized by OD is plotted vs. generations. Error bars represent one standard deviation from the mean of nine independently evolved populations. (C) This circuit repeatedly has a deletion between homologous repeated terminators after 30 generations in the high input evolved populations.

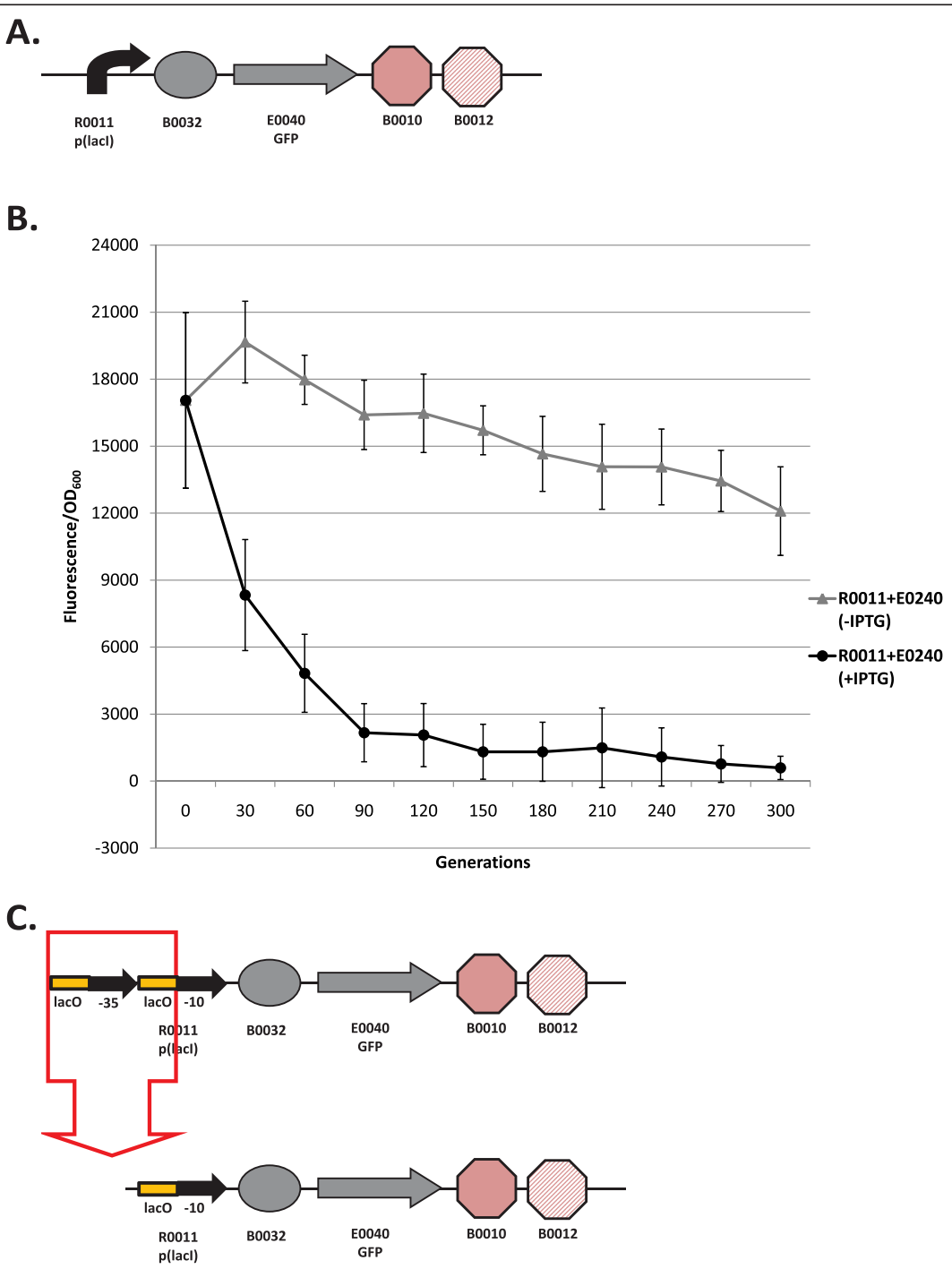
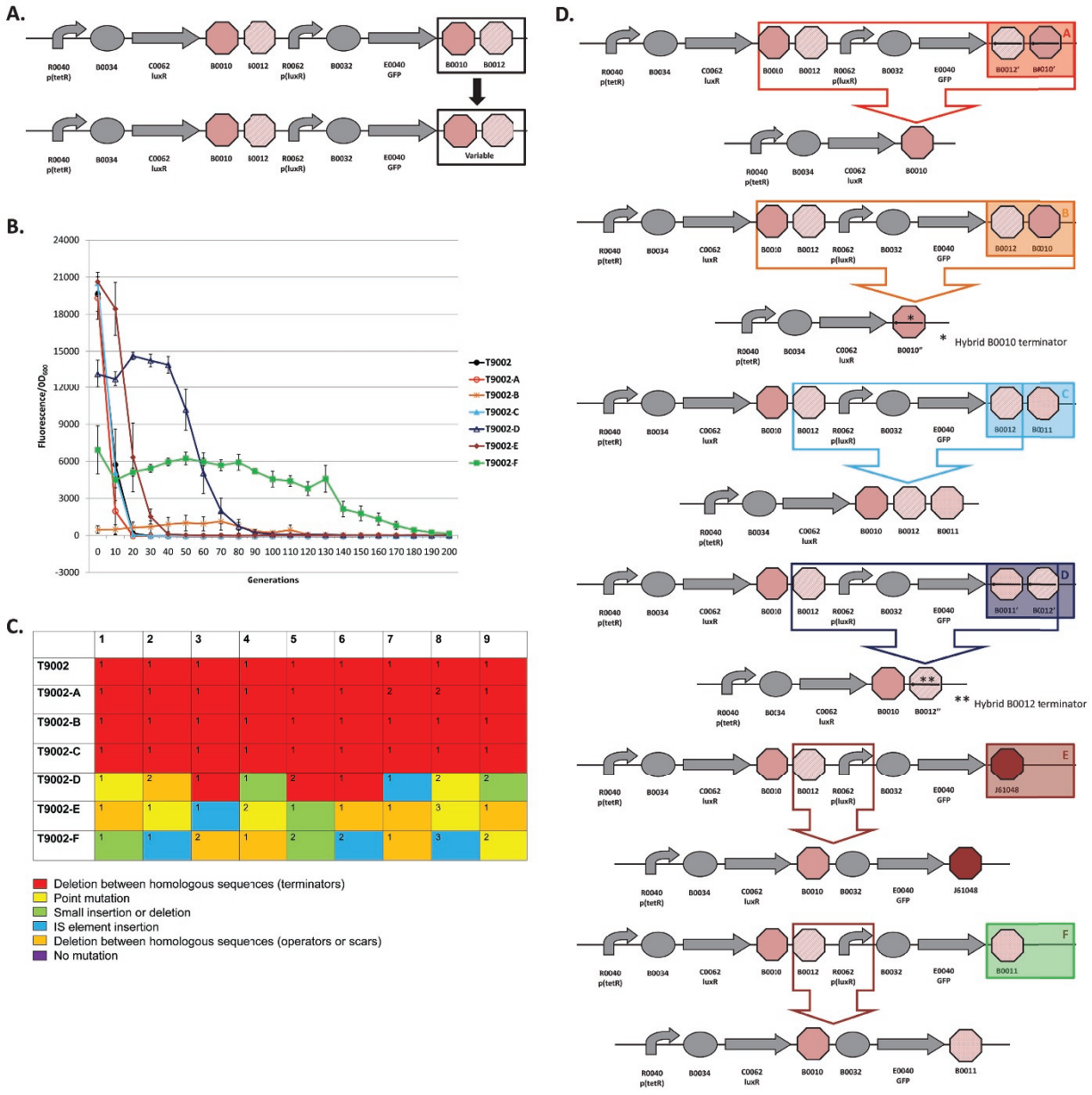


Figure 2 Loss-of-function mutations and evolutionary stability dynamics in I7101. (A) The I7101 genetic circuit. Symbols depict promoters (bent arrows), ribosome binding sites (ovals), coding sequences (arrows), and transcriptional terminators (octagons). I7101 consists of the promoter R0011 and the GFP-expressing element E0240 that is made up of the B0032 RBS, E0040 GFP coding sequence, and B0010-B0012 (B0015) transcriptional terminator. Since *lacI* is constitutively expressed from the chromosome, it represses GFP expression from the *lacI*-regulated promoter R0011. When the inducer Isopropyl-beta-D-thiogalactopyranoside (IPTG) is added to the media, it inhibits LacI and activates GFP expression. If no IPTG is in the media, the circuit is off. (B) Evolutionary stability dynamics of R0011 + E0240 evolved under low (-IPTG) and high (+IPTG) input conditions. Low and high input evolved populations are shown with solid gray triangles and solid black circles, respectively. Evolved populations at different timepoints were grown with IPTG to measure relative GFP levels. Relative fluorescence normalized by OD is plotted vs. generations. Error bars represent one standard deviation from the mean of nine independently evolved populations. (C) This circuit repeatedly has a deletion between homologous operators within R0011 after 90 generations in the high input evolved populations.



	1	2	3	4	5	6	7	8	9
T9002	1	1	1	1	1	1	1	1	1
T9002-A	1	1	1	1	1	1	2	2	1
T9002-B	1	1	1	1	1	1	1	1	1
T9002-C	1	1	1	1	1	1	1	1	1
T9002-D	1	2	1	1	2	1	1	2	2
T9002-E	1	1	1	2	1	1	1	3	1
T9002-F	1	1	2	1	2	2	1	3	2

Figure 3 Loss-of-function mutations and evolutionary stability dynamics in re-engineered T9002 circuits. (A) T9002 re-engineering involves changing the second double transcriptional terminator with varying degrees of homology and orientation to the first double transcriptional terminator. (B) Evolutionary stability dynamics of T9002 (solid black circles) and T9002 re-engineered circuits (various shapes and colors) under high input (+AHL) conditions. Error bars represent one standard deviation from the mean of nine independently evolved populations. (C) Types of mutations in nine independently evolved populations. For nine independently evolved populations, colored boxes correspond to the mutation legend below the table. The most common mutation for a particular type of mutation is labeled with "1" in the boxes above and less common mutations are labeled with increasing numbers. (D) Most common loss-of-function mutations that inactivated the re-engineered T9002 circuits. See Additional File 1, Supplementary Table S1 for mutation details.

use of different terminators will become increasingly important when more complex circuits are constructed and BioBricks become even more widespread in the community. Also, many of the studies on recombination between repeated sequences use antibiotic resistance genes to measure recombination rates and may not relate to actual functioning circuits.

We also re-engineered the I7101 circuit in two separate ways. The first was to re-engineer I7101 with various promoters from a promoter library to test whether

evolutionary stability is inversely correlated with promoter strength. The second was to insert a kanamycin resistance gene (*kanR*) within the circuit to put a selective pressure on the promoter. We engineered two versions of the KanR circuit, one as a GFP-KanR fusion protein with a glycine-serine linker and another where *kanR* is polycistronically expressed with GFP. Finally, we tested the effect of inducible vs. constitutive expression on evolutionary stability with two circuits having different *lacI*-regulated promoters.

Since we learned from previous experiments that evolutionary stability dynamics of genetic circuits have high variability between replicate populations, we evolved nine independent populations of each re-engineered circuit for at least 300 generations. Three experimental replicate populations of three independent tranformants were used to test for independent mutational events. A single transformant may have a mutation at a low level that will eventually sweep through the population, so if only one transformant was used, the same mutation may show up in all replication populations. For each of the nine populations in every circuit, the evolutionary half-life was measured to quantitate the number of generations until the expression level had decreased to half of its initial level (Table 1). Plasmids from a single clone from each evolved population were then sequenced after the population level had decreased to below 10% of the original expression level, or after 500 generations, whichever came first. Additional File 1, Supplementary Table S1 shows all mutations for each population in every circuit.

Table 1 Evolutionary half-life of various genetic circuits

Circuit	Evolutionary Half-life	SD
T9002	7.056	3.657
T9002-A	5.563	1.687
T9002-C	6.650	3.250
T9002-D	56.986	6.188
T9002-E	16.701	4.966
T9002-F	125.443	47.664
R0011 + E0240	19.833	3.818
R0011 + E0240 (inducible)	31.782	17.129
R0010 + E0240	42.363	14.729
R0010 + E0240 (inducible)	45.233	56.350
R0040 + E0240	59.838	13.445
J23101 + E0240	96.694	28.880
J23102 + E0240	36.428	13.338
J23105 + E0240	18.686	11.430
J23151 + E0240	62.844	27.168
R0010 + E0240 kanR polycistronic	71.000	29.031
R0010 + E0240 kanR polycistronic (+kan)	78.091	36.084
R0010 + E0240 kanR fusion	57.757	38.750
R0010 + E0240 kanR fusion (+kan)	66.252	54.056

The mean evolutionary half-life for nine independently evolved populations is shown for each genetic circuit with the standard deviation (SD). See the Methods section for details.

Re-engineered T9002 circuits with different transcriptional terminators: loss-of-function mutations and evolutionary stability dynamics

Figure 3a shows the schematic for re-engineering the last transcriptional terminator in the T9002 circuit. The evolutionary stability dynamics for six re-engineered T9002 circuits and the original T9002 circuit are shown in Figure 3b. Figure 3c shows the type of mutations that occurred in each of the nine replicate evolved populations. Finally, Figure 3d shows the most common mutations for each re-engineered circuit. The six re-engineered circuits are labelled T9002-A through T9002-F in Figure 3b and color-coded to correspond to the same circuit mutations shown in Figure 3d. These circuits were all propagated with the inducer AHL to allow for GFP expression. In the following paragraphs, the loss-of-function mutations and evolutionary stability dynamics for the original T9002 circuit and each re-engineered circuit will be described in detail.

T9002: The original T9002 circuit decreases rapidly to about 30% of the original level after only 10 generations and all function is lost by 20 generations (Figure 3b). The same deletion between homologous terminators as was observed in previous experiments (Figure 1c) was found in all nine replicate populations (Figure 3c). The evolutionary half-life of this circuit was found to be about 7.1 generations on average (Table 1).

T9002-A: The final double terminator in T9002 was re-engineered in the reverse complementary orientation (called B0025 in the Registry) to make T9002-A. The stability of this circuit has approximately the same expression level and stability dynamics as T9002, but has an evolutionary half-life of about 5.6 generations (Figure 3b, Table 1). This decreased stability may be because the terminator in the reverse orientation is more likely to cause replication slippage. Since the expression level is similar to T9002 and therefore the metabolic load should be roughly equivalent, the difference in stability is primarily due to an increased mutation rate. Seven of nine replication populations have a deletion between the first B0010 terminator and the reverse complement of B0010 (Figure 3c and 3d). This effect likely occurs because B0010 has a long hairpin structure, so one half of B0010 can interact with the other half of the reverse complementary B0010 sequence during DNA replication since they have perfect homology. Two of the nine populations had a deletion that formed a triple terminator of B0010-B0012-B0010 (Additional File 1, Supplementary Table S1).

T9002-B: The T9002-B circuit was re-engineered to re-arrange the B0010 and B0012 terminators to have B0012 first and then B0010. The re-arrangement decreases the expression level to almost zero initially and this expression drifts up over time and then decreases to zero (Figure 3b). For this circuit, evolutionary half-life measurements are essentially meaningless due to the randomness of low expression. Notice that other re-engineered T9002 circuits also have decreased expression levels relative to T9002, presumably due to weaker terminator hairpin structures having increased mRNA degradation [30] or transcriptional readthrough that can decrease plasmid copy number [31]. Others have observed that removal of transcriptional terminators can decrease expression levels in general [32]. All nine populations have the same deletion between B0010 terminators (Figure 3c and 3d). Because the B0010 terminator is an inexact hairpin (there are some mismatches), one half of the first B0010 interacts with the other half of the second B0010 terminator, causing a hybrid B0010 terminator (Figure 3d, Additional File 1, Supplementary Table S1).

T9002-C: The T9002-C circuit was re-engineered to have B0012 and B0011 as the final double terminator instead of B0010 and B0012. This circuit has nearly identical stability dynamics as T9002, with an evolutionary half-life of about 6.7 generations on average (Figure 3b, Table 1). All nine populations have the same deletion between B0012 terminators that make a triple terminator of B0010-B0012-B0011 (Figure 3c and 3d). Since the expression level and stability dynamics are roughly equivalent to T9002, the mutation rate between repeated B0010-B0012 terminators (129 bp) is probably about the same as between repeated B0012 terminators (41 bp). Interestingly, no significant stability difference was observed between T9002-C (41 bp homology) and T9002 or T9002-A (both 129 bp homology), despite having similar expression levels. This result suggests that shortening the repeated regions of homologous terminators did not increase evolutionary robustness, contrary to what we expected.

T9002-D: The T9002-D circuit has the same final B0012-B0011 terminator, but is the reverse complement of this sequence. The inclusion of this terminator decreases the initial expression level to about 65% of T9002 (Figure 3b). The evolutionary half-life of this circuit is about 57 generations (Figure 3b, Table 1). Also, the slope of the stability plot is decreased relative to other circuits with higher expression (T9002, T9002-A, T9002-C, and T9002-E) and the stability lag (time for expression to decrease to zero along the x-axis) is increased (Figure 3b). In contrast to other circuits with repeated terminators, only 3/9 have deletions between homologous terminators forming a hybrid B0012 terminator (Figure 3c and 3d). This result is probably because, unlike T9002-C, the second B0012 is the reverse complement, and therefore the only homology in this circuit is the 8-bp of hairpin structure having complementary sequences; the rest of the terminator has a loop structure of non-complementary sequences. In other words, B0010 has sufficient homology in either the forward or reverse orientation to cause replication slippage, but in B0012 replication slippage is more likely to occur only in the forward orientation. The other mutations in this circuit are composed of point mutations, short insertions or deletions, deletions between scar sequences, or insertion sequence (IS) mutations (Figure 3c, Additional File 1, Supplementary Table S1).

T9002-E: The T9002-E circuit, like the T9002-F circuit, was re-engineered to have no homology between terminator sequences. This circuit has the highest initial expression level on average probably because J61048 is a very strong terminator, but has similar expression relative to the T9002, T9002-A, and T9002-C circuits (Figure 3b). Its evolutionary half-life is about 16.7 generations (Figure 3b, Table 1). Thus, relative to other circuits with the similar expression levels, it is the most evolutionary robust circuit, having over 2-fold higher stability than T9002. When the evolutionary half-life is measured for the nine replicate populations, this evolutionary half-life difference compared to T9002 is highly significant (one-tailed t-test, $p = 0.0003$). Notice that the stability slope is similar to T9002, T9002-A, and T9002-C circuits, but the stability lag is increased by about 10 generations. This difference in lag is likely due to a decreased mutation rate since mutations are more random compared to the other similar expression-level circuits (Figure 3c, Additional File 1, Supplementary Table S1). The most common mutation is a deletion between repeated scar sequences that removes the *luxR* promoter and effectively inactivates the circuit function (Figure 3d).

T9002-F: The T9002-F circuit was re-engineered with the B0011 terminator, so it also has no homology between terminator sequences. The B0011 is evidently a weak terminator since its initial expression level is about 4-fold lower than T9002. Its stability dynamics show that it is the most evolutionary robust of the re-engineered T9002 circuits, with an evolutionary half-life of about 125 generations (Figure 3b, Table 1). This result indicates that decreasing homology levels and expression through terminator re-engineering increased the evolutionary half-life of this circuit over 17-fold relative to T9002. Like T9002-E, the mutations in each of the nine populations are mostly random (Figure 3c, Additional File 1, Supplementary Table S1). Also like T9002-E, the most common mutation is a deletion between repeated scar sequences that removes the *luxR* promoter driving GFP expression (Figure 3d). Since T9002-E and T9002-

F likely have similar mutation rates with zero terminator homology, the large stability difference between these circuits can be explained by expression levels alone.

Overall, excluding T9002-B, three of the five re-engineered T9002 circuits are more evolutionary robust than the original circuit. The order of evolutionary robust genetic circuits is: T9002-F > D > E > A = C = T9002. This increase in evolutionary robustness can be attributed to decreased expression levels (due to the terminator re-engineering) and to decreased mutation rate between homologous transcriptional terminators. The re-engineered circuits with homologous transcriptional terminators almost always have deletions between homologous regions, whereas circuits without homology have mutations in other locations in the circuit. Re-engineering this circuit to remove all homology effectively removes a certain class of mutations from occurring. The T9002-E circuit is more evolutionary robust than other circuits with similar expression levels likely due to decreased mutation rate alone. Thus, evolutionary robustness can be increased by removing long repeated sequences from genetic circuits, but even short 8-bp scar sequences have the potential for replication slippage.

Re-engineered I7101 circuits with different promoters: loss-of-function mutations and evolutionary stability dynamics

Figure 4 shows the re-engineering scheme for the I7101 circuit with different promoters (Figure 4a), evolutionary stability dynamics for these re-engineered circuits (Figure 4b), mutations found in the nine evolved populations (Figure 4c), and the most common mutation for each re-engineered circuit (Figure 4d). These circuits were constitutively expressed (no inducer was required to activate the circuit). Many of these promoters were used in the promoter measurement kit, an effort to standardize promoter measurements between promoters [33]. This previous study used medium-copy plasmids whereas we used high-copy plasmids and different environmental conditions. The loss-of-function mutations (if any) and evolutionary stability dynamics for the original I7101 circuit and each re-engineered circuit are described in detail below.

I7101 (R0011 + E0240): The evolutionary stability of the original I7101 circuit was measured constitutively (unlike Figure 2 with inducible expression) and is shown in Figure 4b. The evolutionary half-life of this circuit is about 20 generations (Table 1) and reaches zero after 40 generations (plotted every 30 generations for clarity). Not surprisingly, all nine populations have the same deletion between repeated operator sequences in the promoter (Figure 4c and 4d).

R0010 + E0240: When the R0011 promoter is swapped out for R0010, the wildtype *lacI*-regulated promoter without repeated operator sequences, the initial expression level decreases slightly (Figure 4b). The evolutionary half-life of this circuit is about 42 generations (Table 1) and roughly double that of R0011 + E0240. This circuit is more evolutionary robust than R0011 + E0240 probably due to a combination of decreased expression level and mutation rate. Also, this circuit never reaches zero expression. When this circuit was sequenced, surprisingly no mutations were found in any of the nine populations (Figure 4c). However, expression levels decreased due to unknown mutations in the chromosome (Additional File 1, Supplementary Material).

R0040 + E0240: R0040 is a *tetR*-regulated promoter with repeated *tetO* operator sequences. This circuit's evolutionary half-life is about 60 generations (Figure 4b, Table 1). Unsurprisingly, all nine populations have a deletion between the repeated operator sequences, similar to that of R0011 + E0240 (Figure 4c and 4d).

J23101/J23102/J23105/J23150/J23151 + E0240: All five of these circuits have very different initial expression levels and evolutionary half-lives, but are similar in that they maintain relatively high expression levels after an initial decrease except for J23150 + E0240 that maintains a relatively constant level (Figure 4b, Table 1). Like R0010 + E0240, three circuits do not have any mutations in all nine populations.

In summary for this section, re-engineered I7101 circuits have a large amount of variation in stability. This variation is probably mostly due to differences in expression level, but mutation rate also contributes to this variability. We expected that promoter strength to be inversely proportional to evolutionary half-life, and this hypothesis was tested in the Evolutionary half-life measurements of individual populations section (Additional File 1, Supplementary Material). Only circuits with repeated operator sequences reached zero expression, probably because the deletion removes the entire -35 region and therefore no leaky expression can occur. Promoters without repeated operator sequences should be used to maximize evolutionary stability.

Re-engineered I7101 circuits with a kanamycin resistance gene: loss-of-function mutations and evolutionary stability dynamics

We also re-engineered the I7101 circuit to have a kanamycin resistance gene (*kanR*) in order to put a selective pressure on the promoter (Figure 5a). Both a GFP-KanR fusion coding sequence and polycistronic transcribed coding sequence were engineered (Figure 5a). We also swapped out the R0011 promoter with the R0010 promoter since it was found to be more evolutionary robust than R0011 (Figure 5a). The other parts of this figure show the evolutionary stability dynamics for these re-engineered circuits (Figure 5b), mutations found in the nine evolved populations (Figure 5c), and most

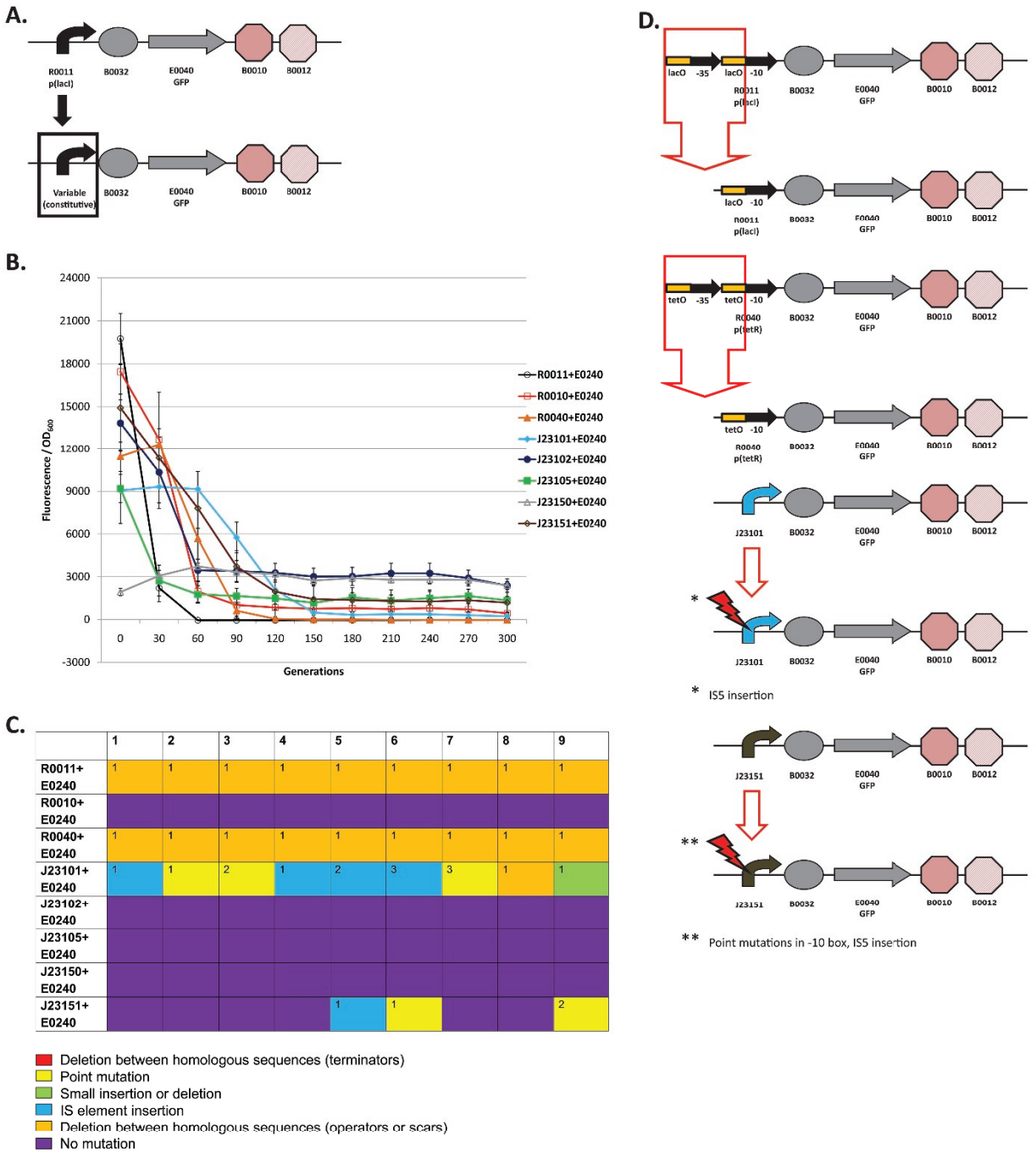


Figure 4 Loss-of-function mutations and evolutionary stability dynamics in re-engineered I7101 circuits (A) I7101 (R0011 + E0240) re-engineering involves swapping out the R0011 promoter. (B) Evolutionary stability dynamics of R0011 + E0240 (open black circles) and re-engineered circuits (various shapes and colors) under constitutive expression. Error bars represent one standard deviation from the mean of nine independently evolved populations. (C) Types of mutations in nine independently evolved populations. For nine independently evolved populations, colored boxes correspond to the mutation legend below the table. The most common mutation for a particular type of mutation is labeled with "1" in the boxes above and less common mutations are labeled with increasing numbers. (D) Most common loss-of-function mutations that inactivated the re-engineered I7101 circuits. See Additional File 1, Supplementary Table S1 for mutation details.

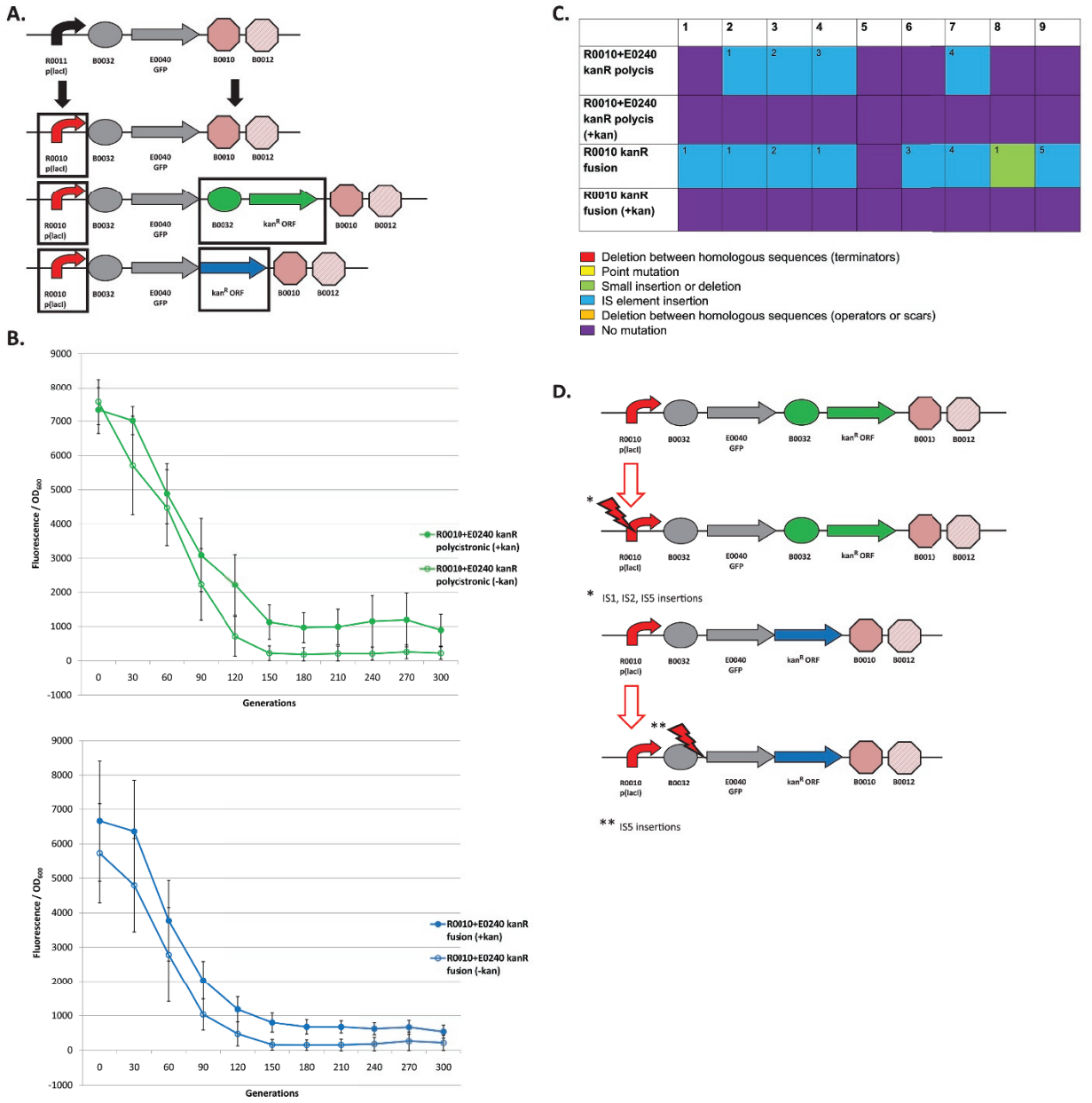


Figure 5 Loss-of-function mutations and evolutionary stability dynamics in re-engineered I7101 circuits with a kanamycin resistance gene. (A) I7101 re-engineering with the addition of a kanamycin resistance (*kanR*) gene. First the R0010 promoter was added instead of R0011 (top). Then, this circuit was re-engineered to polycistronically transcribe *gfp* and *kanR* separately into separate GFP and KanR proteins (middle) and to express a GFP-KanR fusion protein (bottom). (B) Top panel shows the evolutionary stability dynamics of R0010 + E0240 *kanR* polycistronic circuits propagated with kanamycin (solid green circles) and without kanamycin (open green circles). Bottom panel shows the evolutionary stability dynamics of R0010 + E0240 *kanR* fusion circuits propagated with kanamycin (solid blue circles) and without kanamycin (open blue circles). R0010 + E0240 and R0011 + E0240 evolutionary stability dynamics are shown in Figure 4. Error bars represent one standard deviation from the mean of nine independently evolved populations. (C) Types of mutations in nine independently evolved populations. For nine independently evolved populations, colored boxes correspond to the mutation legend below the table. The most common mutation for a particular type of mutation is labeled with "1" in the boxes above and less common mutations are labeled with increasing numbers. (D) Most common loss-of-function mutations that inactivated the re-engineered I7101 circuits with a kanamycin resistance gene. See Additional File 1, Supplementary Table S1 for mutation details.

common mutation for each re-engineered circuit (Figure 5d). These circuits were constitutively expressed and the KanR circuits were propagated with and without kanamycin (kan) in the media. We predicted that the kan propagated circuits would be more evolutionary robust than those without kan in the media.

The upper panel of Figure 5b shows the evolutionary stability dynamics for the R0010 + E0240 *kanR* polycistronic circuits with and without kan in the media. Introduction of the *kanR* coding sequence into the R0010 + E0240 circuit lowers expression by about 60% (Figure 4b and Figure 5b, upper panel). This lowered expression may be due to competition between the RBS used for GFP translation vs. the RBS used for KanR translation. This difference in expression complicates any useful comparisons between the R0010 + E0240 circuit and the R0010 + E0240 *kanR* polycistronic circuit. However, since the KanR circuit was measured with and without kan in the media, we can determine the effect the kanamycin is having on the circuit independent of expression levels. The KanR circuit propagated with kan has an evolutionary half-life of about 78 generations whereas without kan the half-life is 71 generations, but this result is not statistically significant (one-tailed t-test, $p = 0.0896$). However, the circuit propagated with kan remains at a higher level over time (Figure 5b, upper panel).

The lower panel of Figure 5b shows the same comparison between circuits except with the GFP-KanR fusion version of the circuit. Introduction of the *kanR* fusion coding sequence also lowers expression by about 60% and this may be due to folding issues in the GFP-KanR fusion protein. Similar to the polycistronic version of the KanR circuit, the fusion version also only has an evolutionary half-life difference of less than 10 generations when propagated with kan compared to without kan in the media (Table 1 and Figure 5b, lower panel). Again, this result is not statistically significant (one-tailed t-test, $p = 0.1174$). Like the polycistronic version of this circuit, propagation in kan causes evolutionary stability to remain at a higher level over time (Figure 5b, lower panel).

Figure 5c shows the mutations found in each of the nine populations. Interestingly, 4/9 of the KanR polycistronic circuits and 7/9 of the KanR fusion circuits propagated without kan in the media have IS mutations at the generation 200 timepoint. The KanR circuits propagated with kan in the media and the R0010 + E0240 circuit have no mutations at this timepoint. However, after propagating these KanR circuits for 500 generations, 2/9 populations had mutations in the promoter that were not IS mutations. Evidently, mutations on the chromosome confer resistance to kanamycin because introduction of these mutated circuits back into the progenitor does not make these cells kan resistant.

Since only KanR circuits propagated without kan in the media have IS mutations, propagating strains carrying the metabolic load of express resistance to both ampicillin (encoded on the plasmid backbone) and kanamycin could have caused a significant amount of cellular stress and triggered IS transposition. Some evidence suggests that IS transposition can occur in response to stress [34-38], but it also occurs in non-stressful conditions as well [26,38,39]. IS mutations probably occurred in the KanR circuits propagated with kan in the media as well, but cells carrying these circuits were selected against. Additional experiments partially explain why KanR circuits propagated with kan lose expression over time without having mutations in the circuit itself (Additional File 1, Supplementary Material).

The IS mutations in the KanR circuits propagated without kan in the media consisted of IS1, IS2, and IS5 mutations (Figure 5d, Additional File 1, Supplementary Table S1). A hotspot for IS5 mutations "C(T/A)A(G/A)" [40,41] often occurs in the "CTAG" portion of the scar sequence between either the promoter and RBS ("TA**CTAG**AG") or RBS and coding sequence ("TA**CTAG**"). This one unexpected result may be unfortunate for genetic circuits assembled with BioBricks if the host strain carries IS5 elements and if evolutionary stability is an issue. Also, since the IS5 mutations in these circuits were located in a position just downstream of the promoter, we thought that the transposase gene within the IS element might be transcribed and translated. However, the orientation of these IS insertions in these circuits would not allow for expression to occur. Unlike IS5 which has a defined hotspot, the IS1 and IS2 mutations are more random, but appear to transpose in A-T rich regions (Additional File 1, Supplementary Table S1).

Overall, the use of a kanamycin resistance gene within the circuit does not significantly increase evolutionary stability. Although propagation with kanamycin in the media may put a selective pressure on the promoter, other mutational targets evidently can decrease expression of the circuit over time (Additional File 1, Supplementary Material). IS mutations are responsible for loss-of-function in the KanR circuits propagated without kan. Since there are relatively few IS mutations in circuits without *kanR* (Figures 3c, 4c, 5c, and 6c), this circuit may induce IS transposition bursts.

As a final note for this section, we also tried to insert *kanR* into the T9002 circuit that would be polycistronically transcribed with GFP. The cells carrying this circuit did not grow well when AHL was added to the media. Expressing *kanR* using the strong *luxR* promoter may have been somewhat toxic to the cells. Decreasing the strength of the RBS that controlled the translation

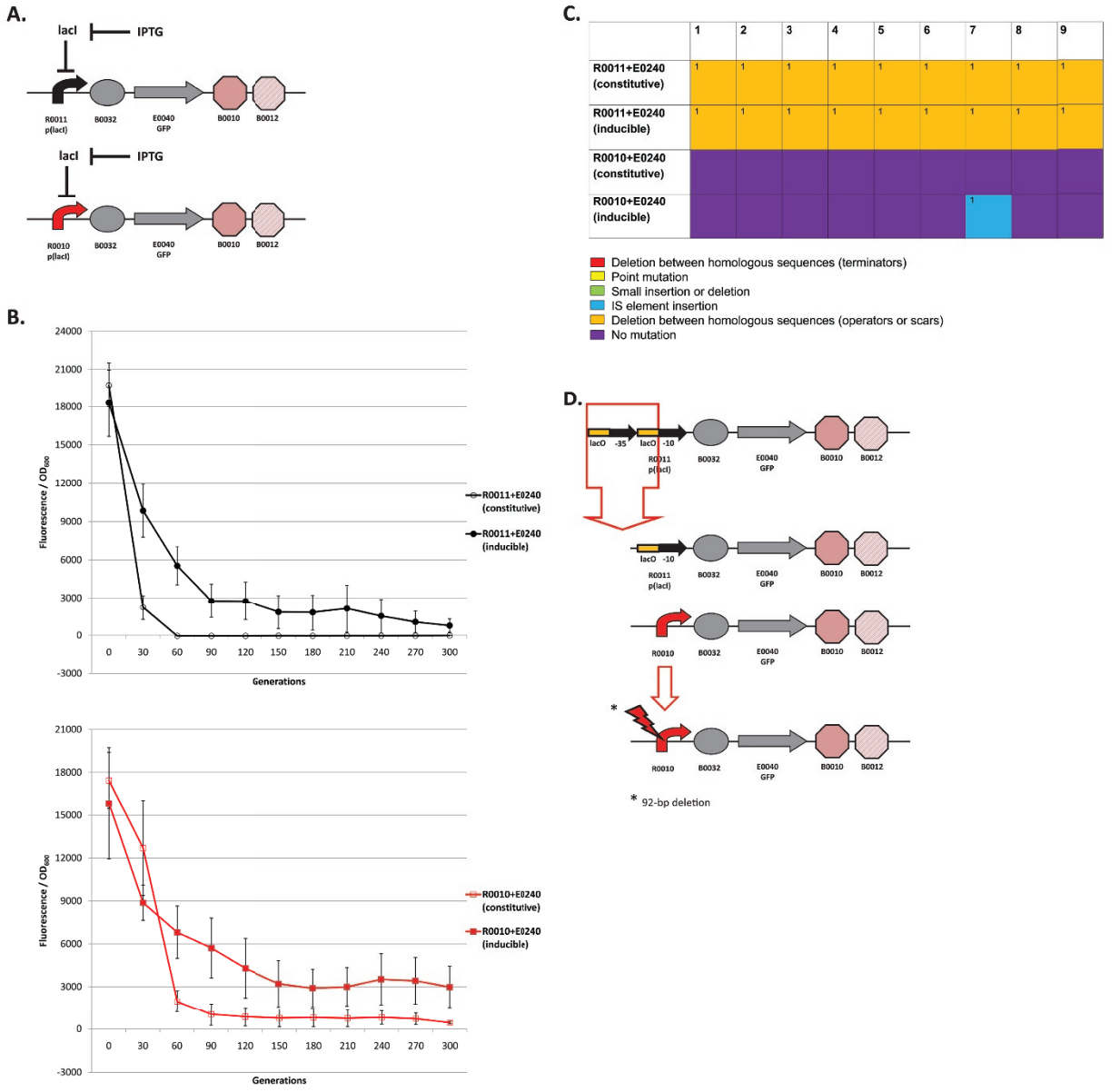


Figure 6 Loss-of-function mutations and evolutionary stability dynamics in two *lacI*-regulated circuits under constitutive vs. inducible expression. (A) Regulation of inducible R0011 + E0240 and R0010 + E0240 circuits. LacI represses transcription of GFP and IPTG de-represses the circuit to allow for GFP expression. (B) Top panel shows the evolutionary stability dynamics of constitutive R0011 + E0240 (open black circles) and inducible R0011 + E0240 (solid black circles). Bottom panel shows the evolutionary stability dynamics of constitutive R0010 + E0240 (open red circles) and inducible R0010 + E0240 (solid red circles). Error bars represent one standard deviation from the mean of nine independently evolved populations. (C) Types of mutations in nine independently evolved populations. For nine independently evolved populations, colored boxes correspond to the mutation legend below the table. The most common mutation for a particular type of mutation is labeled with "1" in the boxes above and less common mutations are labeled with increasing numbers. (D) Most common loss-of-function mutations that inactivated the R0011 + E0240 and R0010 + E0240 inducible and constitutive circuits. See Additional File 1, Supplementary Table S1 for mutation details.

of *kanR* may have helped to avoid this growth deficiency.

Constitutive vs. inducible expression between two *lacI*-regulated circuits: loss-of-function mutations and evolutionary stability dynamics

We also propagated two *lacI*-regulated circuits (R0011 + E0240 and R0010 + E0240) with inducible vs. constitutive expression. We speculated that there might be a difference in stability since the constitutively expressed circuits may have a larger metabolic load. In the inducible circuits, the LacI protein may rebind to the DNA after some time and repress transcription. For inducible expression, we propagated the circuits in MG1655 Z1 strains that constitutively overexpresses *lacI* from its chromosome. Constitutively expressed circuits were propagated in the normal MG1655 strain that does not overproduce LacI. Figure 6 shows the regulation of the inducible circuits (Figure 6a), evolutionary stability dynamics for these re-engineered circuits (Figure 6b), mutations found in the nine evolved populations (Figure 6c), and most common mutation for each re-engineered circuit (Figure 6d).

For both circuits, evolutionary stability is increased on average when inducible expression is used compared to constitutive expression (Figure 6b). The evolutionary half-life of the R0011 + E0240 circuit under inducible control is about 32 generations vs. 20 for constitutive expression (one-tailed t-test, p = 0.0422). For R0010 + E0240, the inducible evolutionary half-life is 45 generations vs. 42 generations for constitutive expression (one-tailed t-test, p = 0.1083). Therefore, the evolutionary half-life is only significantly increased for one of the two circuits, but on average inducible expression increases evolutionary robustness. For the R0011 + E0240 circuit under inducible expression, by 30 generations the circuit is already about 4-fold higher even though it had a lower initial expression (Figure 6b, upper panel). After 60 generations, the circuit under constitutive expression has lost all function and the inducible circuit has 25% of its original expression level (Figure 6b, upper panel). Furthermore, the slopes are quite different between inducible and constitutive expression. The inducible R0010 + E0240 circuit is over 3-fold higher after 60 generations compared to the circuit under constitutive expression (Figure 6b, lower panel). The reason that inducible circuits may be more evolutionary robust than constitutive circuits is that there may a greater metabolic load to constantly express GFP. Perhaps in the inducible circuits, LacI eventually re-inhibits GFP expression and decreases this metabolic load.

The mutations in both circuits are similar for both inducible and constitutive expression (Figure 6c). For R0011 + E0240, all have the same mutation between repeated operator sequences (Figure 6c and 6d). For R0010 + E0240, only one mutation was observed in one inducible population, an IS element insertion (Figure 6c and 6d), indicating that differences in inducible vs. constitutive expression largely do not change the type of mutations that occur.

Evolutionary half-life vs. initial expression level in T9002, T9002-E, R0011 + E0240, and R0010 + E0240 circuits evolved with different inducer concentrations

From the results in previous sections (Figures 3, 4, 5, 6, Table 1), we noticed that circuits with a high initial expression level tended to have low evolutionary stability. Also, particular circuits with high mutation rates had lower stability compared to circuits with lower mutation rates. To test the relationship between initial expression level and evolutionary half-life directly, we evolved four circuits for up to 300 generations propagated with different inducer concentrations. We tested initial expression level vs. evolutionary half-life in T9002 (high mutation rate) vs. T9002-E (lower mutation rate) using different AHL concentrations. We also tested initial expression level vs. evolutionary half-life in R0011 + E0240 (high mutation rate) vs. R0010 + E0240 (lower mutation rate) using different IPTG concentrations in the Z1 strain.

The results of these experiments are shown in Figure 7. Figure 7a shows the mean initial expression level vs. mean evolutionary half-life for eight replicate populations from three different AHL concentrations in T9002 (black) and T9002-E (red). An exponential fit of these mean data points gives a much higher r^2 value than a linear fit (> 0.1) in both cases. T9002 has an r^2 value of 0.954 compared to the r^2 value of 0.955 in T9002-E. The curve for T9002 is shifted to the left from T9002-E due to its higher mutation rate (expression alone cannot account for the shift), but as expression is decreased the evolutionary half-life difference between these two circuits also decreases. This decrease may be because at high expression levels, the fitness difference between the progenitor and mutant cells is the highest, and therefore mutants outcompete functional cells in the population more quickly.

Figure 7b shows the mean initial expression level vs. mean evolutionary half-life for eight replicate populations from four different IPTG concentrations in R0011 + E0240 (black) and R0010 + E0240 (red). The R0011 + E0240 exponential curve has an r^2 value of 0.993 whereas the r^2 value of R0010 + E0240 is 0.992. These r^2 values are also both higher than a linear fit (> 0.02). Similar to the T9002 vs. T9002-E curves (Figure 7a), the curve for R0011 + E0240 is also shifted to the left due to its higher mutation rate and the evolutionary half-life difference decreases at lower expression levels. This difference is not as apparent as in Figure 7a, but the curves

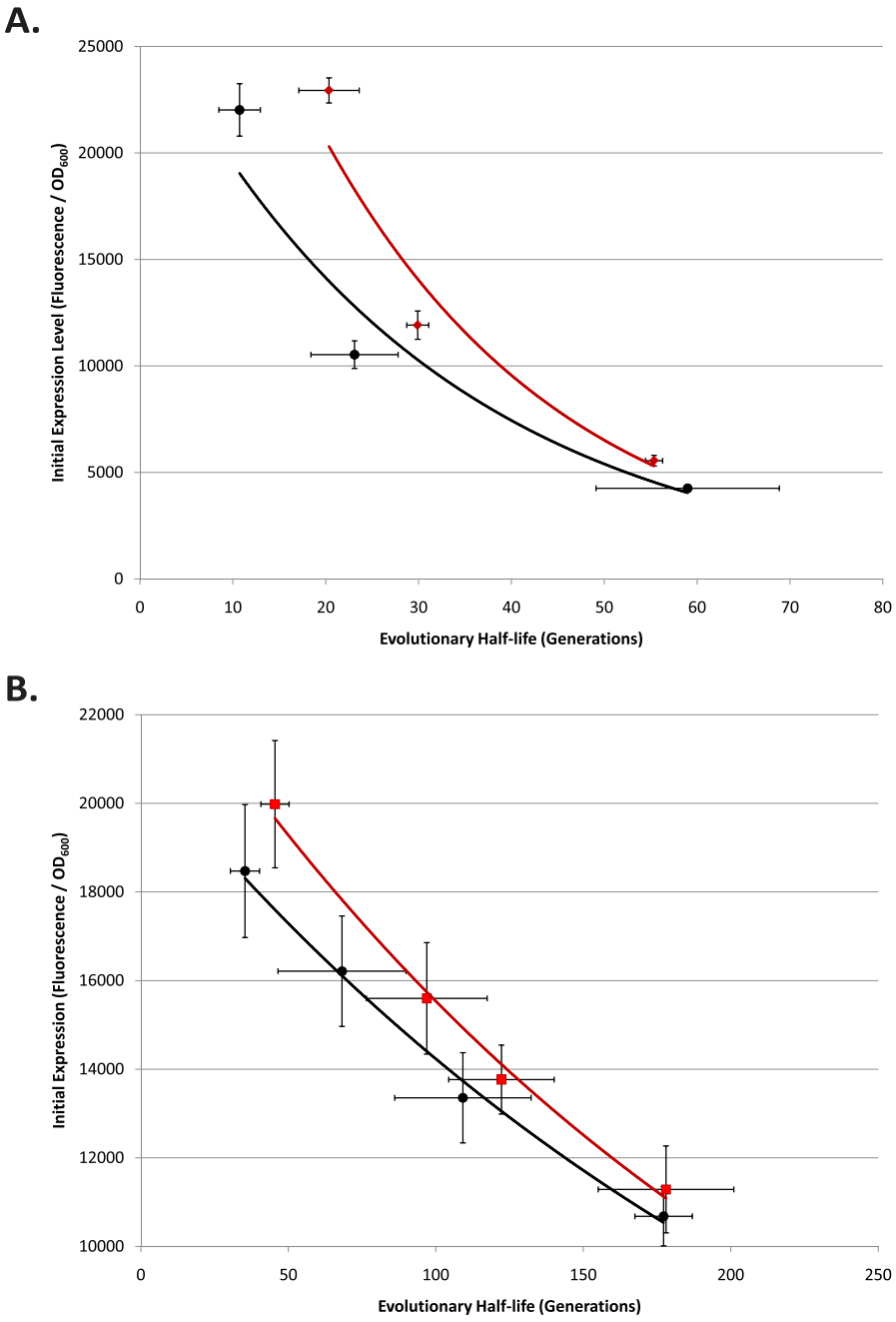


Figure 7 Evolutionary half-life vs. initial expression level in T9002, T9002-E, R0011 + E0240, and R0010 + E0240 circuits evolved with different inducer concentrations. (A) Evolutionary half-life vs. initial expression level is plotted in T9002 (solid black circles) and T9002-E (solid dark red diamonds) circuits evolved with different AHL concentrations. An exponential fit for the mean of each evolutionary half-life vs. initial expression data point is shown by the black line. Error bars for both the x and y-axis represent one standard deviation from the mean of eight independently evolved populations. (B) Evolutionary half-life vs. initial expression level is plotted in R0011 + E0240 (solid black circles) and R0010 + E0240 (solid red circles) circuits evolved with different IPTG concentrations. An exponential fit for the mean of each evolutionary half-life vs. initial expression data point is shown by the black line. Error bars for both the x and y-axis represent one standard deviation from the mean of eight independently evolved populations.

may have been more similar if there were more data points for the IPTG-induced populations at lower expression levels (evolutionary stability was relatively stable at 300 generations for these populations). The other striking difference between Figures 7a and 7b is that the T9002 and T9002-E circuits have a much lower evolutionary stability than the R0011 + E0240 and R0010 + E0240 circuits for any particular expression level. This result could be for a number of reasons that include mutation rate and fitness differences between functional and non-functional cells (due to circuit length, activator vs. repressor regulation, or something particular to the strong *luxR* promoter).

We also performed a regression on the individual T9002, T9002-E, R0011 + E0240, and R0010 + E0240 initial expression level vs. evolutionary half-life for independently evolved populations (Additional File 1, Supplementary Figures S2 and S3). While the r^2 values are not as high for individual data points compared to means (Figure 7) and range from about 0.6 to almost 0.9, the regressions are highly significant ($p < 0.0001$). For these regressions, an exponential fit is always higher than a linear fit in all cases, suggesting that on average, evolutionary half-life exponentially decreases with increasing expression levels. We also performed a regression on the initial expression level vs. evolutionary half-life for independent populations (shown in Figures 3, 4, 5, 6, Additional File 1, Supplementary Figures S4, S5, S6, S7, S8).

Discussion

Genetic circuits lose function over evolutionary time and are found to have a wide variety of mutations that cause their loss-of-function. Circuits with repeated sequences almost always have deletions between these sequences. These repeated sequences include transcriptional terminators, entire promoters, operator sequences within promoters, and occasionally between 8-bp scar sequences. In one re-engineered T9002 circuit, shortening the length of homology from 129 bp to 41 bp did not significantly increase evolutionary stability. Stability was only increased when there was no homology whatsoever between transcriptional terminators. Mutations between repeated sequences without perfect homology in the case of some re-engineered T9002 circuits are usually, but not always predictable.

In circuits without repeated sequences, mutations are more random between evolved replicate populations. Mutations that remove promoter function are most often selected for among all the genetic circuits tested. This result is likely because promoter mutations remove the metabolic load at both the transcriptional and translational levels. Mutations within RBSs are not found and mutations in coding sequences are rare except when that coding sequence is an activator of transcription downstream (as in the case of the *luxR* coding sequence in T9002). In the case of T9002, removing homology between transcriptional terminators only shifts the mutation to one that often removes function of the *luxR* promoter or *luxR* coding sequence instead. A similar story is seen with the KanR circuits. Even when a kanamycin resistance gene is inserted within the circuit and cells with this circuit are propagated with kanamycin to select for promoter function, mutations in the chromosome are selected for instead. Thus, without a selective pressure, removing the possibility of a mutation in one location only causes a mutation in another location. However, if this prevention lowers the mutation rate for a particular mutation, then evolutionary stability can be increased significantly, as shown for the T9002-E circuit.

What is needed is a method to predict the evolutionary stability of circuits from the properties of their parts, but the emergent behaviors of circuits will likely make prediction difficult. At the very least, publishing the evolutionary stability properties of simple circuits in future data sheets may allow engineers to calculate the expected evolutionary stability of more complex circuits. This calculation would likely require software (such as [42]) and mathematical modelling [43] that analyzes each part individually and the entire DNA sequence as a whole to determine the expected evolutionary stability. This calculation would also require standardization for methods to measure evolutionary stability and methods described here are not necessarily the best way. On the other hand, more general methods may be developed that focus less on design of the circuit and more on design of the environment to impose a selective pressure for function of the circuit [44]. Design of a selective environment is ideal, but is difficult to do when the output of most circuits (e.g., GFP) is not linked to survival or growth rate. A cell-sorter device that sorts between functional and non-functional cells may help with this issue, but may not be practical for performing routine experiments.

From our results of what types of mutations are selected for in genetic circuits and the evolutionary stability dynamics, a few simple design principles can be proposed when engineering circuits. The first principle is that high expression of genetic circuits comes with the cost of low evolutionary stability. Although exceptions to this rule certainly occur, a genetic circuit with high expression correlates with a large metabolic load and therefore is predicted to have decreased cellular fitness. When the fitness difference between the functional and non-functional cells in the population is large, evolutionary stability will decrease quickly. Therefore, the initial expression level of the circuit is likely to be a good predictor of evolutionary stability when a circuit with high mutational robustness is desired.

Using a low or medium-copy plasmid will help with stability as long as the expression level does not need to be high. For more complex circuits where a high expression level is needed for proper functioning of the circuit, decreasing expression level then comes at the cost of changing the function of the circuit.

The second design principle is to avoid repeated sequences. This principle may be obvious, but nearly every circuit in the Registry with more than one coding sequence has repeated B0015 terminators. When a circuit has a high metabolic load (higher than T9002) and repeated sequences on a high-copy number plasmid, the circuit will almost always lose function during overnight growth (unpublished results). Re-engineering the T9002 circuit to have two different transcriptional terminators (T9002-E) does significantly increase evolutionary half-life over 2-fold and is independent of expression levels. However, since this circuit has high expression, this improvement only results in an increase of about 10 generations. Decreasing the expression level along with the mutation rate will increase the evolutionary half-life about 17-fold, as seen in the T9002-F circuit. This result suggests simple ways to increase evolutionary stability can be used without changing function of the circuit. For more complex circuits, the community will need to identify many more terminators than those that currently exist in the Registry to design circuits without repeated sequences.

The third design principle is that use of inducible promoters generally increases evolutionary stability. This principle may or may not be significant depending on the circuit used. Inducible circuits are likely more stable due to decreased metabolic load and are preferred since expression can be controlled and fine-tuned, though in some circumstances a constitutive promoter may be desired. Therefore, the use of inducible promoters can be thought of one extra precaution to maximize evolutionary stability, but expression levels and repeated sequences should first be considered.

We emphasize that the design principles proposed may not be general since only relatively simple circuits were tested in this study. Evolutionary stability should be measured in larger and more complex circuits to understand if these design principles apply. Furthermore, these simple design principles should not necessarily be all used simultaneously. A researcher may not want only to design circuits that have low expression, have no repeated regions, and use a promoter that is inducible. For instance, if recombination sites are needed in the circuit, then repeated or inverted sequences may be impossible to avoid. Besides the design for the proper function of the circuit, design for evolutionary robustness should be carefully considered. For this, we need to think about the probability of mutations occurring when the expression level, and therefore metabolic load, is high. In this study, removing repeated regions often shifts mutations to the promoter, and putting a selection on the promoter often shifts the mutation to the chromosome.

Thus, mutations are unavoidable without a selective pressure, but evolutionary stability can likely be improved in the future by better design of selective environments where the circuit is linked to survival and/or growth rate, understanding of mutation rates in genetic circuits, fitness differences between functional and non-functional cells, and improvements to the host strain that decrease mutation rates or buffer metabolic loads more efficiently. Another way to improve evolutionary stability is to engineer an error detection and correction circuit that will correct mutations, but will need careful design since this circuit itself will be prone to mutation. Designing evolutionary robust genetic circuits therefore is somewhat of an artform at the moment besides a few simple design rules, but should be seen as something the engineer can eventually control.

Conclusions

A wide variety of loss-of-function mutations are observed in genetic circuits including point mutations, small insertions and deletions, large deletions, and insertion sequence (IS) element insertions that often occur in the scar sequence between parts. Promoter mutations are selected for more than any other biological part. Genetic circuits can be re-engineered to be more evolutionary robust with a few simple design principles: high expression of genetic circuits comes with the cost of low evolutionary stability, avoid repeated sequences, and the use of inducible promoters increases stability. Inclusion of an antibiotic resistance gene within the circuit does not ensure evolutionary stability.

Methods

Circuit engineering and use of strains

All circuits were either obtained from the Registry of Standard Biological Parts or engineered using the Clontech In-Fusion PCR Cloning Kit with the specific methods described in [19]. All circuits are encoded on the pSB1A2 plasmid, a high copy number plasmid (100-300 plasmids/cell) with an ampicillin resistance gene. Plasmids were transformed into strains via chemical transformation. *Escherichia coli* MG1655 was the strain used for constitutive expression and *Escherichia coli* MG1655 Z1 was used for inducible expression from *lacI*-regulated promoters since this strain is $lacI^q$ (overexpresses *lacI* from its chromosome).

Evolution experiment

For each engineered circuit, plasmid DNA that had been fully sequenced was transformed into either MG1655 or

MG1655 Z1 competent cells. Three individual transformant colonies were grown overnight at 37°C shaking at 250 RPM in + 100 μg/mL ampicillin and supplemented with 50 μg/mL kanamycin for KanR circuits. Freezer stocks were saved of these cultures in 15% glycerol and stored at -80°C. These freezer stocks were streaked out on LB + 100 μg/mL ampicillin plates with appropriate inducer (1×10^{-7} M AHL for T9002 circuits and 1×10^{-3} M IPTG for LacI-regulated promoters) or antibiotic (50 μg/mL kanamycin for KanR circuits) and grown overnight at 37°C. Three colonies were chosen from each transformant (nine total colonies) and inoculated into 1.5 mL LB + 100 μg/mL ampicillin media in Eppendorf deep-well plates sealed with a Thermo Scientific gas permeable membrane to allow for maximum oxygen diffusion. T9002 circuit cultures were supplemented with the inducer 1×10^{-7} M $3OC_6HSL$ (AHL). KanR circuit cultures were supplemented with 50 μg/mL kanamycin. Also, *lacI*-regulated circuits under inducible expression were supplemented with 1×10^{-3} M IPTG. T9002 and R0011 + E0240 were also evolved without inducer as controls. The R0011 + E0240 circuit with a mutation in the promoter was evolved to measure fluorescence background. Cultures were propagated with a serial dilution scheme using a 1:1000 dilution to achieve about 10 generations per day ($\log_2 1000 = 9.97$). Evolved populations were grown for 24 hours at 37°C shaking at 250 RPM. Freezer stocks (with 15% glycerol) of each population were saved at -80°C every day for future study. All replicate populations were evolved in parallel to minimize experimental variability.

Evolutionary stability measurements

Every 24 hours, cell density (OD_{600}) and fluorescence (excitation wavelength: 485/15, emission wavelength: 516/20) of evolved populations were measured in a Biotek Synergy HT plate reader. 24 hours was used as the measurement timepoint because the rate of change of GFP was closest to zero (i.e. closest to steady-state). Evolved populations thus spent about 8-12 hours in lag or exponential phase and the remaining time in stationary phase. For each timepoint, all populations were thoroughly mixed and 200 μl was transferred into black, clear-bottom 96-well plates (Costar). OD was subtracted from blank media to measure the OD without background. Fluorescence was subtracted from the R0011 + E0240 circuit with a mutation in the promoter to measure background fluorescence. Fluorescence was then divided by OD to measure the normalized expression ($Fluorescence/OD_{600}$).

Plasmid sequencing

At appropriate evolutionary timepoints, usually when circuit function had decreased to less than 10% of the original expression level, or 500 generations, the evolved populations were streaked out on LB + 100 μg/mL ampicillin plates, supplemented with 1×10^{-7} M AHL (for T9002 circuits), 50 μg/mL kanamycin (for KanR circuits), or 1×10^{-3} M IPTG (for *lacI*-regulated inducible circuits). Colonies were visualized for fluorescence on a Clare Chemical Dark Reader Transilluminator. Non-fluorescing colonies, or weakly fluorescing colonies if no non-fluorescing colonies were present, were grown overnight in 5 mL of LB + 100 ug/mL ampicillin. Plasmids were extracted using the Qiagen Miniprep Kit or glycerol stocks were sent to the University of Washington High-Throughput Genomics Unit facility http://www.htseq.org. Purified plasmid DNA was sequenced using the VF2 (5'-TGCCACCTGACGTCTAAGAA-3') and VR (5'-ATTACCGCCTTTGAGTGAGC-3') primers specific to the pSB1A2 vector (about 100 bp on either side of the circuit) or primers specific to the circuit.

Quantitative analysis of evolutionary half-life

Evolutionary half-life was calculated for each independently evolved population. First, the slope and y-intercept were calculated using the two normalized expression ($Fluorescence/OD_{600}$) data points on either side of the half maximum expression value on the evolutionary stability plot. A linear regression on those two data points was performed using the equation $y = ax + b$, where y = the half maximum initial expression, a = the slope of the two data points, b = the y-intercept of the two data points, and solving for x gives the evolutionary half-life. This method gave a very accurate half-life estimate in terms of generations and was a better estimate than using third-order polynomial equations which we also calculated.

Experiment to measure evolutionary half-life vs. initial expression level in T9002, T9002-E, R0011 + E0240, and R0010 + E0240 circuits evolved with different inducer concentrations

This experiment was performed as described in the "Evolution experiment" section above except that eight replicate populations were propagated with different inducer concentrations. The results of this experiment are shown in Figure 7. For the T9002 and T9002-E circuits, the AHL concentrations were 1×10^{-7} M (high expression level datapoint on the far left side of Figure 7a), 2×10^{-9} M (medium expression level), and 1×10^{-9} M (low expression level). For the R0011 + E0240 and R0010 + E0240 circuits, the IPTG concentrations were 5×10^{-5} M (high expression datapoint on the far left side of Figure 7b), 3×10^{-5} (medium-high expression level), 2×10^{-5} (medium expression level), and 1×10^{-5} (low expression level). The evolutionary half-life for individual evolved populations was determined as described

in the "Quantitative analysis of evolutionary half-life" section above. For each inducer concentration, the mean evolutionary half-life vs. initial expression level data point was plotted. These data points were fit to an exponential curve since this relationship always had the highest r^2 value.

Plasmid curing and re-transformation

To indirectly determine whether there are mutations on the chromosome, we first cured plasmids from evolved clones. Each evolved population was streaked out on to an LB plate and grown overnight at 37°C. Individual colonies were streaked on to both LB and LB + 100 μg/mL ampicillin plates where each colony was marked with a number. These plates were then grown overnight at 37°C. Colonies that were sensitive to ampicillin were grown overnight in LB and LB + 100 μg/mL ampicillin as a control to ensure sensitivity. These ampicillin sensitive cultures were made electrocompetent and saved in 15% glycerol stocks at -80°C. Plasmids were then re-transformed into these strains via electroporation and plated on selective media.

Additional material

Additional file 1: Supplementary Material. This file contains the genetic circuit mutations in all evolved populations, regressions of initial expression vs. evolutionary half-life measurements and additional experiments that test for mutations on the chromosome of evolved strains.

Acknowledgements

We thank members of the Sauro and Klavins labs for materials and useful discussions. A special thanks goes to Aparna Jorapur for performing the plasmid concentration experiments and to Lucian Smith, Wilbert Copeland, and Kyung Kim for valuable suggestions. We also thank three anonymous reviewers for valuable feedback. This research was funded by the Sauro Startup Funds (University of Washington), National Science Foundation (NSF) Grant in Theoretical Biology (0827592), and BEACON: An NSF Center for the Study of Evolution in Action (0939454).

Authors' contributions

SS conceived of the study, designed the experiments, engineered all the genetic circuits except for those described below, performed all the experiments, analyzed the DNA sequences, and analyzed all other data. BB engineered four of the six re-engineered T9002 circuits, JL engineered six of the eight I7101 circuits from a promoter library, and HS participated in the overall design of the study. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Received: 28 June 2010 Accepted: 1 November 2010
Published: 1 November 2010

References

1. Elowitz MB, Leibler S: **A synthetic oscillatory network of transcriptional regulators.** *Nature* 2000, **403(6767)**:335-338.
2. Gardner TS, Cantor CR, Collins JJ: **Construction of a genetic toggle switch in Escherichia coli.** *Nature* 2000, **403(6767)**:339-342.
3. Basu S, Gerchman Y, Collins CH, Arnold FH, Weiss R: **A synthetic multicellular system for programmed pattern formation.** *Nature* 2005, **434(7037)**:1130-1134.
4. Levskaya A, Chevalier AA, Tabor JJ, Simpson ZB, Lavery LA, Levy M, Davidson EA, Scouras A, Ellington AD, Marcotte EM, *et al*: **Synthetic biology: engineering Escherichia coli to see light.** *Nature* 2005, **438(7067)**:441-442.
5. Anderson JC, Clarke EJ, Arkin AP, Voigt CA: **Environmentally controlled invasion of cancer cells by engineered bacteria.** *J Mol Biol* 2006, **355(4)**:619-627.
6. Entus R, Aufderheide B, Sauro HM: **Design and implementation of three incoherent feed-forward motif based biological concentration sensors.** *Systems and Synthetic Biology* 2007, **1(3)**:119-128.
7. Stricker J, Cookson S, Bennett MR, Mather WH, Tsimring LS, Hasty J: **A fast, robust and tunable synthetic gene oscillator.** *Nature* 2008, **456(7221)**:516-519.
8. Friedland AE, Lu TK, Wang X, Shi D, Church G, Collins JJ: **Synthetic gene networks that count.** *Science* 2009, **324(5931)**:1199-1202.
9. Tsuruta H, Paddon CJ, Eng D, Lenihan JR, Horning T, Anthony LC, Regentin R, Keasling JD, Renninger NS, Newman JD: **High-level production of amorpha-4,11-diene, a precursor of the antimalarial agent artemisinin, in Escherichia coli.** *PLoS One* 2009, **4(2)**:e4489.
10. Steen EJ, Kang Y, Bokinsky G, Hu Z, Schirmer A, McClure A, Del Cardayre SB, Keasling JD: **Microbial production of fatty-acid-derived fuels and chemicals from plant biomass.** *Nature* 2010, **463(7280)**:559-562.
11. Bayer TS, Widmaier DM, Temme K, Mirsky EA, Santi DV, Voigt CA: **Synthesis of methyl halides from biomass using engineered microbes.** *J Am Chem Soc* 2009, **131(18)**:6508-6515.
12. Wang HH, Isaacs FJ, Carr PA, Sun ZZ, Xu G, Forest CR, Church GM: **Programming cells by multiplex genome engineering and accelerated evolution.** *Nature* 2009, **460(7257)**:894-898.
13. Gibson DG, Benders GA, Andrews-Pfannkoch C, Denisova EA, Baden-Tillson H, Zaveri J, Stockwell TB, Brownley A, Thomas DW, Algire MA, *et al*: **Complete chemical synthesis, assembly, and cloning of a Mycoplasma genitalium genome.** *Science* 2008, **319(5867)**:1215-1220.
14. Gibson DG, Glass JI, Lartigue C, Noskov VN, Chuang RY, Algire MA, Benders GA, Montague MG, Ma L, Moodie MM, *et al*: **Creation of a Bacterial Cell Controlled by a Chemically Synthesized Genome.** *Science* 2010.
15. Endy D: **Foundations for engineering biology.** *Nature* 2005, **438(7067)**:449-453.
16. Knight T: **Idempotent Vector Design for Standard Assembly of Biobricks.** *DSpace* 2003 [http://dspace.mit.edu/handle/1721.1/21168].
17. Canton B, Labno A, Endy D: **Refinement and standardization of synthetic biological parts and devices.** *Nat Biotechnol* 2008, **26(6)**.
18. Anderson JC, Dueber JE, Leguia M, Wu GC, Goler JA, Arkin AP, Keasling JD: **BglBricks: A flexible standard for biological part assembly.** *J Biol Eng* 2010, **4(1)**:1.
19. Sleight SC, Bartley BA, Lieviant JA, Sauro HM: **In-Fusion BioBrick assembly and re-engineering.** *Nucleic Acids Res* 2010, **38(8)**:2624-2636.
20. You L, Cox RS, Weiss R, Arnold FH: **Programmed population control by cell-cell communication and regulated killing.** *Nature* 2004, **428(6985)**:868-871.
21. Balagadde FK, You L, Hansen CL, Arnold FH, Quake SR: **Long-term monitoring of bacteria undergoing programmed population control in a microchemostat.** *Science* 2005, **309(5731)**:137-140.
22. Arkin AP, Fletcher DA: **Fast, cheap and somewhat in control.** *Genome Biol* 2006, **7(8)**:114.
23. Glick BR: **Metabolic load and heterologous gene expression.** *Biotechnol Adv* 1995, **13(2)**:247-261.
24. Dekel E, Alon U: **Optimality and evolutionary tuning of the expression level of a protein.** *Nature* 2005, **436(7050)**:588-592.
25. Cooper VS, Lenski RE: **The population genetics of ecological specialization in evolving Escherichia coli populations.** *Nature* 2000, **407(6805)**:736-739.
26. Cooper VS, Schneider D, Blot M, Lenski RE: **Mechanisms causing rapid and parallel losses of ribose catabolism in evolving populations of Escherichia coli B.** *J Bacteriol* 2001, **183(9)**:2834-2841.
27. Lovett ST: **Encoded errors: mutations and rearrangements mediated by misalignment at repetitive DNA sequences.** *Mol Microbiol* 2004, **52(5)**:1243-1253.

28. Bzymek M, Lovett ST: **Instability of repetitive DNA sequences: the role of replication in multiple mechanisms.** *Proc Natl Acad Sci USA* 2001, **98(15)**:8319-8325.
29. Bi X, Liu LF: **recA-independent and recA-dependent intramolecular plasmid recombination. Differential homology requirement and distance effect.** *J Mol Biol* 1994, **235(2)**:414-423.
30. Smolke CD, Keasling JD: **Effect of gene location, mRNA secondary structures, and RNase sites on expression of two genes in an engineered operon.** *Biotechnol Bioeng* 2002, **80(7)**:762-776.
31. Stueber D, Bujard H: **Transcription from efficient promoters can interfere with plasmid replication and diminish expression of plasmid specified genes.** *EMBO J* 1982, **1(11)**:1399-1404.
32. Kaur J, Rajamohan G, Dikshit KL: **Cloning and characterization of promoter-active DNA sequences from Streptococcus equisimilis.** *Curr Microbiol* 2007, **54(1)**:48-53.
33. Kelly JR, Rubin AJ, Davis JH, Ajo-Franklin CM, Cumbers J, Czar MJ, de Mora K, Glieberman AL, Monie DD, Endy D: **Measuring the activity of BioBrick promoters using an in vivo reference standard.** *J Biol Eng* 2009, **3**:4.
34. Ohtsubo Y, Genka H, Komatsu H, Nagata Y, Tsuda M: **High-temperature-induced transposition of insertion elements in burkholderia multivorans ATCC 1761.** *Appl Environ Microbiol* 2005, **71(4)**:1822-1828.
35. Twiss E, Coros AM, Tavakoli NP, Derbyshire KM: **Transposition is modulated by a diverse set of host factors in Escherichia coli and is stimulated by nutritional stress.** *Mol Microbiol* 2005, **57(6)**:1593-1607.
36. de Visser JA, Akkermans AD, Hoekstra RF, de Vos WM: **Insertion-sequence-mediated mutations isolated during adaptation to growth and starvation in Lactococcus lactis.** *Genetics* 2004, **168(3)**:1145-1157.
37. Eichenbaum Z, Livneh Z: **UV light induces IS10 transposition in Escherichia coli.** *Genetics* 1998, **149(3)**:1173-1181.
38. Sleight SC, Orlic C, Schneider D, Lenski RE: **Genetic basis of evolutionary adaptation by Escherichia coli to stressful cycles of freezing, thawing and growth.** *Genetics* 2008, **180(1)**:431-443.
39. Schneider D, Lenski RE: **Dynamics of insertion sequence elements during experimental evolution of bacteria.** *Res Microbiol* 2004, **155(5)**:319-327.
40. Engler JA, van Bree MP: **The nucleotide sequence and protein-coding capability of the transposable element IS5.** *Gene* 1981, **14(3)**:155-163.
41. Schoner B, Kahn M: **The nucleotide sequence of IS5 from Escherichia coli.** *Gene* 1981, **14(3)**:165-174.
42. Chandran D, Bergmann FT, Sauro HM: **TinkerCell: modular CAD tool for synthetic biology.** *J Biol Eng* 2009, **3**:19.
43. Chandran D, Copeland WB, Sleight SC, Sauro HM: **Mathematical modeling and synthetic biology.** *Drug Discovery Today* 2009, **5(4)**:299-309.
44. Bayer TS, Hoff KG, Beisel CL, Lee JJ, Smolke CD: **Synthetic control of a fitness tradeoff in yeast nitrogen metabolism.** *J Biol Eng* 2009, **3**:1.

doi:10.1186/1754-1611-4-12
Cite this article as: Sleight *et al.*: Designing and engineering evolutionary robust genetic circuits. *Journal of Biological Engineering* 2010 **4**:12.

4.3 Controlling *E. coli* Gene Expression Noise

IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, VOL. 9, NO. 4, AUGUST 2015 497

Controlling *E. coli* Gene Expression Noise

Kyung Hyuk Kim, Kiri Choi, Bryan Bartley, and Herbert M. Sauro

***Abstract*—Intracellular protein copy numbers show significant cell-to-cell variability within an isogenic population due to the random nature of biological reactions. Here we show how the variability in copy number can be controlled by perturbing gene expression. Depending on the genetic network and host, different perturbations can be applied to control variability. To understand more fully how noise propagates and behaves in biochemical networks we developed stochastic control analysis (SCA) which is a sensitivity-based analysis framework for the study of noise control. Here we apply SCA to synthetic gene expression systems encoded on plasmids that are transformed into *Escherichia coli*. We show that (1) dual control of transcription and translation efficiencies provides the most efficient way of noise-versus-mean control. (2) The expressed proteins follow the gamma distribution function as found in chromosomal proteins. (3) One of the major sources of noise, leading to the cell-to-cell variability in protein copy numbers, is related to bursty translation. (4) By taking into account stochastic fluctuations in autofluorescence, the correct scaling relationship between the noise and mean levels of the protein copy numbers was recovered for the case of weak fluorescence signals.**

***Index Terms*—Gene expression noise, noise control, stochastic control analysis, stochasticity, synthetic biology, two state model.**

I. Introduction

CELL-TO-CELL variability in protein copy numbers within isogenic populations are typically observed in various types of cells due to underlying random biochemical reaction processes [1]–[3]. The variability can lead to noise-induced cellular phenotypes such as cellular differentiation [3], multiple stability [4], and either sensitivity enhancement or suppression [5], [6]. Here we investigate the ability to differentially control the noise and mean levels of gene expression in *E. coli*.

Such differential control in gene expression has been achieved in different organisms such as yeast [7]–[9], soil bacteria [10], and mammalian cell lines [11]. In *E. coli*, systematic noise control have not been performed by perturbing promoter DNA sequences and ribosome binding sites, while most studies have been focused on genome-wide expression without such perturbations [2], [16], [21]. Here we aim to understand differential control of mean and noise levels of protein concentrations at the single cell levels of *E. coli*.

Manuscript received January 12, 2015; revised June 23, 2015; accepted June 28, 2015. Date of publication September 10, 2015; date of current version September 22, 2015. This work was supported by the National Science Foundations (NSF MCB 1158573). This paper was recommended by Associate Editor R. Sarpeshkar.

The authors are with the Department of Bioengineering, University of Washington, Seattle, WA 98195 USA (e-mail: kkim@uw.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TBCAS.2015.2461135

The approach we use is based on stochastic control analysis (SCA) [12], [13], a body of theory we developed and reported in previous publications. SCA is a sensitivity analysis framework, that is a direct extension of metabolic control analysis [14], [15] to the stochastic regime [13]. This approach is based on a *local* sensitivity analysis that can be applied to study first-order effects of finite-size perturbations. SCA can identify which parameters in stochastic systems—here, gene regulatory circuits—need to be varied by how much to achieve a desired control aim. This includes orthogonal control of noise levels with respect to mean levels, and simultaneous changes in noise and mean levels in the same or opposite directions for the same or different protein species. SCA can provide control efficiency and strength to identify the most effective control schemes that are experimentally relevant [12]. Here, we apply SCA experimentally to *E. coli* genetic systems.

In this paper, gene circuits are encoded on plasmid backbones, which are transformed into *E. coli* MG1655. The circuits express green fluorescent proteins (GFP) under the lac-promoter. We perturbed the expression system by inducing the promoter with isopropyl β-D-1-thiogalactopyranoside (IPTG) and using a library of ribosome binding sites (RBS). We found that by taking into account stochastic fluctuations in autofluorescence, scaling relationship between GFP signal noise and mean levels can be extended to weak signal regions, where autofluorescence becomes moderately strong. This implies that when fluorescent signals are not strong enough compared to autofluorescence, stochasticity in autofluorescence can be systematically taken into account to characterize cellular systems. In addition, we aimed to understand what the major sources of GFP signal noise are by investigating the scaling relationship between the GFP signal noise and mean levels via promoter induction and RBS perturbation. We found that one of the major noise sources is bursty translation.

II. Stochastic Control Analysis: Review

SCA [12], [13] is a local sensitivity analysis based on control coefficients, which are defined approximately as percentage change in a response signal (y) divided by the percentage change in a system parameter (p)

$$C_p^y = \frac{p}{y}\frac{dy}{dp} = \frac{d\log y}{d\log p}.$$

We note that the slope in the log-log plot of y versus p corresponds to C_p^y. The response signal can be the mean or noise levels of mRNAs or proteins. The parameters can include transcription and translation efficiencies, degradation rates of mRNAs and proteins, dilution rate due to cell growth, and reaction rates of transcription-factor binding and unbinding from promoter regions, etc. Another important quantity in

SCA, is the control vector, each element of which corresponds to a control coefficient for a given response signal (y)

$$\mathbf{C}_{\mathbf{p}}^{y} = \left(C_{p_1}^{y}, C_{p_2}^{y}, \cdots, C_{p_N}^{y}\right)$$

where N defines the number of parameters (dimension of the parameter space) that will be varied to control the value of y. In this paper, we are mostly interested in dual control of transcription and translation efficiencies, i.e., $N = 2$. One of the important properties of the control vector is that its inner-product with a parameter perturbation vector $\delta\mathbf{p}$ becomes the amount of change in the response signal δy

$$\mathbf{C}_{\mathbf{p}}^{y} \cdot \frac{\delta \mathbf{p}}{\mathbf{p}} = \frac{\delta y}{y}.$$

We can quantify which parameter value, and by how much it should be controlled to achieve specific control aims. For example, consider a case where the noise level of a protein needs to be reduced by 9%, while its mean level should remain the same. Here, the noise level (n) is defined by the variance divided by the squared mean value, i.e., squared coefficient of variation. Two control vectors for the noise and mean levels, $\mathbf{C}_{\mathbf{p}}^{n}$ and $\mathbf{C}_{\mathbf{p}}^{m}$, need to be computed based on a given mathematical model. System parameters need to be perturbed while satisfying

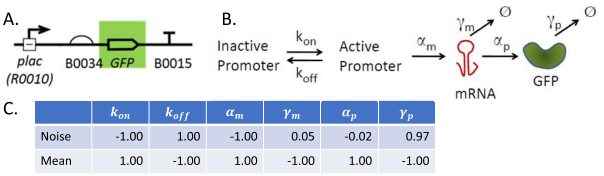
$$\frac{\delta n}{n} = \mathbf{C}_{\mathbf{p}}^{n} \cdot \frac{\delta \mathbf{p}}{\mathbf{p}} = -0.09 \text{ and } \frac{\delta m}{m} = \mathbf{C}_{\mathbf{p}}^{m} \cdot \frac{\delta \mathbf{p}}{\mathbf{p}} = 0.$$

The perturbation vector $\delta\mathbf{p}/\mathbf{p}$ satisfying these two equations can be solved, but the solutions can be infinite. In that case, it is important to select the optimal control scheme (perturbation vector) among the possible solutions. For this, the control efficiency and strength were introduced [12]. Based on these two quantities, one can choose desired control schemes that are appropriate to systems of interest with the maximum control strength and/or efficiency.

III. SCA for a Single Gene Expression Cassette

We constructed plasmid expression systems that express green fluorescent protein (GFP) under *lac*-promoters in *E. coli* [Fig. 1(a)]. The plasmid copy number in a single cell fluctuates in time because a set of plasmids are randomly partitioned during cell division and are synthesized in a stochastic fashion. Thus, the copy number of *lac*-promoters per cell fluctuates. For simplicity, we will assume that the plasmid copy number is tightly controlled, i.e., constant at the first level of approximation. The total number of *plac* will be the sum of the number of inactive and active *lac*-promoters [Fig. 1(b)], which will be set to a constant, N_p. We call this the two-state model. The plasmid backbone that we used is pGA3K3 with the replication origin, p15A ($N_p = 10 - 30$). Based on this two-state model, we computed control vectors for the mean and noise levels of GFP fluorescence as shown in Fig. 1(c) (refer to the Materials and Methods and [12] for the control vector computation).

The computed control coefficients show that noise can be controlled efficiently by varying k_{on}, k_{off}, α_m or γ_p; $C_{\mathrm{k}_{on}}^{n} = C_{\alpha_m}^{n} = -1.00$, $C_{\mathrm{k}_{off}}^{n} = 1.00$ and $C_{\gamma_p}^{n} = 0.97$, indicating that, for example, with an increase in α_m by 10%, n will reduce by 10% (this is a first-order approximation, because control coefficients are defined locally). Similarly, with an increase in γ_p by 10%, n will increase by 9.7%. For the mean level (m), any model parameter will efficiently change m, because the absolute values of all the control coefficients for m are equal to one.



	k_{on}	k_{off}	α_m	γ_m	α_p	γ_p
Noise	-1.00	1.00	-1.00	0.05	-0.02	0.97
Mean	1.00	-1.00	1.00	-1.00	1.00	-1.00

Fig. 1. GFP expression system. (a) GFP is expressed under the *lac*-promoter (BioBrick part BBa_R0010) with a ribosome binding site (BBa_B0034). This expression cassette was placed in a low-medium copy number plasmid backbone (pGA3K3; origin of replication p15A). The 'T' symbol represents a terminator (double terminator used here to ensure transcription termination). (b) Two-promoter-state model. When the promoter is active, mRNA is transcribed with a rate constant α_m. From the transcript, GFP is translated with a rate constant α_p. mRNA and GFP degrade or are diluted with net rate constants γ_m and γ_p, respectively. (c) Control coefficients for noise and mean levels are listed. All control coefficients in the same row add up to zero (up to rounding error), satisfying summation theorems in SCA [13]. Parameters (unit: hr^{-1}): $k_{on} = 50$, $k_{off} = 51000$, $\alpha_m = 160$, $\gamma_m = 30$, $\alpha_p = 1400$, and $\gamma_p = 1$ (refer to the Material and Methods for the detailed description of the mathematical model and its parameters).

IV. Mean Level Control

From the computed control coefficients, the mean protein levels can be controlled without changing the noise level (with a minor change, $\sim$10 folds less than the change in the mean levels) by varying either α_p or γ_m. To confirm this theoretical prediction, we changed the translation efficiency α_p by using a library of both ribosome binding sites (RBSs) and spacer sequences as shown in Fig. 2. Among them, four different spacers—TACTAG, AAAAAA $= (\mathrm{A})_6$, $(\mathrm{A})_{10}$, and $(\mathrm{A})_{13}$—that are placed between B0034 and the start codon showed distinct GFP expression levels when *plac* is fully active ($[\mathrm{IPTG}] = 1$ mM). Here, the introduced spacer sequences are presumed to change ribosome binding affinity, in particular, translation initiation—the limiting step for a translation rate [23], [24]. We note that strong RBSs can recruit many ribosomes to mRNAs, causing an implication depending on the availability of ribosomes. This can apply an upper limit in the value of α_p.

Based on our flow cytometry data, the mean level was successfully varied by using different spacer sequences as shown in Fig. 3 and Fig. 4(a). We compared three different cases: Points A, B, and C in Fig. 3, corresponding to $[\mathrm{IPTG}] = 1$ mM. As shown in Fig. 4(a), the rescaled probability density functions (pdfs) were overlapped with a minor discrepancy. This scale invariance confirms that the noise levels of all the points are the same.

1) Scale Invariance in the Gamma Distribution: Furthermore, the observed invariance implies a special property that we need to consider carefully. This invariance property is satisfied by the gamma distribution function as shown in the Materials and Method section when the burst size is rescaled together. This implies that the difference between the system parameters of Points A, B, and C is only the burst size. For these Points, different spacer sequences were used between B0034 and the start codon, while the *lac*-promoter was fully induced (saturated).

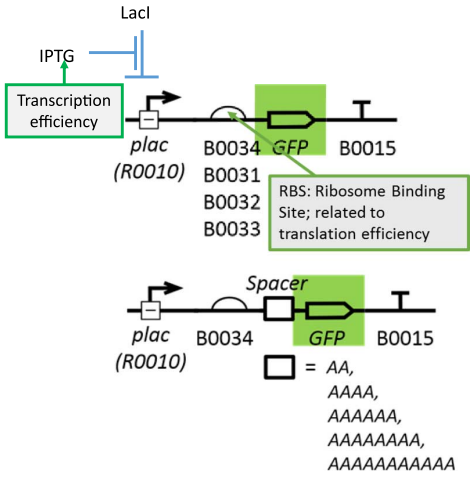


Fig. 2. Perturbations in the GFP expression systems. The GFP expression cassette is placed in the plasmid backbone pGA3K3 in *E.coli* MG1655Z1 that constitutively expresses LacI. IPTG concentrations were varied for a given complex of ribosome binding site and spacer. ∼10 fold increase in the GFP noise level can be achieved without changing its mean level.

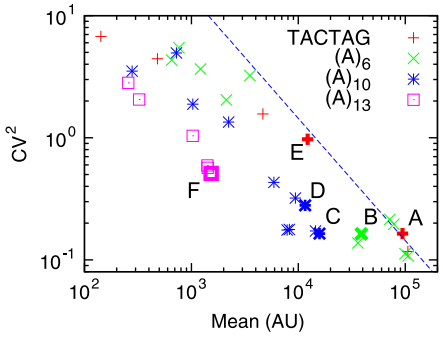


Fig. 3. Scaling relationship between noise and mean levels. Four different cases of spacer sequences between the ribosome binding site BBa_B0034 and the start codon are shown. The same symbol represents the same spacer with different [IPTG]. The noise levels (squared coefficient of variation) are inversely proportional to the mean level. For comparison, a line with a slope −1 is drawn. The contribution to the noise level by background fluorescence signals was removed via the noise level correction method (Materials and Methods). For the IPTG concentration information, we refer to the Supplementary Notes Fig. 2.

Thus, the translation rate constant α_p is expected to be varied for these three Points and the burst size must be closely related to α_p, which is consistent with theoretical prediction based on our model [Eq. (3)]. This result supports that the observed distribution functions are the gamma distributions (confer to [25], [26] about claims for other types of distribution functions). To confirm this, we fit the GFP pdfs to the gamma distribution functions as shown in Fig. 5. We confirmed that the pdfs follow the gamma distributions well.

V. Noise Level Control

As discussed above, the noise level can be efficiently controlled by varying k_{on}, k_{off}, α_m and γ_p. However, when these parameters are changed, the mean level also changes with the same fold difference but in the opposite direction; for example, in Fig. 1(c), $C^n_{\alpha_m}$ and $C^m_{\alpha_m}$ are -1.00 and 1.00, meaning that

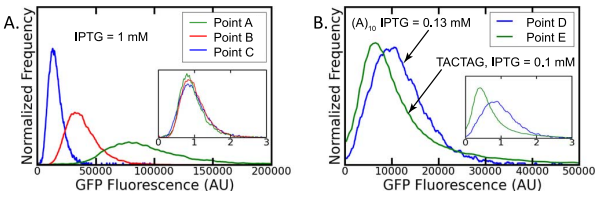


Fig. 4. Probability density functions (pdfs) of GFP fluorescence signals measured from a flow cytometer. (a) Orthogonal mean level control: Points A, B, and C in Fig. 3 correspond to [IPTG] = 1 mM. In the inset plot, both the pdfs were re-scaled by the mean values of their respective GFP fluorescence signals, so that the transformed pdfs are centered around one. (b) Orthogonal noise level control: Points D and E. Both the [IPTG] and the spacer sequences were varied. [IPTG] = .13 mM for Point D and .1 mM for Point E. Autofluorescence was removed via the fluorescence histogram correction method (Materials and Methods).

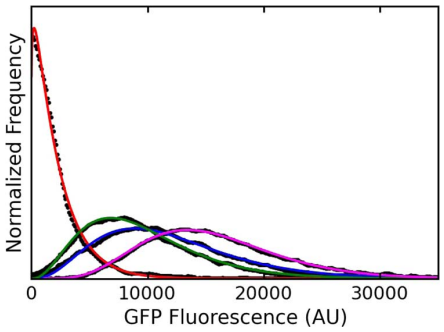


Fig. 5. True GFP signal distribution function for the $(A)_{10}$ cases with different IPTG concentrations: The fluorescence histogram correction was applied to remove autofluorescence effects. The true GFP signal distribution satisfies the Gamma distribution functions.

when α_m is increased by x%, the noise level decreases by x%, while the mean level increases by x%. Thus, to change the noise level without changing the mean level, we must vary at least two different parameters simultaneously.

Since the mean level can be controlled almost independently of the noise level by changing α_p, we will vary α_p along with one of the parameters in $\{k_{on}, k_{off}, \alpha_m\}$ to compensate for the change. The reason that we did not choose to vary γ_p is that this parameter is highly dependent on cell growth rate, rather than protein degradation in *E. coli*; GFP lifetime is much longer than the cell doubling time ∼1 hr in M9 media.

Based on the SCA, an individual change in k_{on}, k_{off}, and α_m and any combination of the individual changes can vary the noise and mean levels while satisfying the same scaling law: $n = c/m$ with c a constant (not varied). We note that the ratio of control coefficients for n and m for a given parameter, e.g., k_{on}, has a graphical meaning: In Fig. 3, when [IPTG] is varied, the corresponding data point shifts (e.g., Point C → D) and the slope of the shift in the log-log plot corresponds to the ratio of the control coefficients: $C^n_{k_{on}}/C^m_{k_{on}} = (d\log n/d\log m)|_{k_{on}}$. For k_{on}, k_{off}, and α_m, the ratios are the same: $(C^n_{k_{on}}/C^m_{k_{on}}) = (C^n_{k_{off}}/C^m_{k_{off}}) = (C^n_{\alpha_m}/C^m_{\alpha_m}) = -1$. This implies that the directions of data point shifts in the log-log plot of n versus m are identical for each individual perturbation of k_{on}, k_{off}, and α_m, and thus for any combination of these three individual perturbations. Therefore, the shift of data points with the slope

of -1, observed when varying IPTG concentrations as shown in Fig. 3, cannot determine which parameters among k_{on}, k_{off}, and α_m were affected by IPTG concentration changes. We note that in [21] promoter perturbations in *E. coli* was claimed to affect k_{off} only.

As shown in Fig. 3, by using a library of RBS as well as different concentrations of IPTG, the noise level was controlled and $\sim$10 fold change in the noise level was achieved without changing the mean level. What is the biological reason that noise can be increased in this way? In other words, what causes to increase the value of the Fano factor? In the scaling relationship, $n = c/m$, c is the Fano factor, which is expressed for the case of *E. coli* (Materials and Methods)

$$c \simeq 1 + b(1 + b_m)$$

where

$$b = \frac{\alpha_p}{\gamma_m}$$

and

$$b_m = \frac{\alpha_m}{k_{on} + k_{off}} \frac{k_{off}}{k_{on} + k_{off}}.$$

b is the translational burst size, quatifying the number of proteins that are synthesized from a single mRNA during the mRNA lifetime $(1/\gamma_m)$. b_m is the transcriptional burst size, quantifying the number of mRNA that are synthesized per plasmid during the time-scale $(1/(k_{on} + k_{off}))$ of gene switching (refer to the Materials and Methods). The Fano factor depends on both transcriptional and translational bursts. In our case of lacI^q expression of LacI, we can neglect the transcriptional burst (Materials and Methods). Thus, the Fano factor becomes $c = 1 + b$, and n can be expressed as

$$n \simeq \frac{1 + b}{m}. \tag{1}$$

The Fano factor can be increased by applying stronger translation efficiencies (from Point C to A) and remains the same by decreasing [IPTG] (from Point A to E), leading to the increase in the noise level without changing the mean level.

The translational bursts lead to longer-tail pdfs, more precisely, higher cutoff values in the pdfs (in the gamma distribution, there is an exponential factor $e^{-x/b}$ and b acts as a cutoff value): Fig. 4(b) shows that a longer tail in the GFP pdf can be generated by using stronger translation efficiency (Point D $\rightarrow$ Point E).

Another interpretation for the observed longer tail in Fig. 4(b) is that the major source of fluctuations in the protein copy numbers is in mRNA copy numbers, which merely get amplified by the translation rate in a non-bursty way

$$N_{pr}(t) \sim \left(\frac{\alpha_p}{\gamma_p}\right) N_{rna}(t).$$

The variance in protein expression levels becomes

$$\mathrm{Variance}(N_{pr}) \sim \left(\frac{\alpha_p}{\gamma_p}\right)^2 \mathrm{Variance}(N_{rna})$$

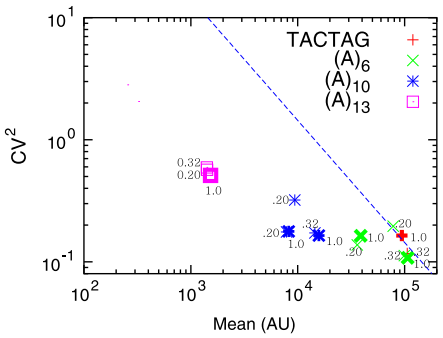


Fig. 6. GFP mean and noise levels for the cases that *lac*-promoters are fully induced ([IPTG] = 0.20, 0.32, 1.0 mM). Two biological replicates were used for $(A)_6$ and $(A)_{10}$.

resulting in that the noise level does not depend on α_p

$$n = \frac{\mathrm{Variance}(N_{pr})}{\mathrm{Mean}(N_{pr})^2} \sim \frac{\mathrm{Variance}(N_{rna})}{\mathrm{Mean}(N_{rna})^2}.$$

Since the protein mean level m must be proportional to the translation rate constant α_p (i.e., $m = \beta\alpha_p$ with β a constant), we obtain again similar scaling relationship

$$n \text{ is independent of } \alpha_p. \Leftrightarrow n \propto \frac{\alpha_p}{\beta\alpha_p} = \frac{\alpha_p}{m}.$$

The Fano factor again increases with α_p. Therefore, based on the scaling relationship alone, it is difficult to differentiate whether the translation is bursty or not.

VI. Translation is Bursty

We claim that translation processes are bursty. The data points in bold in Fig. 6 correspond to different RBS strength but the same level of [IPTG] equal to 1 mM, where the *lac*-promoter becomes constitutively active. The noise values for TACTAG, $(A)_6$, and $(A)_{10}$ were similar, but the noise level for $(A)_{13}$ was higher than the rest. This difference cannot be explained in the non-bursty translation scenario, because n should be independent of α_p, i.e., RBS strength. In the bursty translation scenario, the noise level can be dependent on the value of α_p, especially when the value of α_p is similar to that of γ_m. Since m is proportional to α_p $(m = \beta\alpha_p)$, Eq. (1) becomes

$$n = \frac{1 + \alpha_p/\gamma_m}{\beta\alpha_p} = \frac{1}{\beta\alpha_p} + \frac{1}{\beta\gamma_m}. \tag{2}$$

For a strong RBS such as the TACTAG case, α_p can be roughly around 1400 hr^{-1} (Materials and Methods). In this case, $\alpha_p/\gamma_m \sim 1400/30 \simeq 47$, i.e. much larger than 1. Thus, Eq. (2) becomes $n \simeq ((\alpha_p/\gamma_m)/\beta\alpha_p) = 1/\gamma_m\beta$, resuling in that n is independent of α_p, which is what we observed from Point A to C. For the case of $(A)_{13}$ (Point F), the RBS strength is reduced by $\sim$60 times (by comparing the mean levels of Point A and F) and $\alpha_p/\gamma_m \simeq 47/60 \simeq 0.8$. Thus, n becomes dependent on α_p [Eq. (2)]. As α_p decreases, n increases. This is consistent with our observation.

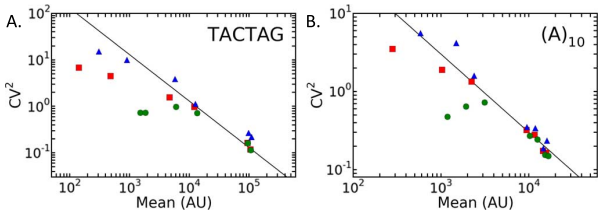


Fig. 7. Effect of autofluorescence on scaling relationship between n and m: Autofluorescence was removed in two different ways via (1) the noise level correction method (red squares) and (2) the fluorescence histogram correction method (blue triangles) (refer to the Materials and Methods). To show the trend clearly, the data corresponding to the same biological replicate are only shown.

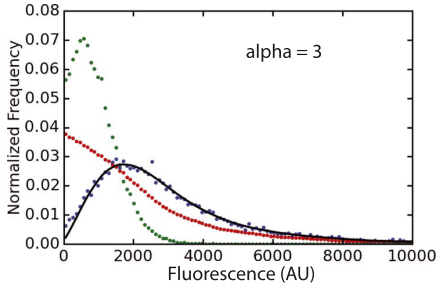


Fig. 8. Autofluorescence compensation: The green dots correspond to autofluorescence (IPTG $=$ 0 case), the blue dots to the measured GFP signals, and the red dots to the optimized solution S, i.e., the pdf of the true GFP signals. The black line is to verify the optimized solution S can generate the measured GFP pdfs via convolution (refer to the Materials and Methods).

VII. Scaling Relationship Between the Noise and Mean Levels

Fig. 7 shows that the scaling relationship $n = c/m$ can be observed after autofluorescence was systematically removed, even for the small mean value region, where the autofluorescence interferes with the true GFP signals. We took into account the stochasticity in autofluorescence and assumed that the fluctuations in the autofluorescence signals are statistically independent of the true GFP signals. Under this assumption, we compensated for the autofluorescence effect in two different ways: (1) direct noise level correction (red squares in Fig. 7) and (2) fluorescence histogram correction (blue triangles in Fig. 7; an example is presented in Fig. 8) (Materials and Methods). This implies that it is important to take into account the stochasticity of the autofluorescence signals when characterizing the systems by using the pdfs of fluorescence signals.

VIII. Conclusions

In summary, we perturbed the strength of ribosome binding sites and investigated scaling relationship between the mean and noise levels of the expressed proteins. We confirmed that translational bursts are one of the important sources of noise at the protein level by using our numerical sensitivity analysis method, SCA, and the analytical structure of noise propagation. To investigate the scaling relationship further in detail, we compensated the effect of autofluorescence by taking into account stochasticity in the autofluorescence and recovered the expected scaling relationship even when autofluorescence becomes moderately strong. This shows that the autofluorescence can be systematically removed and its compensation can be applied to characterize cellular systems.

Appendix
Materials and Methods

A. GFP Expression Circuits and Strains

All genetic components used in this manuscript are BioBrick parts, from which genetic circuits were constructed by using the Gibson assembly method [30]. The constructed circuits were integrated into a low-to-medium copy number plasmid pGA3K3 with a Kanamycin resistance gene and *Escherichia coli* MG1655 Z1 was transformed with the plasmids. The strain (lacI^q) constitutively overexpresses LacI from its chromosome.

B. Cell Growth and Flow Cytometry Measurements

E. coli strains were grown to OD600 $\sim$ 0.2 in 2 mL Luria-Bertani (LB) media (Becton Dickinson) with kanamycin 50 μg/mL at 37°C and 300 rpm in a shaker. The cultures were diluted 1:200 into 200 μL prewarmed fresh M9 media (Teknova 2M1990) in 96-well plates (Costar 3904) with kanamycin 50 μg/mL. 12 different IPTG concentrations (0 mM, 0.02 $\sim$ 1 mM) were used for each well (refer to the Supplementary Notes for more detailed information on IPTG concentrations) and grown to OD600 $=$ 0.3–0.4 in a shaker (37°C, 300 rpm). For the flow cytometry measurements, the grown cultures were diluted 1:4 in 1xPBS. A Sony Biotechnology ec800 flow cytometer was used with a 525 nm filter and a 488 nm excitation laser for GFP fluorescence. 100,000 events were collected for each sample and gated by using a 2-D normal distribution (Bioconductor flowCore norm2filter function with scale.factor $=$ 1) [31] within the R software environment as well as by using python package FlowCytometryTools (http://gorelab.bitbucket.org/flowcytometrytools/#). To prevent well-well contamination we executed a Medium Flush cycle after each sample well. When computing the mean and noise levels of GFP signals, background fluorescence was removed by using the mean and noise levels of GFP signals, or the signal histogram for the case without IPTG for each different gene circuit.

C. Mathematical Model

A two-state model [16], [27], [28] is introduced to describe active and inactive states of a promoter along with transcription and translation processes

$$N_i \underset{k_{off}N_a}{\overset{k_{on}N_i}{\rightleftharpoons}} N_a$$
$$\xrightarrow{\alpha_m N_a} N_{rna} \xrightarrow{\gamma_m N_{rna}}$$
$$\xrightarrow{\alpha_p N_{rna}} N_{pr} \xrightarrow{\gamma_p N_{pr}}$$

where N_i denotes the number of inactive promoters, N_a that of active promoters, N_{rna} the RNA copy number, and N_{pr} the protein copy number. All the above reaction events are generated stochastically. The noise level, n, of N_{pr} can be analytically solved (refer to the Supplementary Notes of [27] for all the detailed derivation)

$$n = \mathrm{CV}^2 = \frac{c}{m}$$

with m denoting the mean value of N_{pr} and the Fano factor c is expressed as

$$c \simeq 1 + \alpha_p \tau_m \left(1 + \alpha_m \frac{k_{off}}{k_{on} + k_{off}} \frac{\tau_p}{1 + (k_{on} + k_{off})\tau_p}\right).$$

Here, we assumed that the protein lifetime τ_p, defined by $1/\gamma_p$, is much larger than the mRNA lifetime τ_m. We note that control coefficients for n can be computed from this equation analytically. We assume that the inactive and active promoter states switch back and forth many times during the protein lifetime ($k_{on} + k_{off} \gg \gamma_p$). We believe this is the case of our experiments and refer to the parameter value estimation described below. In this case, the above equation can be further approximated

$$c \simeq 1 + \alpha_p \tau_m \left(1 + \frac{\alpha_m}{k_{on} + k_{off}} \frac{k_{off}}{k_{on} + k_{off}}\right). \quad (3)$$

Here, $1/(k_{on} + k_{off})$ is the time scale of the promoter state switching and $k_{off}/(k_{on} + k_{off})$ is a suppression weight because the promoter state follows the binomial distribution (due to the fact that the total promoter number is constant) instead of the Poisson distribution. Thus, the second term in the parenthesis can be considered as a transcriptional burst size b_m. In our case, $\alpha_m/(k_{on} + k_{off}) \sim 160/(50 + 51000) \simeq 0.003$. Thus, c can be further simplified

$$c \simeq 1 + \alpha_p \tau_m$$

implying that the change in the IPTG concentration has no effect on the Fano factor, c, thus moving along the line of slope -1 as shown in Fig. 3.

D. Model Parameter Estimation

Transcription rate constant, $\alpha_m = 160\ \text{hr}^{-1}$: The lac-promoter strength, when fully induced with IPTG, was shown $\sim$1.5 time stronger than J23101 by directly measuring the transcript levels with our malachite-green aptamer probes (refer to Fig. 6.3 of [22]). For J23101, α_m was estimated at $0.03\ \text{sec}^{-1} = 110\ \text{hr}^{-1}$ [32]. Thus, α_m for our lac-promoter can be estimated at $160\ \text{hr}^{-1}$.

We used the translation rate constant, $\alpha_p = 1400\ \text{hr}^{-1}$ (Supplementary Notes in [33]), the dilution rate, $\gamma_p = 1\ \text{hr}^{-1}$, and the degradation rate constant of mRNA, $\gamma_m = 30\ \text{hr}^{-1}$.

Gene inactivation, $N_a \xrightarrow{k_{off} N_a} N_i$: The number of the inactive promoters is denoted by N_i and that of the active promoters, N_a. The sum of N_i and N_a is equal to the copy number of the plasmids, N_p (considering that one lac-promoter is included per plasmid). Here, we used $N_p \sim 10$ (http://parts.igem.org/Part:pSB3K3; pGA3K3 is a variant of pSB3K3). k_{off} is related to the search time for LacI to find lac-promoter. When there exist one LacI molecule and one lac-promoter within an *E. coli* cell, the search time is less than $6\ \text{min} = 0.1$ hr [21]. In the case of N_a unoccupied lac-promoters and N_{lacI} copies of $LacI$, the search time becomes $0.1/N_{lacI}N_a$, which is equal to the inverse of the inactivation rate $(1/k_{off}N_a)$. Therefore, k_{off} is estimated to be $10N_{lacI}\ \text{hr}^{-1}$. N_{lacI} can be roughly estimated from the fact that the strength of the lacI^q is similar to the promoter J23101 (α_m of J23101 is $110\ \text{hr}^{-1}$) [32]. Thus, the genomic expression level of LacI, N_{lacI}, becomes $\alpha_p[\text{mRNA}]_{lacI}/\gamma_p = \alpha_p\alpha_m/\gamma_p\gamma_m = 1400 \cdot 110/1 \cdot 30 \simeq 5100$. Thus, k_{off} can be roughly estimated as $5.1 \times 10^4\ \text{hr}^{-1}$.

Gene activation, $N_i \xrightarrow{k_{on} N_i} N_a$: The activation rate constant k_{on} is related to how fast the genomically-expressed LacI detached from its specific promoter *plac* (BBa_R0010). Considering that the dissociation constant is in the range of 0.1–1 pM $= 10^{-4} - 10^{-3}$ (copy number unit; here we used 1 nM corresponds to roughly 1 molecule number in the volume of *E. coli*) [34], [35], k_{on} can be in the range of $k_{off} \times (10^{-4} - 10^{-3}) = 5.1 - 51\ \text{hr}^{-1}$.

E. Noise Level Correction

The mean level was corrected with a simple subtraction. The noise level was corrected by using the property that the observed variance (Variance_o) is the sum of the GFP variance (Variance_g) and the background signal variance (Variance_b) under the assumption that the GFP signals are statistically independent of the background signals. More precisely, the noise level of GFP signals, defined by the square coefficient of variation, can be obtained by

$$\text{CV}^2 = \frac{\text{Variance}_o - \text{Variance}_b}{(\text{Mean}_o - \text{Mean}_b)^2}$$

where the subscripts o and b denote observed and background signals, respectively.

F. Fluorescence Histogram Correction

The effect of autofluorescence was removed from the GFP signal histogram, more precisely probability mass function (pmf), by assuming that the autofluorescence is statistically independent of the true GFP signals [36]. Under this assumption, the pmf of the measured GFP signals, $T(\nu)$, is related to both the autofluorescence pmf $C(\nu)$ and the true GFP signal pmf $S(\nu)$ via convolution

$$T(\nu) = \int_0^\nu C(\nu - \nu')S(\nu')d\nu'.$$

$S(\nu)$ is obtained by minimizing the fitness function

$$M^\alpha(C, T, S) = \int_0^a \left[\int_0^\nu C(\nu - \nu')S(\nu')d\nu' - T(\nu)\right]^2 d\nu + \alpha\|S(\nu)\|.$$

Here, a is the value of ν beyond which $T(\nu)$ is essentially zero, and in our study, we used the entire range of pmf. $\|S(\nu)\|$ is a regularization term, defined as

$$\|S(\nu)\| = \int_0^a \left[g_0 \left(S(\nu)\right)^2 + g_1 \left(dS(\nu)/d\nu\right)^2\right] d\nu$$

where g_0 and g_1 are positive regularization constants. The optimized solution of $S(\nu)$ is obtained by following the procedure described below.

1) Remove background noise that is equipment-specific. Fluorescence signals of strength 0 and 1 (Sony Biotechnology ec800) were considered as background noise and removed.

TABLE I
CONSTANTS USED IN THE AUTOFLUORESCENCE COMPENSATION ALGORITHM

Constants	Values
a	1000
α	0.5
g_0	0.001
g_1	1.0
dS	10^{-6}

Then, $T(\nu)$ (for the induction case of [IPTG] > 0) and $C(\nu)$ (for the case of [IPTG] $= 0$) were computed from the fluorescence signals using 1000 equal-width bins to obtain individual bin-sizes. Here, the bin-size of T is larger than that of C.

2) To compute the convolution, we will set the bin-size of C equal to that of T. Compute C again from the raw data using the bin-size of T, and append an array of zero at the end of C to make the total bin number equal to 1000.
3) Set the initial values of S equal to T.
4) Generate two different random numbers ν_1 and ν_2 in the range of [0, 999]. $S(\nu_1)$ and $S(\nu_2)$ were added and subtracted, respectively, by a constant $dS = 10^{-6}$: $S(\nu_1) \rightarrow S(\nu_1) + dS$ and $S(\nu_2) \rightarrow S(\nu_2) - dS$. When $S(\nu_2) - dS$ is less than zero, set $S(\nu_1)$ equal to $S(\nu_1) + S(\nu_2)$ and then $S(\nu_2)$ equal to 0. In this way, the new S is automatically re-normalized and guaranteed to be non-negative.
5) Compute M^α. If M^α decreases, we accept the change and, otherwise, reject it and revert $S(\nu)$ to the old S values before the update.
6) Repeat the steps 4 and 5 until M^α converges and compare $\int_0^\nu C(\nu - \nu')S_{op}(\nu')d\nu'$ and $T(\nu)$, where S_{op} is the obtained optimized solution of S. If $S_{op}(\nu)$ shows oscillation, reduce the value of α while rebalancing g_0 and g_1 and go back to the step 4. If S_{op} is noisy, increase the value of α while rebalancing g_0 and g_1 and go back to the step 4.

The constants used for the optimization are listed in Table I. The analysis was performed with Python 2.7.9 with Numpy 1.9.2, Scipy 0.15.1, and Spyder 2.3.4. Our python code is provided in the Supplementary Notes.

G. Nonlinear Regression

The gamma distribution function was used to fit our flow cytometry data. Protein copy number N_{pr} can be converted to fluorescence signal intensity x: $N_{pr} = c_s x$ with c_s a scaling constant. The gamma distribution function can be rescaled

$$\begin{aligned} p(N_{pr}; a, b) &= p(c_s x; a, b) = \frac{(c_s x)^{a-1} e^{-c_s x/b}}{\Gamma(a) b^a} \\ &= c_s^{-1} \frac{x^{a-1} e^{-x/\left(\frac{b}{c_s}\right)}}{\Gamma(a) \left(\frac{b}{c_s}\right)^a} \\ &= c_s^{-1} p\left(x; a, \frac{b}{c_s}\right). \end{aligned}$$

Here, Γ is a gamma function, and

$$a \equiv \frac{\alpha_m}{\gamma_p} N_a$$

is the number of mRNA produced per cell doubling time, called burst frequency with N_a the number of active promoters, and

$$b \equiv \frac{\alpha_p}{\gamma_m}$$

is the number of proteins produced during the mRNA lifetime, called burst size. Therefore, the fluorescence intensity should also follow the gamma distribution if its corresponding copy number follows the gamma distribution, with the burst size rescaled with c_s. Nonlinear regression was carried by using the Scipy curve_fit function (http://www.scipy.org/), which employs the Levenberg-Marquardt algorithm for the least squares fitting to estimate a and b.

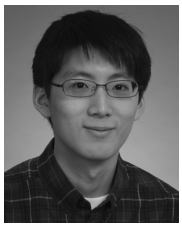
REFERENCES

[1] M. B. Elowitz and S. Leibler, "A synthetic oscillatory network of transcriptional regulators," *Nature*, vol. 403, no. 6767, pp. 335–338, Jan. 2000.
[2] A. Sanchez, S. Choubey, and J. Kondev, "Regulation of noise in gene expression," *Annu. Rev. Biophys.*, vol. 42, pp. 469–491, Jan. 2013.
[3] G. Balázsi, A. van Oudenaarden, and J. J. Collins, "Cellular decision making and biological noise: From microbes to mammals," *Cell*, vol. 144, no. 6, pp. 910–925, Mar. 2011.
[4] S.-L. To and N. Maheshri, "Noise can induce bimodality in positive transcriptional feedback loops without bistability," *Science*, vol. 327, no. Feb., pp. 1142–1146, 2010.
[5] K. H. Kim, H. Qian, and H. M. Sauro, "Nonlinear biochemical signal processing via noise propagation," *J. Chem. Phys.*, vol. 139, p. 144108, 2013.
[6] J. Paulsson, O. G. Berg, and M. Ehrenberg, "Stochastic focusing: Fluctuation-enhanced sensitivity of intracellular regulation," *Proc. Nat. Acad. Sci. USA*, vol. 97, no. 13, pp. 7148–7153, Jun. 2000.
[7] W. J. Blake, G. Balàzsi, M. A. Kohanski, F. J. Isaacs, K. F. Murphy, Y. Kuang, C. R. Cantor, D. R. Walt, and J. J. Collins, "Phenotypic consequences of promoter-mediated transcriptional noise," *Mol. Cell*, vol. 24, pp. 853–865, Dec. 2006.
[8] K. F. Murphy, G. Balázsi, and J. J. Collins, "Combinatorial promoter design for engineering noisy gene expression," *Proc. Nat. Acad. Sci. USA*, vol. 104, pp. 12726–12731, Jul. 2007.
[9] K. F. Murphy, R. M. Adams, X. Wang, G. Balázsi, and J. J. Collins, "Tuning and controlling gene expression noise in synthetic gene networks," *Nucleic Acids Res.*, vol. 38, pp. 2712–2726, May 2010.
[10] H. Maamar, A. Raj, and D. Dubnau, "Noise in gene expression determines cell fate in Bacillus subtilis," *Science*, vol. 317, pp. 526–529, Jul. 2007.
[11] M. R. Birtwistle, A. von Kriegsheim, M. Dobrzyński, B. N. Kholodenko, and W. Kolch, "Mammalian protein expression noise: Scaling principles and the implications for knockdown experiments," *Mol. BioSyst.*, vol. 8, pp. 3068–3076, 2012.
[12] K. H. Kim and H. M. Sauro, "Adjusting phenotypes by noise control," *PLoS Comput. Biol.*, vol. 8, no. 1, p. e1002344, Jan. 2012.
[13] K. H. Kim and H. M. Sauro, "Sensitivity summation theorems for stochastic biochemical reaction systems," *Math. Biosci.*, vol. 226, no. 2, pp. 109–119, Aug. 2010.
[14] D. A. Fell, "Metabolic control analysis: A survey of its theoretical and experimental development," *Biochem. J.*, vol. 286, pp. 313–330, 1992.
[15] H. Kacser and J. A. Burns, "The control of flux," *Biochem. Soc. Trans.*, vol. 23, pp. 341–366, 1995.
[16] Y. Taniguchi, P. J. Choi, G.-W. Li, H. Chen, M. Babu, J. Hearn, A. Emili, and X. S. Xie, "Quantifying E. coli proteome and transcriptome with single-molecule sensitivity in single cells," *Science*, vol. 329, no. 5991, pp. 533–538, Jul. 2010.
[17] E. M. Ozbudak, M. Thattai, I. Kurtser, A. D. Grossman, and A. van Oudenaarden, "Regulation of noise in the expression of a single gene," *Nature Genetics*, vol. 31, pp. 69–73, 2002.
[18] A. Sanchez and I. Golding, "Genetic determinants and cellular constraints in noisy gene expression," *Science*, vol. 342, no. 6163, pp. 1188–1193, Dec. 2013.
[19] J. Elf, G.-W. Li, and X. S. Xie, "Probing transcription factor dynamics at the single-molecule level in a living cell," *Science*, vol. 316, no. 5828, pp. 1191–1194, May 2007.

[20] J. R. Kelly, A. J. Rubin, J. H. Davis, C. M. Ajo-Franklin, J. Cumbers, M. J. Czar, K. de Mora, A. L. Glieberman, D. D. Monie, and D. Endy, "Measuring the activity of BioBrick promoters using an *in vivo* reference standard," *J. Biol. Eng.*, vol. 3, p. 4, 2009.
[21] L.-H. So, A. Ghosh, C. Zong, L. Sepülveda, R. Segev, and I. Golding, "General properties of transcriptional time series in Escherichia coli," *Nature Genetics*, vol. 43, pp. 554–560, 2011.
[22] W. Copeland, "Biological probes to measure transcription dynamics in E. coli," Ph.D. dissertation, Univ. Washington, Seattle, WA, USA, 2015, Print.
[23] H. M. Salis, E. A. Mirsky, and C. A. Voigt, "Automated design of synthetic ribosome binding sites to control protein expression," *Nat. Biotechnol.*, vol. 27, no. 10, pp. 946–950, Oct. 2009.
[24] R. G. Egbert and E. Klavins, "Fine-tuning gene networks using simple sequence repeats," *Proc. Nat. Acad. Sci. USA*, vol. 109, no. 42, pp. 16 817–16 822, Oct. 2012.
[25] H. Salman, N. Brenner, C.-K. Tung, N. Elyahu, E. Stolovicki, L. Moore, A. Libchaber, and E. Braun, "Universal protein fluctuations in populations of microorganisms," *Phys. Rev. Lett.*, vol. 108, no. 23, p. 238105, Jun. 2012.
[26] S. Ghosh, K. Sureka, B. Ghosh, I. Bose, J. Basu, and M. Kundu, "Phenotypic heterogeneity in mycobacterial stringent response," *BMC Syst. Biol.*, vol. 5, no. 1, p. 18, Jan. 2011.
[27] A. Bar-Even, J. Paulsson, N. Maheshri, M. Carmi, E. O'Shea, Y. Pilpel, and N. Barkai, "Noise in protein expression scales with natural protein abundance," *Nat. Genet.*, vol. 38, no. 6, pp. 636–643, Jun. 2006.
[28] V. Shahrezaei and P. S. Swain, "Analytical distributions for stochastic gene expression," *Proc. Nat. Acad. Sci. USA* vol. 105, no. 45, pp. 17 256–17 261, Nov. 2008 [Online]. Available: http://dx.doi.org/10.1073/pnas.0803850105
[29] O. K. Silander, N. Nikolic, A. Zaslaver, A. Bren, I. Kikoin, U. Alon, and M. Ackermann, "A genome-wide analysis of promoter-mediated phenotypic noise in Escherichia coli," *PLoS Genet.*, vol. 8, no. 1, p. e1002443, Jan. 2012.
[30] D. G. Gibson, L. Young, R.-Y. Chuang, J. C. Venter, C. A. Hutchison, and H. O. Smith, "Enzymatic assembly of DNA molecules up to several hundred kilobases," *Nat. Methods*, vol. 6, no. 5, pp. 343–345, May 2009.
[31] F. Hahne, N. LeMeur, R. R. Brinkman, B. Ellis, P. Haaland, D. Sarkar, J. Spidlen, E. Strain, and R. Gentleman, "flowCore: A Bioconductor package for high throughput flow cytometry," *BMC Bioinformat.*, vol. 10, p. 106, Jan. 2009.
[32] S. Zucca, L. Pasotti, G. Mazzini, M. G. C. De Angelis, and P. Magni, "Characterization of an inducible promoter in different DNA copy number conditions," *BMC Bioinformat.*, vol. 13, p. S11, 2012.
[33] B. Canton, A. Labno, and D. Endy, "Refinement and standardization of synthetic biological parts and devices," *Nature Biotechnol.*, vol. 26, no. 7, pp. 787–793, 2008.
[34] P. Wong, S. Gladney, and J. D. Keasling, "Mathematical model of the lac operon: Inducer exclusion, catabolite repression, diauxic growth on glucose and lactose," *Biotechnol. Progr.*, vol. 13, pp. 132–143, 1997.
[35] Y. Setty, A. E. Mayo, M. G. Surette, and U. Alon, "Detailed map of a cis-regulatory input function," *Proc. Nat. Acad. Sci. USA*, vol. 100, pp. 7702–7707, 2003.
[36] J. P. Corsetti, S. V. Sotirchos, C. Cox, J. W. Cowles, J. F. Leary, and N. Blumburg, "Correction of cellular autofluorescence in flow cytometry by mathematical modeling of cellular fluorescence," *Cytometry*, vol. 9, no. 6, pp. 539–547, 1988.

Kyung Hyuk Kim received the Ph.D. degree, studying non-equilibrium stochastic processes, from the Physics Department at the University of Washington (UW), Seattle, WA, USA.

Currently, he is an Acting Assistant Professor in the Department of Bioengineering at UW. His research interests include modularity, stochasticity, and control in cellular systems for applications in synthetic biology. Fascinated by stochasticity in living cells, he enjoys researching noisy biological data and using his advanced mathematical training to uncover patterns in the structural and dynamical properties of cellular systems.



Kiri Choi is working toward the Ph.D. degree in bioengineering, specializing in biological physics, structure and design, at the University of Washington, Seattle, WA, USA.

His research interests include systems and synthetic biology modeling, complex systems, algorithm development, and machine learning.

Bryan Bartley received the B.S. degree in biochemistry from the University of Washington (UW), Seattle, WA, USA.

Currently, he is working toward the Ph.D. degree in bioengineering at UW. He has a broad background in both experimental and computational biology, including experience in biomedical research and industrial biotechnology.

Herbert M. Sauro received the B.Sc. degree in microbiology/biochemistry, the M.S. degree in mathematics and computer science, and the Ph.D. degree in systems biology from Oxford Brookes University, Oxford, U.K.

Currently, he is an Associate Professor in the Department of Bioengineering, University of Washington, Seattle, WA, USA. His main research interest is understanding the operating principles of biological networks. He is the author of several textbooks on linear algebra, enzyme kinetics, and mathematical modeling in biology. He is the cofounder of the Systems Biology Markup Language (SBML).

4.4 Mapping Genetic Design Space with Phylosemantics

Mapping Genetic Design Space with Phylosemantics

Bryan A. Bartley[1], Michal Galdzicki[2], Robert Sidney Cox III[3], Herbert M. Sauro[1]
[1]University of Washington, Department of Bioengineering, Seattle, WA 98195
[2]Arzeda Corp, 2715 W Fort St, Seattle, WA 98199
[3]Prospect Bio, 2627 Hanover St, Palo Alto CA 94304

ABSTRACT

Synthetic biology often employs evolutionary and combinatorial approaches to generate large libraries of genetic variants. In a basic example of a variant library, the biological function of a single genetic component is varied by introducing point mutations. Ultimately, however, synthetic biologists are more concerned with generating combinatorial variants at the system level[1]. Such libraries may include different genetic components arranged in permutative configurations[2], especially order and orientation permutations. In more advanced cases, variants may even encode semantically different networks[3], such as different logic gates.

Here we demonstrate use of a phylosemantic tree to systematically map and explore genetic design space. The tree classifies combinatorial promoters into families which exhibit similar patterns of gene expression, revealing design patterns. Phylosemantics is a combination of phylogenetics and semantic alignments. Semantic alignments are not a new idea[4], [5], but to our knowledge, this is the first application of semantic alignments to engineered genetic systems, or, more specifically, *genetic architectures* in the context of synthetic biology. Applications may include rationally guided DNA assembly or comparative analysis of genetic architectures. We believe phylosemantics could be a useful abstraction technique for the biodesign community and might help synthetic biologists understand how a variety of related designs are related to each other. We are seeking industry or academic collaborators who are interested in applying phylosemantic approaches to a case study and who might be willing to share annotated sequence data.

1. RESULTS & DISCUSSION

The Cox combinatorial promoter library[2] is based on an abstract design composed of three operators arranged sequentially in distal, medial, and proximal positions (Fig. 1). The boundary between positions are defined by the -35 and -10 sigma70 RNA polymerase binding sites. Promoter variants were derived by varying operator types at each position (repressor, neutral, or activator) while also testing different sequence variants for a given type. For example repressor operators include variants of LacI or TetR operators, while activator variants include LuxR or AraC.

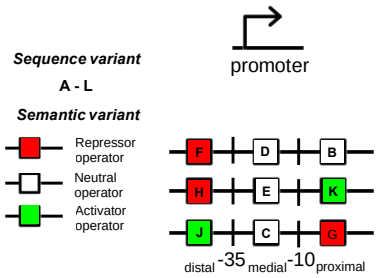


Fig 1. The Cox combinatorial promoter library

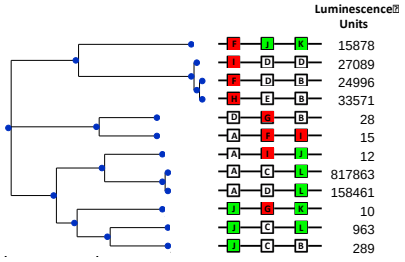


Fig 2. Phylosemantic tree of combinatorial promoters

Of 288 promoters, we selected 12 and mapped them with a phylosemantic tree (Fig 2). The length of branches of the tree correspond to semantic distance between variant designs. Tabulated next to each variant design are the basal levels of gene expression measured for each variant promoter. The advantage of graphing these data with a phylosemantic tree is that some design patterns become more apparent.

The first four variants are all related because they each have a repressor operator distally. Despite the presence of a repressor operator, these promoters exhibit high expression anyway. Repression in this family of promoters appears to fail. Thus, a design rule which may be inferred is that *repressor operators in distal position are ineffective.*

The middle cluster contains similar promoters with a medial repressor operator. Promoters with a medial repressor operator exhibit very low gene expression consistent with repression. This makes sense from a biophysical perspective—a repressor bound in medial position will sterically hinder RNA polymerase binding. A design rule may thus be stated that *repressor operators in medial position exhibit a pronounced "off" effect.*

The lower cluster is largely characterized by activators in the proximal position. This cluster is interesting because it suggests that proximal activator operators exhibit high basal expression when uninduced. The high expression levels coming from

proximal L activation operators suggest leaky expression. Consequently, it is likely that this activator does not have much dynamic range. In order to assess the full range of activation, we would need to graph induced conditions versus the uninduced conditions currently shown. *Activation operators in proximal position may have a poor range of response and be energetically costly due to leaky expression.*

These design rules are not new observations, and were originally described by Cox et al. Thus the design rules we inferred from the phylosemantic tree are aligned with the results of their previous study. The purpose of this exercise was to cross-validate the phylosemantic approach on the Cox library because the general design rules are already known. Some of these design rules can clearly be seen when presented in the context of the phylosemantic tree in Fig. 2. For brevity and clarity, our phylosemantic analysis only includes 12 of 288 promoters. Analysis of the full promoter library might reveal other interesting patterns.

2. METHODS

Phylosemantics, as we define it here, extends conventional phylogenetic approaches with the use of abstract genetic encodings and semantic weights. Phylogenetics is a well-established method for classifying gene variants into evolutionary families based on similarity in primary sequence structure. The typical workflow begins with multiple sequence alignment (MSA) of variant sequences. From the MSA, a pairwise distance matrix is obtained. In phylosemantics, this distance matrix is augmented with semantic weights which may be derived from an ontology, such as the Sequence Ontology[6], or a simple, custom ontology defined by the user. A phylosemantic tree is derived from the semantically-weighted distance matrix.

Phylosemantics requires genetic data that is annotated with terms from which a semantic score may be calculated. For this reason, standardized annotations based on ontologies are especially useful. Annotations may be computationally predicted, or created by biologists using annotation tools like sequence editors. The utility of phylosemantics will always be tied to the richness of annotated sequence data which is available. For this reason, we have leveraged the data standard called Synthetic Biology Open Language[7], [8] (SBOL) which supports rich annotations and easy exchange of genetic data.

For this study we used several open-source software libraries. For semantic representation of combinatorial promoters we used the C++ library LibSBOL 2.1.1[9]. For sequence alignment of abstract sequence encodings we used the C++ library Seqan 2.2.0[10]. For visualization of trees we used the Python phylogenomics module ETE Toolkit[11].

3. FUTURE WORK

So far we have only applied this approach to genetic designs consisting of biological parts arranged in primary sequence. In the future, more sophisticated semantic alignments could be performed, for example on hierarchical genetic structures or the networks they encode. These advanced approaches might become valuable as biological designs become ever more complex and the availability of standardized, well-annotated genetic data improves.

Phylosemantics encompasses a number of related approaches that might apply in different scenarios. For example, different formulae for calculating semantic distance can produce trees that are more useful for one type of analysis versus another. Another choice with interesting implications is whether to construct a rooted versus unrooted tree. Other scenarios in which phylosemantics might be useful include:

- Phylosemantic classification might be used to classify permutations of genes in different orientations.
- Phylosemantics could enable biodesign automation efforts by helping synthetic biologists plan rational assembly strategies given a collection of DNA templates.
- Phylosemantic classification might also be useful for biologists more interested in studying natural systems than building synthetic systems. For example, a comparative study of the architecture of variant proteasome systems might be helped by a more systematic approach[12].

We are seeking industry or academic collaborators who are interested in applying phylosemantic approaches to a case study and who might be willing to share annotated sequence data.

4. ACKNOWLEDGMENTS

This work was supported by NSF#1355909 Synthetic Biology Open Language Resource

5. REFERENCES

[1] C. C. Guet et al., “Combinatorial synthesis of genetic networks,” *Science (80-.).*, vol. 296, no. 5572, pp. 1466–1470, 2002.

[2] R. S. Cox et al., “Programming gene expression with combinatorial promoters,” *Mol. Syst. Biol.*, vol. 3, no. 1, p. 145, 2007.

[3] A. A. K. Nielsen et al., “Genetic circuit design automation,” *Science (80-.).*, vol. 352, no. 6281, p. aac7341, 2016.

[4] M. Schulz et al., “Retrieval, alignment, and clustering of computational models based on semantic annotations,” *Mol. Syst. Biol.*, vol. 7, no. 1, p. 512, 2011.

[5] J. Collado-Vides, “Towards a unified grammatical model of σ70 and σ54 bacterial promoters,” *Biochimie*, vol. 78, no. 5, pp. 351–363, 1996.

[6] K. Eilbeck et al., “The Sequence Ontology: a tool for the unification of genome annotations,” *Genome Biol.*, vol. 6, no. 5, p. R44, 2005.

[7] M. Galdzicki et al., “The Synthetic Biology Open Language (SBOL) provides a community standard for communicating designs in synthetic biology.,” *Nat. Biotechnol.*, vol. 32, no. 6, pp. 545–50, 2014.

[8] N. Roehner et al., “Sharing structure and function in biological design with SBOL 2.0,” *ACS Synth. Biol.*, vol. 5, no. 6, pp. 498–506, 2016.

[9] “LibSBOL.” [Online]. Available: http://sbolstandard.org/software/libraries/.

[10] A. Döring et al., “SeqAn an efficient, generic C++ library for sequence analysis,” *BMC Bioinformatics*, vol. 9, no. 1, p. 11, 2008.

[11] J. Huerta-Cepas et al., “ETE 3: Reconstruction, analysis, and visualization of phylogenomic data,” *Mol. Biol. Evol.*, vol. 33, no. 6, pp. 1635–1638, 2016.

[12] R. De Mot, “Actinomycete-like proteasomes in a Gram-negative bacterium,” *Trends Microbiol.*, vol. 15, no. 8, pp. 335–338, 2007.

Chapter 5

CLOSING THE LOOP IN THE DESIGN-BUILD-TEST CYCLE WITH THE SEMANTIC WEB

5.1 Introduction

Chapter 1 introduced the Synthetic Biology Open Language data exchange standard. When originally released, the SBOL standard supported design tasks for synthetic biology, specifically, designing the composition of hierarchical, synthetic DNA constructs prior to assembly or implementation in the laboratory [11]. Unfortunately, SBOL did not support knowledge-sharing for experimentalists who actually implement these constructs in the lab and perform measurements on them. Much of the practical experience of experimentalists remains buried in hand-written notebooks or scientific publications using natural language descriptions which cannot be easily searched or reproduced by others. Measurement data are often lost on workstations belonging to individual researchers, and these data are often not passed from researcher to researcher when personnel are replaced in the lab. All of these circumstances create a barrier that make it difficult to translate synthetic biology from laboratory-scale, proof-of-principle experiments to industrial-scale applications. The unifying theme of this dissertation is the development of computer-aided technologies and knowledge-sharing applications that support synthetic biologists through the entire engineering problem-solving process, as shown in Figure 5.1. This chapter discusses ongoing work to extend the SBOL standard to realize this vision.

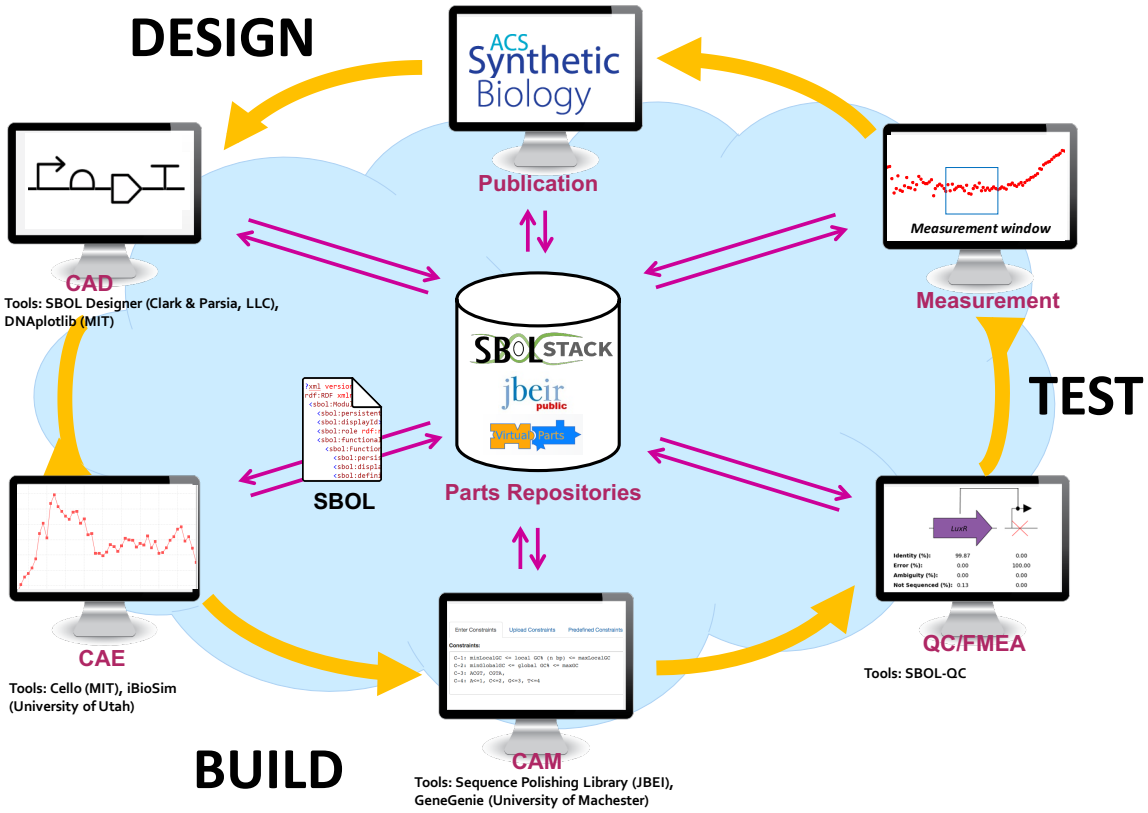


Figure 5.1: The design-build-test cycle in synthetic biology consists of many specialized tasks that may be automated with the help of computer-aided technologies and the SBOL data exchange standard

5.2 The Design-Build-Test Framework for Engineering Problem Solving

Synthetic biologists have adopted from other engineering fields a framework for problem-solving known as the design-build-test cycle. The design-build-test cycle is the scientific method applied in the context of engineering. Stages of the cycle include designing an initial prototype, testing that prototype, analyzing its performance against specific metrics, learning what worked and what did not work, designing a new prototype based on what was learned, and completing the cycle again. Ideally each cycle generates new understanding that feeds

back into new cycles as alternative approaches or reformulated problems. For this reason, it is sometimes called the design-build-test-learn cycle.

The collaborative spirit of synthetic biology closely resembles that of other engineering fields like automotives and aerospace. For example, in the 1980s the American Motor Corporation instituted product life-cycle management (PLM) to help teams of engineers collaborate. Their automated PLM systems supported design-build-test workflows for engineers using computer-aided design tools and networked databases containing designs and documentation. [29, 16] A principle of PLM is that every physical component has a virtual, digital life that tracks data about its actual life. Chrysler, which purchased AMC and incorporated their PLM technologies, credited it with reducing R&D costs by 50% and increasing the timely development of attractive new automotive models. This example illustrates how excellent speed and quality of problem-solving can be achieved with supporting infrastructure and tool chains in place. Synthetic biologists need similar support to achieve visionary goals such as engineering synthetic genomes.

The synthetic biology development cycle includes steps such as using well-characterized DNA parts from online databases, assembling new genetic programs from DNA sequences, synthesizing and assembling DNA constructs in the laboratory, quality control, quantitative characterization of a DNA part's encoded behavior, and submitting characterized parts to inventories so that other engineers may re-use them. Each of these steps may be carried out by different individuals with specialized skills and equipment.An aspirational goal of synthetic biology is to close the design-build-test loop using automated tools.

At a high level, the SBOL standard defines essential types of data that are need to describe and represent a genetic system. This specification is called a data model. A data model can be illustrated using box diagrams that show conceptual building blocks and the relationships that can exist between them. The boxes represent "classes" within the data model. Each

class describes a record of data. An everyday analogy for a "class" is a paper form (e.g., tax document) with fields that one might fill in by hand. A business database may use an internal data model that includes classes for Customer, Address, and OrderHistory. Each record or class may contain additional properties, like name, zip_code, or order_number. In contrast, the SBOL data model includes classes more relevant to synthetic biology, like `Components`, `Modules`, and `Sequences`.

This chapter is based on a formal proposal to the SBOL development community, now being voted upon, to support full design-build-test-learn workflows. This proposal, if ratified, will greatly expand the utility of the SBOL standard for practicing synthetic biologists.

- Decouples the design-build-test-learn process, according to the foundational principles for engineering biology [10].
- Enables domain experts working in different domains of synthetic biology to collaborate in cross-disciplinary and cross-institutional workflows
- Links experimental data about biological samples to their target design
- Captures workflow provenance, a description of the events of a workflow, including which is crucial for scientific reproducibility
- Provides a framework for laboratory automation of design-built-test protocols
- Support model-based design for synthetic biology

In any large-scale engineering effort, the effort may be broken down into smaller tasks carried out by individual specialists. For example, within a single lab, the Densmore lab at Boston University, one person might use a software tool such as Eugene [4] to design a large collection of genetic circuit variants, while another person might use a tool such as Raven [1] to plan the most efficient molecular cloning process for this large library, while

a third person might use a tool like TASBE [3] to perform flow cytometry analysis on the fluorescent reporters expressed by these DNA circuits. This hypothetical workflow is shown in Figure 5.2. This specialization and division-of-labor is what Endy refers to as "decoupling", one of the foundational principles of synthetic biology [10].

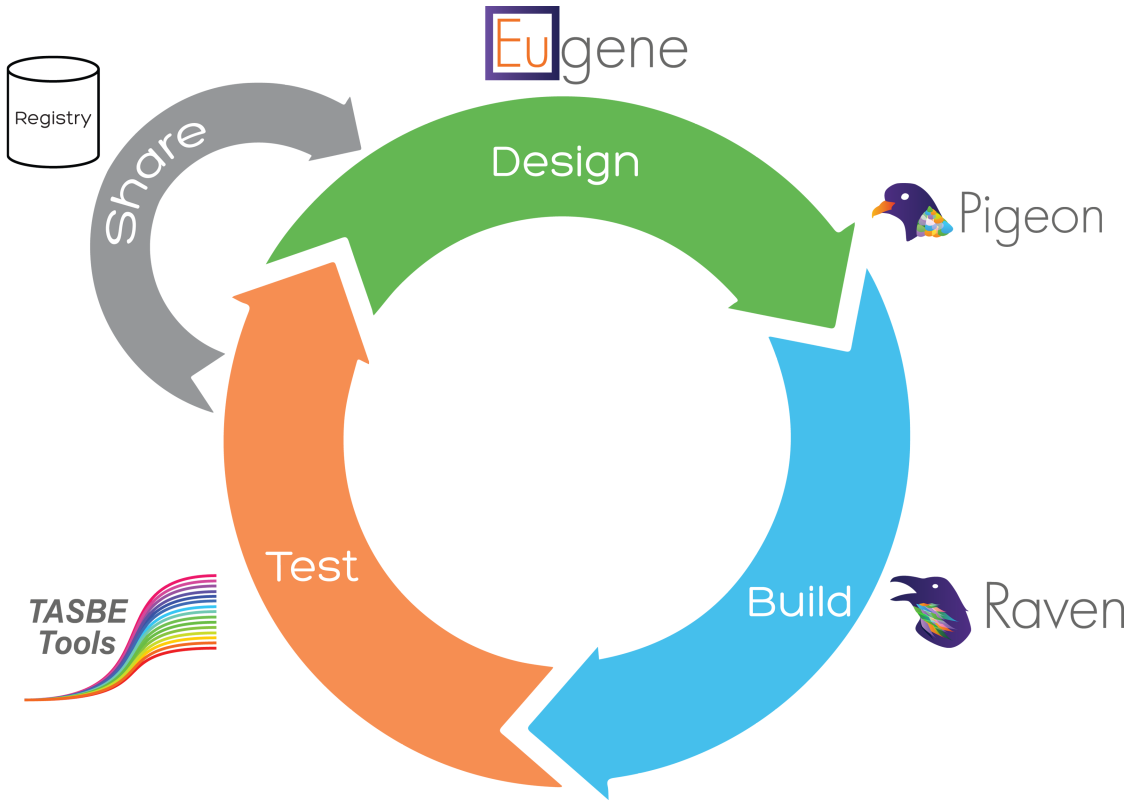


Figure 5.2: Design-build-test cycles for synthetic biology are supported by many software tools. See http://2014.igem.org/Team:BostonU/ChimeraExample

Moreover, different tasks may be performed at different institutions by separate domain experts. Assembly of DNA may be outsourced to a company like Synthetic Genomics (La Jolla, CA), while characterization might be performed by the measurement and analysis pipeline hosted at the Centre for Synthetic Biology and Innovation (CSynBI) at Imperial

College London. Modeling and simulation of biological systems might be carried out by our own Department of Bioengineering here at the University of Washington. Each institution represents a domain of expertise which needs to track its own internal workflows, but also needs to inter-operate with other institutions.

The aim of this specification is to capture workflow provenance, a complete description of evaluation and enactment of computational and laboratory protocols in a workflow. This is crucial to verification, reproducibility, and automation in synthetic biology. Within this model for workflow provenance, other critical use cases can be supported, such as performing model-based design or linking experimental data directly to the biological constructs that the data describes. These workflow histories can then be searched and reproduced with the help of automated tools.

Figure 5.5 illustrates the many use cases that were considered and discussed while developing this proposal. (A use case may be thought of as a particular scenario describing how a user intends to use software to achieve a goal.) Each colored square represents a data object, a record of descriptive data. This proposal is explained in terms of data objects which represent Designs, Builds, Tests, and Models. The Design represents the engineer's conceptual intention, while the Build represents an actual realization of that intention in the laboratory. For example, a Design object contains information about the target sequence for a synthetic construct, while the Build object might contain identifying information for an actual sample in the laboratory. A Test object contains experimental data. A Model is a mathematical idealization of a biological part's behavior derived from experimental observation. The arrows between these data objects represent links which connect data objects with each other into a composite data structure representing a workflow.

This proposal uses the terminology Design, Build, and Test for clarity of discussion because it closely parallels terminology used in synthetic biology literature, but the actual

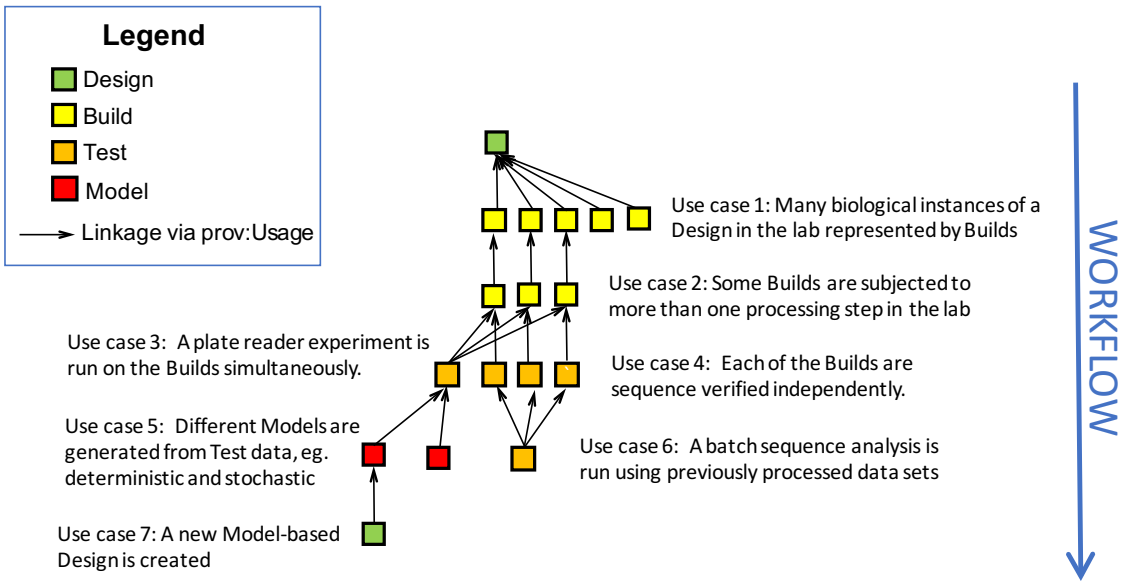


Figure 5.3: Use case 1 describes linking many biological instances (plasmid clones, cellular clones, etc.) to the design which generated them. Use case 2 describes linking sequential stages of a build process which requires more than one processing step in the laboratory. Use case 3 describes a simple case of linking experimental data, such as sequencing data, to a sample. Use case 4 describes performing an experimental test on multiple samples at once in batch, for example in a 96-well plate. Use case 5 describes a scenario in which different models (eg. deterministic vs stochastic) are derived from experimental data. Use case 6 describes a case in which data from multiple experiments are integrated into a single analysis. Use case 7 describes the creation of a model-based design.

implementation of this proposal at the level of computer programming uses a different set of terminology. For example Build objects are actually called `Implementations`, although the semantics or meaning of these terms are similar. This inconsistency in terminology reflects differing and sometimes divisive philosophical perspectives within the SBOL standards development community. While some members of the development community would prefer to use conceptual terminology reflecting the actual terminology used by practitioners in the synthetic biology field, some software developers do not like to introduce new terminology

into the standard and would prefer to re-use ("overload") existing classes to minimize the overhead of implementing changes to software and database infrastructure. As a means to achieve compromise between these different perspectives, Python and C++ software libraries (pySBOL and libSBOL, respectively) were developed to support a feature called extensible data modeling, which will be further discussed in the next section. Throughout this chapter, the terms Design, Build, and Test are preferred for clarity and simplicity of discussion.

The SBOL standard is formally specified in a specification document which, along with written descriptions, uses a diagrammatic language called the Unified Modeling Language (UML) to specify the format of data structures. Using explicit UML diagrams, Figure 5.4 depicts a data structure showing a single step in a workflow. A stage of a workflow is represented as an `Activity` which links together the inputs and outputs of the process. Each `Activity` may be further subclassified as "design", "build", "test", or "learn" activity, and each type of activity has an expected type of input and output. Each `Activity` may be associated with a person, a software tool, and/or a protocol (written in either a human-readable or machine-executable language).

To make this abstraction a little more comprehensible, consider the activity of building a house. The builders require a blueprint (Design), but they also require certain material inputs such as boards and nails (Builds) which are then assembled or composed into the house (Build). Panel A of Figure 5.4 illustrates a "build" activity as applied to a DNA assembly procedure. In this example, the Design is like a blueprint for a genetic construct. The biochemical inputs to the DNA assembly procedure, whether they are synthetic oligonucleotides, DNA restriction fragments, PCR products, or other type of laboratory reagent, are represented by Builds. The output of this activity is a new Build which represents the genetic construct.

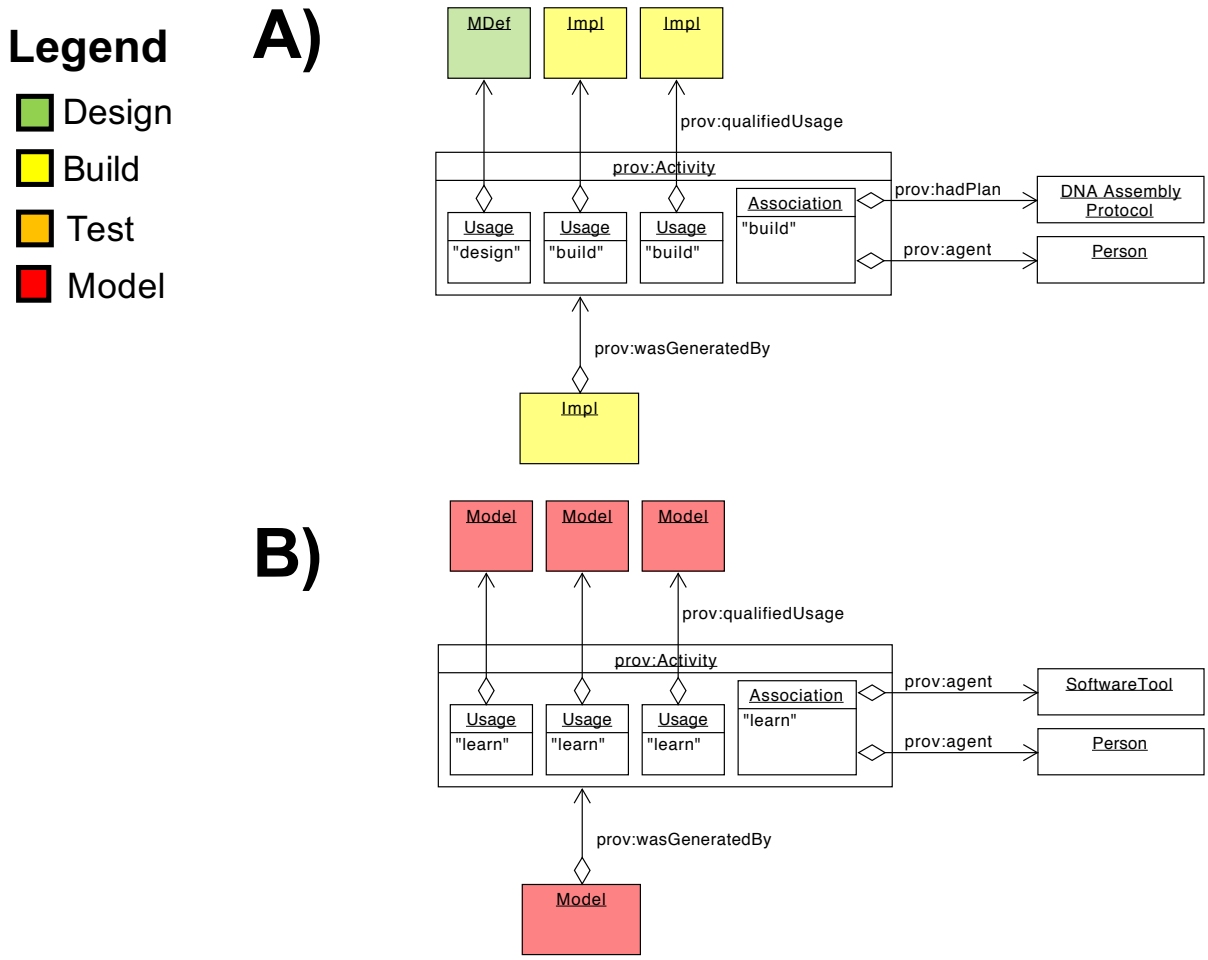


Figure 5.4: (A) Assembly provenance. (B) Modeling provenance.

A goal of synthetic biology is to describe the behavior of synthetic biological systems in mathematically precise terms using dynamic models. Panel B of Figure 5.4 represents a "learn" Activity for model construction. In this example the inputs are models which are merged to compose a more complex system model using a computational tool such as SemGen [13].

While Figure 5.4 illustrates workflows *within* a specialized domain of practice, Figure 5.5 demonstrates how workflows *between* different domains of synthetic biology can be inte-

grated. This figure depicts a data structure representing an idealized design-build-test-learn workflow.

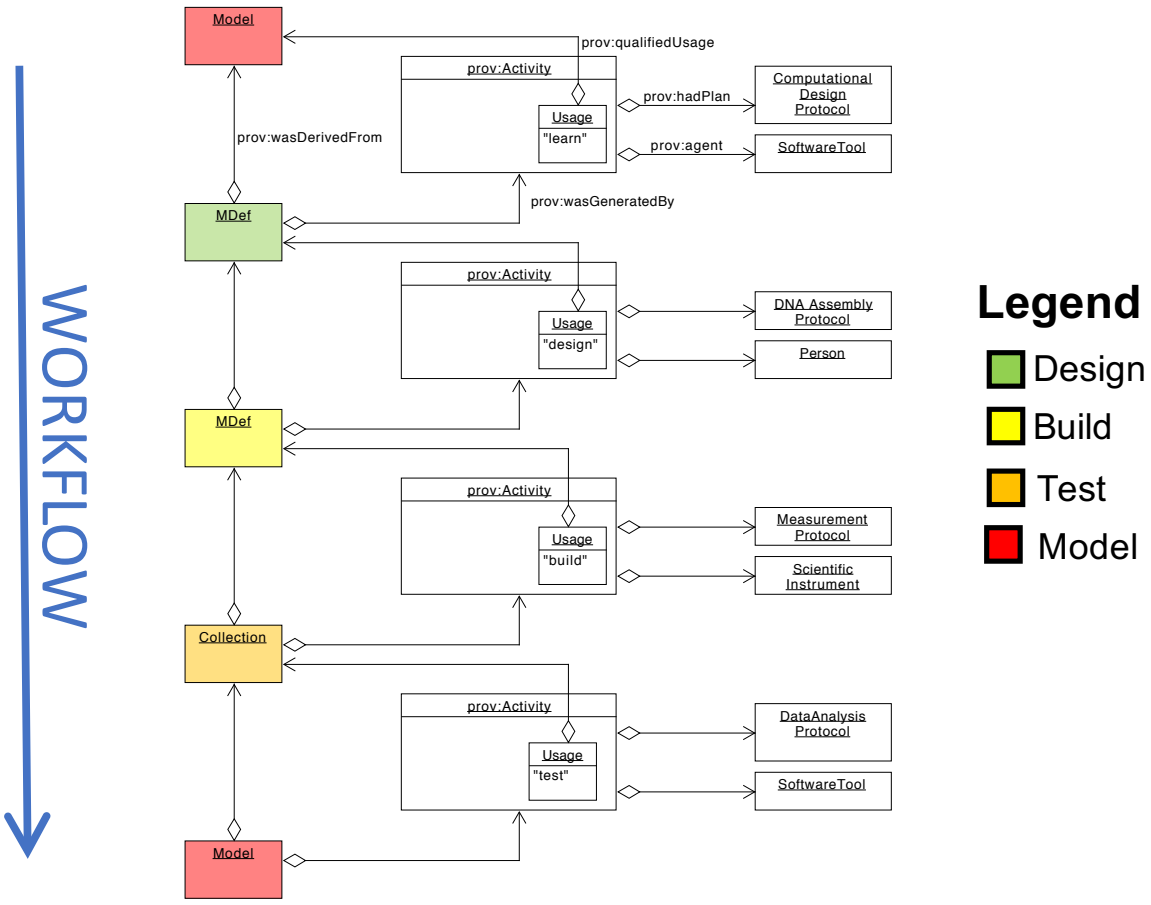


Figure 5.5: An example data structure representing an idealized workflow for model-based design

This particular example represents an idealized workflow for model-based design. The workflow begins with a Model which describes the hypothesized behavior of a system. Using a computational tool, a new Design (`ModuleDefinition`) is composed of biological parts which links back to its Model. A genetic construct is then produced in the laboratory via an

assembly protocol, and this biological sample is represented by a Build (`Implementation`). Now that the Build has been constructed in the laboratory, an automated measurement protocol may be performed, thus generating Test data (represented by a `Collection` of Attachments). Finally, a new `Model` is derived from these data using some kind of fitting algorithm. (Perhaps the most familiar example to biologists would be using Excel to fit a linear curve to a set of standards, but could also include fitting a dynamic model to trajectory data.) The final `Model` may not match the beginning `Model`, as the observed behavior may not match the prediction. This example thus illustrates one complete iteration through a design-build-test-learn cycle.

5.3 Extensible Data Modeling with SBOL

Although SBOL is branded as a standard for synthetic biology, the details of its implementation more properly belong in the realm of bioinformatics or computer science rather than synthetic biology. The low level data format of SBOL is based on resource description framework (RDF), a data standard defined by the World Wide Web Consortium (W3C). It is a standard that allows networked tools, including software applications, web-based tools, databases, and scientific instruments in the "cloud" to communicate, creating a "semantic" web. One advantage of RDF is that it simplifies integration of data derived from different sources across the web. Another advantage of RDF is that it structures natural language into machine-readable grammars so that software tools can automate reasoning tasks. Although most casual users of the web may have no experience with semantic web technology, it is becoming increasingly important to both large enterprises for product life cycle management (PLM) [24], the NIH's Big Data To Knowledge initiative [30] and other "big data" approaches.

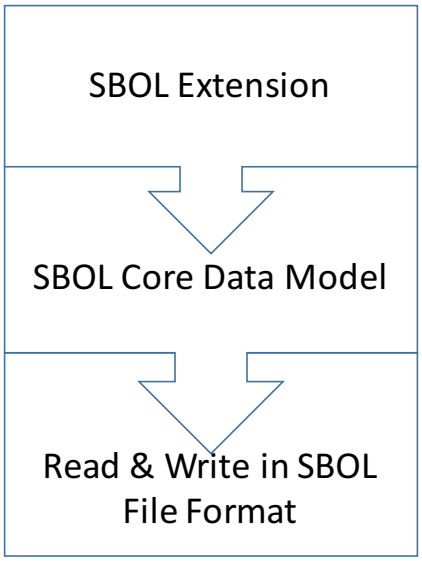


Figure 5.6: The SBOL standard specifies a conceptual data model for describing biological systems on top of a machine-readable file format. Extension data models, and indeed arbitrary data models for knowledge representation in any domain, can be developed which use the SBOL file format.

The purpose of the SBOL data model, which was introduced in the previous section, is to add a layer of human-readable knowledge representation on top of SBOL's machine-readable RDF data format. This makes it possible to design software based on how humans solve problems and represent knowledge, rather than how computers process data. These distinctly different aspects of the SBOL standard as a human-readable data model versus SBOL as a machine-readable file format based will become important later.

The significance of SBOL as a semantic web standard can be highlighted by considering it in the historical context of data exchange standards. Writing in the journal Computer Aided Design, authors from the Manufacturing Engineering Laboratory at the National Institute of Standards and Technology, describe the historical evolution of data exchange standards as follows [24]:

> The current set of the underlying linguistic structures populating PLM support is fragmented and incomplete in coverage. These incompatible linguistic structures represented by the current standards have evolved in a bottom-up fashion – based on localized needs and technology provider-centric definitions – for supporting particular aspects of the product life cycle

In other words, standards developers boxed themselves into a corner by taking a narrow view of the applications of standards within their own domain of knowledge, and as a consequence they now have difficulties integrating data across different fields of practice. This is shown in Figure 5.7. It is precisely this outcome that can be avoided by leveraging semantic web technologies such as SBOL. As a semantic web standard, SBOL supports a concept called

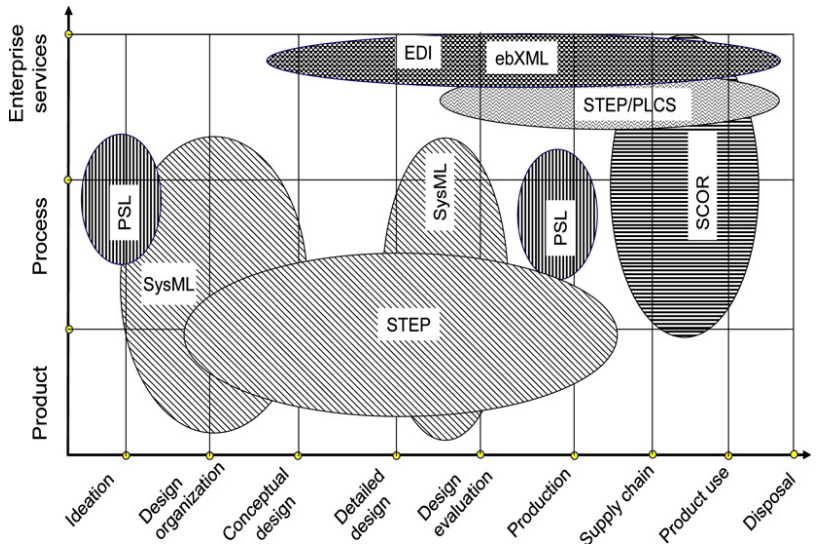


Figure 5.7: Data exchange standards for produce lifecycle management are not interoperable. [24]

extensibility that allows users to integrate SBOL data with other types of data. For example, an engineer may wish to combine descriptive data about a microfluidic device, not currently within the scope of the SBOL data model, with descriptive data about a synthetic biological

system under study in the device. An advantage of the SBOL file format is that it allows developers to embed custom data within SBOL documents, such that these data can be exchanged without being damaged or lost. However, this mechanism for including so-called extension data results in a data exchange scenario described here as "weak" data exchange. In weak data exchange, extension data are preserved, but they are preserved in the form of generic data structures which do not present a meaningful interface to a human user (see 5.8. In contrast, "strong" data exchange can be achieved through extensible data modeling, described below.

As described in Chapter 2 I have developed open source Python (pySBOL) and C++ (libSBOL) libraries which read, write, and manipulate SBOL. These libraries support a unique feature called extensible data modeling which distinguishes it from other libraries that have been developed within the SBOL development community and which makes a a novel contribution to the fields of bioinformatics and standards development. Extensible data modeling makes it easy for users to extend and customize the SBOL data model to their needs, as well as lowering the maintenance cost of software libraries for future open source development. An "extension" to the data model adds new classes or new properties to an existing class. Such extensions are easy to develop and distribute so that other users can read and write extension data in SBOL files. SBOL extensions require no additional library installation or modification of existing libraries. They are a simple C++ header file or Python file which is included with a client project. Most importantly, these extensions enable users to share data using explicit class and property names rather than generic data structures, achieving human-readable knowledge representation and strong data exchange.

Figure 5.9 illustrates why extensible data modeling is useful to a standards development effort. The left panel diagrams the SBOL 2.0 data model in its most recent publication [25]. The right panel diagrams the Manufacturing Core Ontology (MCO) [5], a data model for

product lifecycle management. There is conceptual similarity between some of the classes in both data models. For example, SBOL Components and MCO Parts roughly map to each other. The point of this comparison is to illustrate that the SBOL data model can reasonably be expected to expand as the standard grows to support more use cases for synthetic biology. An advantage of extensible data modeling is how easy it is to expand the data model without major software development effort. Since SBOL is intended to be an open-source development effort, extensible data modeling lowers the barrier to entry and maintenance cost for new contributions from the open-source community.

The example extensions in Figure 5.10c illustrate how extensions are currently being used in practice by the community. The first example demonstrates an extension model for describing how an experiment is organized, experimental groups, and samples. This extension is being used at BBN Technologies (Boston, MA) as part of the DARPA Synergistic Discovery & Design project.

SBOL extension classes might be used to represent organisms in cell banks. This is the purpose of the Host extension, which describes a host organism, its genome, and its mutations. One reason this might be useful is for automating the design of biological systems composed of a genetic circuit coupled to a host organism. Some genetic systems may be compatible only with certain host organisms, and this decision-making process can be automated. Currently I have implemented an SBOL knowledgebase which contains data about *E. coli* strains culled from multiple sources.

A final example illustrates how SBOL might be extended to represent 3-dimensional biological structures, for 3-D CAD of synthetic biomolecules. This can be achieved simply by adding a new kind of Location class that contains `x, y, z` coordinates. Modular protein complexes like polyketide synthases might be synthetically designed using this feature. The

3D structure extension is being developed in collaboration with researchers at Newcastle University in the UK.

This discussion about extensibility and interoperability also has an important context in the field of computational biology. Many computational standards relating to different domains of biology have emerged in the last decade, prompting the many communities behind these efforts to combine efforts through the Computational Modeling in Biology Network (COMBINE) [18, 20, 7, 11, 23]. A goal of the COMBINE is to develop interoperable standards that allow different sources of data to be integrated. The best approach for this goal is still an ongoing discussion topic at the workshops involving all the standards communities. The extensible data modeling approach employed by the pySBOL and libSBOL libraries demonstrates a proof-of-principle for future development of interoperable standards.

Extensibility is a feature that invites open and easy participation, experimentation, and innovation in the standards process. Rather than a centralized approach to standardization, extensibility allows a more open-source, grassroots approach. Coders can easily modify existing extensions, changing some properties of a class while keeping others, for instance. Such modifications could result in a diverse ecosystem of overlapping, interoperable extensions, rather than a single centralized, monolithic SBOL standard. It is expected that more useful extensions will be formally adopted into the SBOL core specification as the community converges towards a consensus. This is a distinctly semantic web approach to standards development, and in some ways a radical social experiment in collaborative science.

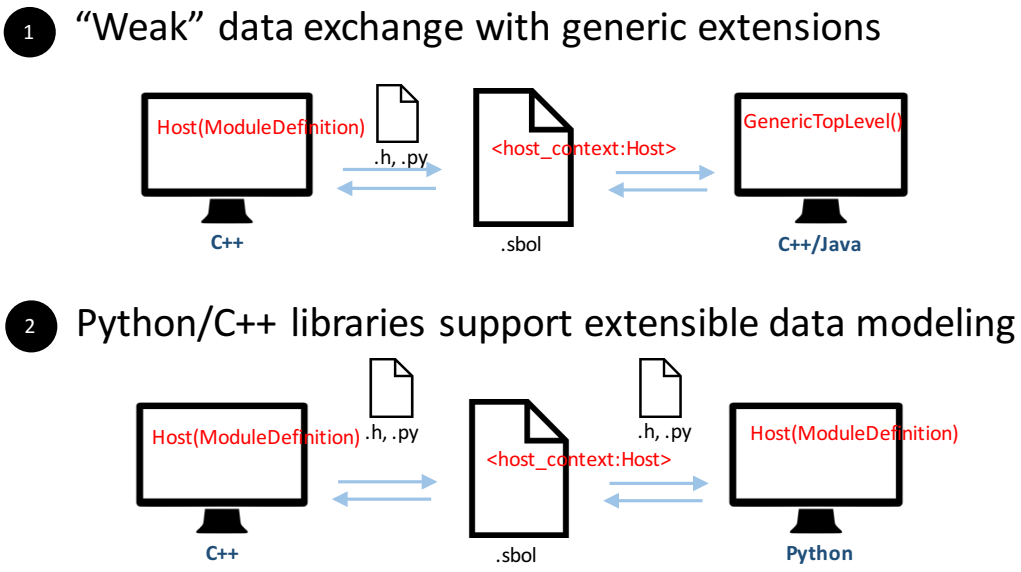
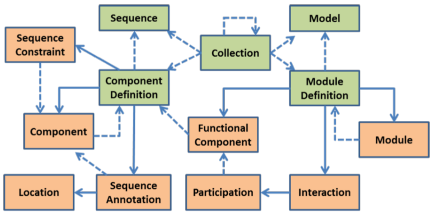
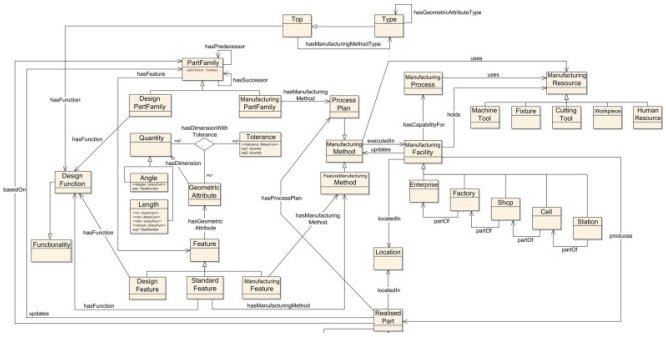


Figure 5.8: "Weak" data exchange versus "strong" data exchange. An SBOL extension allows users to define new classes in the data model, but software libraries which support only weak data exchange lose this human readability. In the top panel, one user defines a new class called `Host` (to represent a host cell). To a user of the Java library, this information is imported as a generic object. In contrast, Python and C++ libraries support strong data exchange which preserves the `Host` class during data exchange. The extension preserves and presents meaningful information to human users.

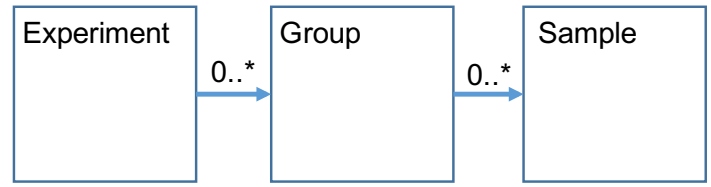


SBOL in 2016

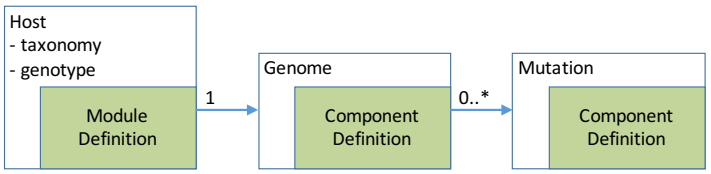


SBOL in 2020

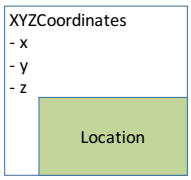
Figure 5.9: A comparison of data models in the Synthetic Biology Open Language versus the Manufacturing Core Ontology



(a)



(b)



(c)

Figure 5.10: (A) Extension classes for representing experimental design (B) A Host extension for strain repositories and cell banks (C) Representing a 3D structure using SBOL

BIBLIOGRAPHY

[1] Evan Appleton, Jenhan Tao, Traci Haddock, and Douglas Densmore. Interactive assembly algorithms for molecular cloning. *Nature methods*, 11(6):657–662, 2014.

[2] Bryan Bartley, Jacob Beal, Kevin Clancy, Goksel Misirli, Nicholas Roehner, Ernst Oberortner, Matthew Pocock, Michael Bissell, Curtis Madsen, Tramy Nguyen, et al. Synthetic biology open language (sbol) version 2.0. 0. *Journal of Integrative Bioinformatics (JIB)*, 12(2):902–991, 2015.

[3] Jacob Beal, Ron Weiss, Fusun Yaman, Noah Davidsohn, and Aaron Adler. A method for fast, high-precision characterization of synthetic biology devices. 2012.

[4] Lesia Bilitchenko, Adam Liu, Sherine Cheung, Emma Weeding, Bing Xia, Mariana Leguia, J Christopher Anderson, and Douglas Densmore. Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems. *PloS one*, 6(4):e18882, 2011.

[5] Nitishal Chungoora, Robert I Young, George Gunendran, Claire Palmer, Zahid Usman, Najam A Anjum, Anne-FrançOise Cutting-Decelle, Jennifer A Harding, and Keith Case. A model-driven ontology approach for manufacturing system interoperability and knowledge sharing. *Computers in Industry*, 64(4):392–401, 2013.

[6] Mike T Cooling, V Rouilly, Goksel Misirli, J Lawson, Tommy Yu, Jennifer Hallinan, and Anil Wipat. Standard virtual biological parts: a repository of modular modeling components for synthetic biology. *Bioinformatics*, 26(7):925–931, 2010.

[7] Autumn A Cuellar, Catherine M Lloyd, Poul F Nielsen, David P Bullivant, David P Nickerson, and Peter J Hunter. An overview of cellml 1.1, a biological model description language. *Simulation*, 79(12):740–747, 2003.

[8] Karen Eilbeck, Suzanna E Lewis, Christopher J Mungall, Mark Yandell, Lincoln Stein, Richard Durbin, and Michael Ashburner. The sequence ontology: a tool for the unification of genome annotations. *Genome biology*, 6(5):R44, 2005.

[9] Michael B Elowitz and Stanislas Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403(6767):335–338, 2000.

[10] Drew Endy. Foundations for engineering biology. *Nature*, 438(7067):449–453, 2005.

[11] Michal Galdzicki, Kevin P Clancy, Ernst Oberortner, Matthew Pocock, Jacqueline Y Quinn, Cesar A Rodriguez, Nicholas Roehner, Mandy L Wilson, Laura Adam, J Christopher Anderson, et al. The synthetic biology open language (sbol) provides a community standard for communicating designs in synthetic biology. *Nature biotechnology*, 32(6):545–550, 2014.

[12] Timothy S Gardner, Charles R Cantor, and James J Collins. Construction of a genetic toggle switch in escherichia coli. *Nature*, 403(6767):339–342, 2000.

[13] John H Gennari, Maxwell L Neal, Michal Galdzicki, and Daniel L Cook. Multiple ontologies in action: Composite annotations for biosimulation models. *Journal of biomedical informatics*, 44(1):146–154, 2011.

[14] Daniel G Gibson, Lei Young, Ray-Yuan Chuang, J Craig Venter, Clyde A Hutchison, and Hamilton O Smith. Enzymatic assembly of dna molecules up to several hundred kilobases. *Nature methods*, 6(5):343–345, 2009.

[15] Timothy S Ham, Zinovii Dmytriv, Hector Plahar, Joanna Chen, Nathan J Hillson, and Jay D Keasling. Design, implementation and practice of jbei-ice: an open source biological part registry platform and tools. *Nucleic acids research*, page gks531, 2012.

[16] S Hill. How to be a trendsetter: Dassault and ibm plm customers swap tales from the plm front. *COE Newsnet*, 2003.

[17] Nathan J Hillson, Hector A Plahar, Jacob Beal, and Ranjini Prithviraj. Improving synthetic biology communication: Recommended practices for visual depiction and digital submission of genetic designs, 2016.

[18] Michael Hucka, Andrew Finney, Herbert M Sauro, Hamid Bolouri, John C Doyle, Hiroaki Kitano, Adam P Arkin, Benjamin J Bornstein, Dennis Bray, Athel Cornish-Bowden, et al. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[19] Clyde A Hutchison, Ray-Yuan Chuang, Vladimir N Noskov, Nacyra Assad-Garcia, Thomas J Deerinck, Mark H Ellisman, John Gill, Krishna Kannan, Bogumil J Karas, Li Ma, et al. Design and synthesis of a minimal bacterial genome. *Science*, 351(6280):aad6253, 2016.

[20] Nicolas Le Novere, Michael Hucka, Huaiyu Mi, Stuart Moodie, Falk Schreiber, Anatoly Sorokin, Emek Demir, Katja Wegner, Mirit I Aladjem, Sarala M Wimalaratne, et al. The systems biology graphical notation. *Nature biotechnology*, 27(8):735–741, 2009.

[21] Curtis Madsen, James Alastair McLaughlin, Goksel Mısırlı, Matthew Pocock, Keith Flanagan, Jennifer Hallinan, and Anil Wipat. The sbol stack: a platform for storing, publishing, and sharing synthetic biology designs. *ACS synthetic biology*, 5(6):487–497, 2016.

[22] Priscilla EM Purnick and Ron Weiss. The second wave of synthetic biology: from modules to systems. *Nature reviews Molecular cell biology*, 10(6):410–422, 2009.

[23] Jacqueline Y Quinn, Robert Sidney Cox III, Aaron Adler, Jacob Beal, Swapnil Bhatia, Yizhi Cai, Joanna Chen, Kevin Clancy, Michal Galdzicki, Nathan J Hillson, et al. Sbol visual: A graphical language for genetic designs. *PLoS Biol*, 13(12):e1002310, 2015.

[24] Sudarsan Rachuri, Eswaran Subrahmanian, Abdelaziz Bouras, Steven J Fenves, Sebti Foufou, and Ram D Sriram. Information sharing and exchange in the context of product lifecycle management: Role of standards. *Computer-Aided Design*, 40(7):789–800, 2008.

[25] Nicholas Roehner, Jacob Beal, Kevin Clancy, Bryan Bartley, Goksel Misirli, Raik Grunberg, Ernst Oberortner, Matthew Pocock, Michael Bissell, Curtis Madsen, et al. Sharing structure and function in biological design with sbol 2.0. *ACS synthetic biology*, 5(6):498–506, 2016.

[26] Herbert M Sauro. Modularity defined. *Molecular systems biology*, 4(1):166, 2008.

[27] Reshma P Shetty, Drew Endy, and Thomas F Knight. Engineering biobrick vectors from biobrick parts. *Journal of biological engineering*, 2(1):5, 2008.

[28] Sean C Sleight, Bryan A Bartley, Jane A Lieviant, and Herbert M Sauro. In-fusion biobrick assembly and re-engineering. *Nucleic acids research*, page gkq179, 2010.

[29] Gerald I Susman. *Integrating design and manufacturing for competitive advantage*. Oxford University Press, 1992.

[30] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3, 2016.

Appendix A: Synthetic Biology Open Language Specification Document

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de/

Synthetic Biology Open Language (SBOL) Version 2.0.0

Bryan Bartley[1], Jacob Beal[2], Kevin Clancy[3], Goksel Misirli[4], Nicholas Roehner[5*], Ernst Oberortner[6], Matthew Pocock[4], Michael Bissell[8], Curtis Madsen[4], Tramy Nguyen[7], Zhen Zhang[7], John H. Gennari[1], Chris Myers[7], Anil Wipat[4] and Herbert Sauro[1]

[1]University of Washington, USA

[2]Raytheon BBN Technologies, USA

[3]ThermoFisher Scientific, USA

[4]Newcastle University, UK

[5]Boston University, USA

[6]DOE Joint Genome Institute, USA

[7]University of Utah, USA

[8]Amyris, Inc., USA

Summary

Synthetic biology builds upon the techniques and successes of genetics, molecular biology, and metabolic engineering by applying engineering principles to the design of biological systems. The field still faces substantial challenges, including long development times, high rates of failure, and poor reproducibility. One method to ameliorate these problems would be to improve the exchange of information about designed systems between laboratories. The *Synthetic Biology Open Language* (SBOL) has been developed as a standard to support the specification and exchange of biological design information in synthetic biology, filling a need not satisfied by other pre-existing standards. This document details version 2.0 of SBOL, introducing a standardized format for the electronic exchange of information on the structural and functional aspects of biological designs. The standard has been designed to support the explicit and unambiguous description of biological designs by means of a well defined data model. The standard also includes rules and best practices on how to use this data model and populate it with relevant design details. The publication of this specification is intended to make these capabilities more widely accessible to potential developers and users in the synthetic biology community and beyond.

*To whom correspondence should be addressed. Email: nicholasroehner@gmail.com

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

Synthetic Biology Open Language (SBOL) Version 2.0.0

Editors:

Bryan Bartley	*University of Washington, USA*
Jacob Beal	*Raytheon BBN Technologies, USA*
Kevin Clancy	*ThermoFisher Scientific, USA*
Goksel Misirli	*Newcastle University, UK*
Nicholas Roehner	*Boston University, USA*

Chair:

Herbert Sauro *University of Washington, USA*

editors@sbolstandard.org

Additional authors, by institution:

Ernst Oberortner	*DOE Joint Genome Institute, USA*
Curtis Madsen, Matthew Pocock, Anil Wipat	*Newcastle University, UK*
Tramy Nguyen, Zhen Zhang, Chris Myers	*University of Utah, USA*
John H. Gennari	*University of Washington, USA*
Michael Bissell	*Amyris, Inc., USA*

Version 2.0.0

July 31, 2015

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

Contents

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

1 Purpose

Synthetic biology builds upon the techniques and successes of genetics, molecular biology, and metabolic engineering by applying engineering principles to the design of biological systems. These principles include standardization, modularity, and design abstraction. The field still faces substantial challenges, including long development times, high rates of failure, and poor reproducibility. A common factor of these challenges is the exchange of information about designed systems between laboratories. When designing a synthetic system, synthetic biologists need to exchange information about multiple types of molecules and their expected behavior in the design. Furthermore, there are often multiple degrees of separation between a specified nucleic acid sequence (e.g., a sequence that encodes an enzyme or transcription factor) and the molecular interactions that a designer intends to result from said sequence (e.g., chemical modification of metabolites or regulation of gene expression), yet these different perspectives need to be connected together in the engineering of biological systems.

The *Synthetic Biology Open Language* (SBOL) has been developed as a standard to support the specification and exchange of biological design information in synthetic biology, filling a need not satisfied by other pre-existing standards. Previous nucleic acid sequence description formats lack key capabilities. For example, simple sequence encoding formats such as FASTA encode almost nothing about design rationale. More sophisticated formats such as GenBank and Swiss-Prot support a flat annotation of sequence features that is well suited to the description of natural systems, but is unable to represent the multi-layered design structure common to engineered systems. Figure 1 shows the relationship of selected prior sequence description formats to SBOL 1.1 and SBOL 2.0. Modelling languages, such as the Systems Biology Markup Language (SBML) Finney et al. (2006) can be used represent biological processes, but are not sufficient to represent the associated nucleotide or amino acid sequences. Synthetic biology needs a structured standard that defines how to represent relevant molecules and their functional roles within a designed system, standardized rules on how such information is encoded in a file format, and software libraries to enable the exchange of such data between participating laboratories and as part of the publication process.

To help address these challenges, SBOL introduces a standardized format for the electronic exchange of information on the structural and functional aspects of biological designs. The standard has been designed to support the explicit and unambiguous description of biological designs by means of a well defined data model. The standard further describes the rules and best practices on how to use this data model and populate it with relevant design details. SBOL uses existing Semantic Web practices and resources, such as *Uniform Resource Identifiers* (URIs) and ontologies, to unambiguously identify and define genetic design elements. The definition of the data model and associated format, the rules on the addition of data within the format and the representation of this in electronic data files are intended to make the SBOL standard a useful means of promoting global data exchange between laboratories and between software programs.

This document details version 2.0 of SBOL. The previous version 1.1 of the SBOL standard focused on representing the structural aspects of genetic designs. Users of the standard were able to exchange information on DNA designs, but they could not represent molecules other than DNA or the functional aspects of designs beyond DNA sequence features. The SBOL 2.0 data model defined in this specification extends the version 1.1 data model to provide for the most pressing data exchange needs identified by the SBOL community. In particular, the extended data model can:

- Represent non-DNA structural components of a biological design, including RNA, proteins, small molecules and other physical components.
- Describe the behavioral aspects of a biological design, such as the intended or expected molecular interactions, and link to mathematical models written in other standards
- Associate structure and function so that a single design can be understood in terms of its structure, behavior, or both.
- Support rich annotation of biological designs, so that classes of design data that are not explicitly covered by this specification can be safely exchanged

doi:10.2390/biecoll-jib-2015-272

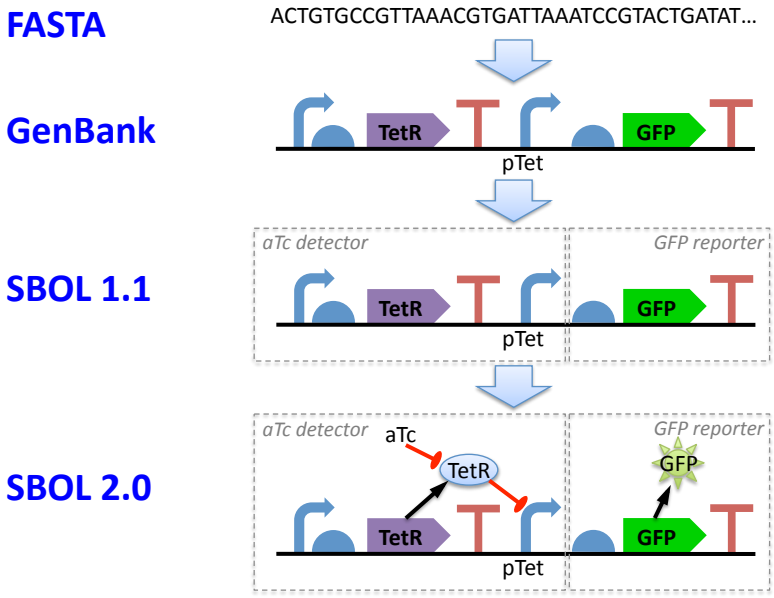


Figure 1: *SBOL 2.0 extends prior sequence description formats to represent both the structure and function of a genetic design in a modular, hierarchical manner.*

Taken together, these extensions enable SBOL to support the description and exchange of hierarchical, modular representations of both the intended structure and function of designed biological systems.

While the ultimate goal of SBOL is to describe synthetic biological designs such that they can be reproduced in the lab with a high degree of fidelity, SBOL 2.0 does not yet provide a complete catalog of the different classes of data that are necessary to achieve this goal. For example, SBOL 2.0 does not yet include data on environmental and host context, or details on how the performance of a design is measured. To enable progress towards capturing these types of data, SBOL 2.0 provides an annotation mechanism that allows SBOL to be easily extended (see Section 7.11). Three scenarios are envisaged for extending SBOL:

- Critical data related to the reproducibility of designs. These include what growth media was used, what temperature the organisms were grown at, or where the recombinant DNA was integrated into the host genome or a plasmid.
- Tool specific data. These could include tool settings specific to the design that is being loaded, such as which windows are to be opened or which settings are to be initialized. Tool makers could also include encrypted proprietary information related to a company or client in an extension.
- Data that are non-essential for reproducibility but are nevertheless useful to many users. There are many cases where specific communities of users require data that cannot be explicitly represented using the SBOL data model. These include data on visualization, evolutionary stability, or other .

The extension mechanism is therefore a critical part of SBOL 2.0 and will allow others in the community to incorporate their own custom data into SBOL files and contribute to community efforts to expand the scope of SBOL.

The SBOL 2.0 specification also adds a number of measures to simplify adoption and validation of compatibility with the standard. First, unlike the SBOL 1.1 specification, the SBOL 2.0 specification explicitly incorporates the primary

serialization format for its data model to better show how the standard can be used. Second, the specification includes a set of validation rules for determining the compatibility of a document with SBOL 2.0, most of which are machine-verifiable. Finally, the specification includes a set of recommended best practices that can allow software tools to take best advantage of the standard and effectively exchange data.

In addition, care has been taken to ensure that SBOL 2.0 is backwards-compatible with previous versions. While the changes made in SBOL 2.0 do mean that a SBOL 1.x file is not a valid SBOL 2.0 file, there does exist a direct mapping from the SBOL 1.x data model to the SBOL 2.0 data model, making it possible to automatically convert any SBOL 1.x file to an SBOL 2.0 file. Since SBOL 2.0 can encode all data previously encoded in SBOL 1.1, developers are encouraged to upgrade their SBOL 1.1 compliant software tools to use SBOL 2.0 software libraries.

Lastly, the SBOL standard has been developed in collaboration between both "wet" bench scientists and "dry" scientific modelers and software developers that are active within the synthetic biology community. As before with SBOL version 1.1, this open community has met to discuss and agree upon the data exchange needs that version 2.0 of the SBOL standard is intended to address. These discussions have informed the efforts of developers within the community to produce a SBOL 2.0 specification after several rounds of proposal and revision. This specification has been evaluated by the community for its ability to represent a wide range of synthetic biology designs and share these designs between different laboratories. This specification has also informed the development of software libraries that implement the standard, and software tools that employ the standard by means of these libraries, thereby providing further testing of SBOL 2.0. The publication of this specification is intended to make these capabilities more widely accessible to potential developers and users in the synthetic biology community and beyond.

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

2 Relation to other BBF RFCs

BBF RFC 107 replaces BBF RFC 87 (the SBOL 1.1 standard).

BBF RFC 107 updates BBF RFC 30 (RDF-based framework for synthetic biology data), as it proposes a standard conforming to BBF RFC 30.

BBF RFC 107 also implicitly supersedes the previously replaced BBF RFC 84 (SBOL 1.0, replaced by BBF RFC 87) and BBF RFC 31 (PoBoL, replaced by BBF RFC 84).

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

3 Copyright and License Statement

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

4 A Brief History of SBOL

The SBOL effort was kickstarted in 2006 with the goal of developing a data exchange standard for genetic designs. Herbert Sauro (University of Washington) secured a modest grant from Microsoft in the field of computational synthetic biology, which was used to fund the initial meeting in Seattle on April 26-27, 2008. This workshop was organized by Herbert Sauro, Sean Sleight, and Deepak Chandran, and included talks by Raik Gruenberg, Kim de Mora, John Cumbers, Christopher Anderson, Mac Cowell, Jason Morrison, Jean Peccoud, Ralph Santos, Andrew Milar, Vincent Rouilly, Mike Hucka, Michael Blinov, Lucian Smith, Sarah Richardson, Guillermo Rodrigo, Jonathan Goler, and Michal Galdzicki.

Michal's early efforts were instrumental in making SBOL successful. As part of his doctoral work, Mike led the development of PoBol, as SBOL was originally known. He organized annual workshops from 2008 to 2011 and kept the idea of developing a genetic design standard alive. The original SBOL 1.0 was developed by a small group of dedicated researchers calling themselves the Synthetic Biology Data Exchange Working Group, meeting at Stanford in 2009 and Anaheim, CA in 2010. During the Anaheim meeting, the community decided to write a letter to Nature Biotechnology highlighting the issue of reproducibility in synthetic biology. This letter was initiated by Jean Peccoud and submitted by participants of the Anaheim meeting, including Deepak Chandran, Douglas Densmore, Dmytriv, Michal Galdzicki, Timothy Ham, Cesar Rodriquez, Jean Peccoud, Herbert Sauro, and Guy-Bart Stan. The overall pace of development quickened when several new members joined at the next workshop in Blacksburg, Virginia on January 7-10, 2011. This early work was also supported by an STTR grant from the National Institute of Health (NIH #1R41LM010745 and #9R42HG006737, from 2010-13) in collaboration with Clark & Parsia, LLC (Co-PIs: John Gennari and Evren Sirin). New members included Cesar Rodriguez, Mandy Wilson, Guy-Bart Stan, Chris Myers, and Nicholas Roehner.

The SBOL Developers Group was officially established at a meeting in San Diego in June 2011. Rules of governance were established, and the first SBOL editors were elected: Mike Galdzicki, Cesar Rodriguez, and Mandy Wilson. At our next meeting in Seattle in January 2012, Herbert Sauro was elected the SBOL chair, and two new editors were added: Matthew Pocock and Ernst Oberortner. New developers joining at these workshops included several representatives from industry, Kevin Clancy, Jacob Beal, Aaron Adler, and Fusun Yaman Sirin. New members hailing from Newcastle University included Anil Wipat, Matthew Pocock, and Goksel Misirli.

Development of the first software library (libSBOLj) based on the SBOL standard was initiated by Allan Kuchinsky, a research scientist from Agilent, at the 2011 meeting. By the time of the 2012 meeting, the first data exchange between software tools using SBOL was conducted when a design was passed from Newcastle University's VirtualParts Repository to Boston University's Eugene tool, and finally to University of Utah's iBioSim tool.

SBOL 1.0 was officially released in October 2011. In March 2012, SBOL 1.1 was released, the version that this document replaces. SBOL 1.1 did not make any major changes, but provided a number of small adjustments and clarifications, particularly around the annotation of sequences. Multi-institutional data exchange using SBOL 1.1 was later demonstrated in Nature Biotechnology Galdzicki et al. (2014).

While SBOL 1.1 had a number of significant advantages over the GenBank representation of DNA sequences, such as representing hierarchical organization of DNA components, it was still limited in other respects. The major topic of discussion at the 8th SBOL Workshop at Boston University in November 2012 was how to address these shortcomings through extensions. Several extensions were discussed at this meeting, such as a means to describe genetic regulation, which later became important classes in the current 2.0 specification.

A general framework for SBOL 2.0 emerged at the 9th SBOL workshop at Newcastle University in April 2013. Subsequently, Nicholas Roehner, Matthew Pocock, and Ernst Oberortner drafted a proposal for SBOL 2.0, and Nicholas presented this proposal at the SEED conference in Los Angeles in July 2014 Roehner et al. (2015). The proposed 2.0 data model was discussed over the course of the 10th, 11th, and 12th workshops. The actual specification document you are now reading was drafted at the 13th workshop in Wittenberg, Germany by the authors. The SBOL 2.0 data model presented here is essentially the result of these meetings and ongoing discussions conducted through the SBOL Developers mailing lists.

doi:10.2390/biecoll-jib-2015-272

The Computational Modeling in Biology Network (COMBINE) holds regular workshops where synthetic biologists and systems biologists can work toward a common goal of integrating biological knowledge through inter-operable and non-overlapping data standards. In April of 2014, several SBOL Developers attended a COMBINE workshop and then proposed that SBOL join this larger standards community. The proposal passed and SBOL workshops have been co-located with COMBINE meetings since the 11th workshop at the University of Southern California in August 2014.

Current development of this SBOL 2.0 specification is funded in large part by a grant from the National Science Foundation (DBI-1355909 and DBI-1356041). This document and the supporting software libraries are due in no small part to this support. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

5 SBOL Specification Vocabulary

5.1 Term Conventions

This document indicates requirement levels using the controlled vocabulary specified in IETF RFC 2119 and reiterated in BBF RFC 0. In particular, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

- The words "MUST", "REQUIRED", or "SHALL" mean that the item is an absolute requirement.
- The phrases "MUST NOT" or "SHALL NOT" mean that the item is an absolute prohibition.
- The word "SHOULD" or the adjective "RECOMMENDED" mean that there might exist valid reasons in particular circumstances to ignore a particular item, but the full implications need to be understood and carefully weighed before choosing a different course.
- The phrases "SHOULD NOT" or "NOT RECOMMENDED" mean that there might exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications needs to be understood and the case carefully weighed before implementing any behavior described with this label.
- The word "MAY" or the adjective "OPTIONAL" mean that an item is truly optional.

5.2 SBOL Class Names

SBOL defines the following “top-level” and dependent classes:

Collection: Represents a user-defined container for organizing a group of SBOL objects.

ComponentDefinition: Describes the structure of designed entities, such as DNA, RNA, and proteins, as well as other entities they interact with, such as small molecules or environmental properties.

- *Component*: Pointer class. Incorporates a child ComponentDefinition *by reference* into exactly one parent ComponentDefinition. Represents a specific occurrence or instance of an entity within the design of a more complex entity. Because the same definition might appear in multiple designs or multiple times in a single design, a single ComponentDefinition can have zero or more parent ComponentDefinitions, and each such parent-child link requires its own, distinct Component.
- *Location*: Specifies the base coordinates and orientation of a genetic feature on a DNA or RNA molecule or a residue or site on another sequential macromolecule such as a protein.
- *SequenceAnnotation*: Describes the Location of a notable sub-sequence found within the Sequence of a ComponentDefinition. Can also link to and effectively position a child Component.
- *SequenceConstraint*: Describes the relative spatial position and orientation of two Component objects that are contained within the same ComponentDefinition.

GenericTopLevel: Represents a data container that can contain custom data added by user applications.

Model: Links to quantitative or qualitative computational models that might be used to predict the functional behavior of a biological design.

ModuleDefinition: Describes a “system” design as a collection of biological components and their functional relationships.

- *FunctionalComponent*: Pointer class. Incorporates a child ComponentDefinition *by reference* into exactly one parent ModuleDefinition. Represents a specific occurrence or instance of an entity within

doi:10.2390/biecoll-jib-2015-272

the design of a system. Because the same definition might appear in multiple designs or multiple times in a single design, a single `ComponentDefinition` can have zero or more parent `ModuleDefinition`s, and each such parent-child link requires its own, distinct `FunctionalComponent`.

- *`Interaction`*: Describes a functional relationship between biological entities, such as regulatory activation or repression, or a biological process such as transcription or translation.
- *`MapsTo`*: When a design (`ComponentDefinition` or `ModuleDefinition`) includes another design as a sub-design, the parent design might need to refer to a `ComponentInstance` (either a `Component` or `FunctionalComponent`) in the sub-design. In this case, a `MapsTo` needs to be added to the instance for the sub-design, and this `MapsTo` needs to link between the `ComponentInstance` in the sub-design and a `ComponentInstance` in the parent design.
- *`Module`*: Pointer class. Incorporates a child `ModuleDefinition` *by reference* into exactly one parent `ModuleDefinition`. Represents a specific occurrence or instance of a subsystem within the design of a larger system. Because the same definition in multiple designs or multiple times in a single design, a single `ModuleDefinition` can have zero or more parent `ModuleDefinition`s, and each such parent-child link requires its own, distinct `Module`.
- *`Participation`*: Describes the role that a `FunctionalComponent` plays in an `Interaction`. For example, a transcription factor might participate in an `Interaction` as a repressor or as an activator.

`Sequence`: Generally represents a contiguous series of monomers in a macromolecular polymer such as DNA, RNA, or protein. A `Sequence` can also encode the atoms and bonds of a molecule with non-linear structure (see Section 7.6).

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

6 Overview of SBOL

Synthetic biology designs can be described using:

- Structural terms, e.g., a set of annotated sequences or information about the chemical makeup of components.
- Functional terms, e.g., the way that components might interact with each other and the overall behavior of a design.

In broad strokes, the prior SBOL 1.1 standard focused on conveying physical, structural information, whereas SBOL 2.0 expands the scope to include functional aspects as well. The physical information about a designed genetic construct includes the order of its constituents and their descriptions. Specifying the exact locations of these constituents and their sequences allow genetic constructs to be defined unambiguously and reused in other designs. SBOL 2.0 extends SBOL 1.1 in several ways: it extends physical descriptions to include entities beyond DNA sequences, and it supports functional descriptions of designs.

As an example, consider the design of an expression cassette, such as the one found in the plasmid pUC18 Norrander et al. (1983). This device is designed to detect successful versus unsuccessful molecular cloning. As an overall system, the device is designed to grow either blue-colored (unsuccessful) or white-colored (successful) colonies in the presence of IPTG and the chemical X-gal. Internally, the device has a number of parts, including a promoter, the lac repressor binding site, and the lacZ coding sequence. These parts have specific component-level interactions with IPTG and X-gal, as well as native host gene products, transcriptional machinery and translational machinery that collectively cause the desired system-level behavior.

Knowledge of how such a device functions within the context of a host and how it might be adapted to new experimental applications has generally been passed on through working with fellow scientists or reading articles in papers and books. But there has been no systematic way to communicate the integration of sequences with functional designs, so users typically have had to look in many different places to develop an understanding of a system. The SBOL 2.0 standard allows designers to describe these functional characteristics and connect them to the physical parts and sequences that make up the design.

SBOL 2.0 includes two main classes that match the structural/functional distinction above:

- The `ComponentDefinition` object describes the physical aspects of the designed system, such as its DNA or RNA sequences, and the physical relationships among sub-components, as when one sequence contains another as a sub-sequence.
- The `ModuleDefinition` object describes interactions of the designed system, such as specific binding relationships and repression and activation relationships.

Figure 2 shows a simplified view of these classes, as well as other helper classes in SBOL. To continue with the pUC18 example, the description would begin with a top-level `ModuleDefinition`. The `ModuleDefinition` specifies the structural elements that make up the cassette by referencing a number of `ComponentDefinition` objects. These would include the DNA component for the promoter and the small molecule component for IPTG, for example. The `ComponentDefinition` objects can be organized hierarchically. For example, the plasmid `ComponentDefinition` might reference `ComponentDefinition`s for the promoter, coding sequence, etc. Each `ComponentDefinition` object can also include the actual `Sequence` information (if available), as well as `SequenceAnnotation` objects that identify the locations of the promoters, coding sequences, etc., on the `Sequence`. In order to specify functional information, the `ModuleDefinition` can specify `Interaction` objects that describe any qualitative relationships among components, such as how IPTG and X-gal interact with the gene products. Finally, a `ModuleDefinition` object can point to a `Model` object that provides a reference to a complete computational model using a language such as SBML, CellML, Matlab, etc. Finally, all the of elements of the genetic design can be grouped together within a `Collection`.

doi:10.2390/biecoll-jib-2015-272

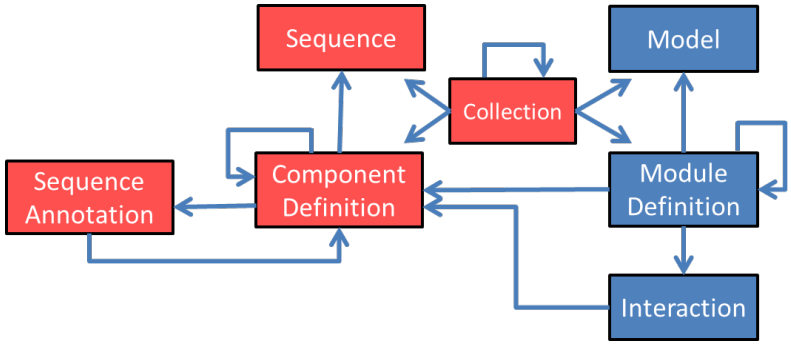


Figure 2: *Main classes of information represented by the SBOL standard, and their relationships. Red boxes are classes from the SBOL 1.1 that focused on structure, whereas blue classes are some of the new classes that support the functional aspects of designs.*

Whereas Figure 2 provides a broad overview of SBOL, Figure 3 provides a detailed, implementation-level overview of the class structure for the SBOL 2.0 data model. This figure relies on the semantics of the *Unified Modeling Language* (UML), which will be presented in more detail in the next section. Figure 3 distinguishes between *top level* classes, in green, and other supporting classes (note that Figure 2 also includes all of the top level classes). In Figure 3, dashed arcs represent "refersTo", whereas a solid arrow represents ownership. In UML, the meaning of ownership is that if a parent class is deleted, so are all of its owned children. Thus, a `Collection` does not own its `ComponentDefinition` objects, because these can stand on their own. All of the supporting classes (in orange) have to be owned by some top-level class, directly or indirectly.

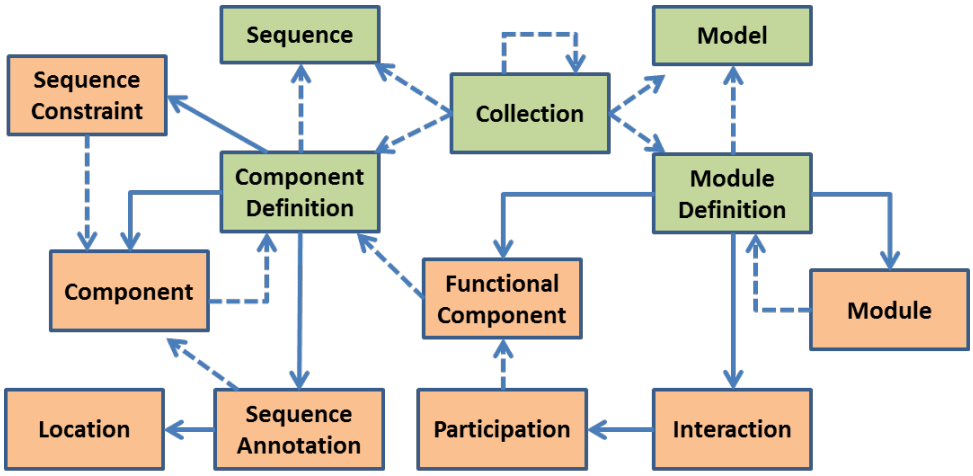


Figure 3: *Main classes of information represented by the SBOL 2.0 standard, and their relationships. Green boxes are "top level" classes, while the other classes are in support of these classes. Solid arrows indicates ownership, whereas a dashed arrow indicates that one class refers to an object of another class.*

Figure 3 additionally shows that when it is possible to incorporate a single object into multiple parents, we always incorporate that object by reference. We do not directly incorporate it by copy, because when an object is used many times, keeping many copies becomes spatially inefficient and difficult to maintain. Instead, each

doi:10.2390/biecoll-jib-2015-272

reference is handled by a pointer object. Pointers refer from a parent to a child. There are three distinct pointer classes: `Component`, `Module`, and `FunctionalComponent`. A `Component` points from a `ComponentDefinition` to a child `ComponentDefinition`, incorporating it by reference into the parent structure. A `Module` points from a `ModuleDefinition` to a child `ModuleDefinition`, likewise incorporating the child by reference into the parent system. Similarly, a parent `ModuleDefinition` on the functional side of a model might incorporate a child `ComponentDefinition` from the model's physical side by means of a `FunctionalComponent` reference. These three pointer classes allow the efficient reuse of definitions in multiple locations.

SBOL 2.0 provides a few helper classes. `Location` generalizes the positioning information from SBOL 1.1 to allow discontinuous ranges and cuts to be annotated. `SequenceConstraint` generalizes the relative positioning information among `Component`s. There are also `Participation`s, which allow `Interaction` objects to specify the roles of their participants while referencing the `FunctionalComponent`s, so that these can stand on their own. Additionally, there is the `MapsTo` class (not shown), which enables connections to be made between `Component`s and `FunctionalComponent`s across various levels of the design hierarchy. The next section provides complete definitions and details for all of these classes.

There is one final, critical element of SBOL 2.0: its extension mechanism. This extension mechanism enables the storage of application specific information within an SBOL document. It is also intended to support the prototyping of data representations whose format is not yet a matter of consensus within the community. In particular, each SBOL entity can be annotated using the *Resource Description Framework* (RDF). Moreover, application specific entities in the form of RDF documents can be included as `GenericTopLevel` entities. SBOL libraries make these annotations and entities available to tools as generic properties and objects that are preserved during subsequent read and write operations.

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

7 SBOL Data Model

In this section, we describe the types of biological design data that can belong to an SBOL document and the relationships between these data types. The SBOL data model is specified using Unified Modeling Language (UML) 2.0 diagrams (OMG 2005). Subsections Section 7.1, Section 7.2, Section 7.3 review the basics of UML diagrams and explain the naming conventions and generic data types used in this specification. The remaining sections then describe the SBOL data model in detail. Complete SBOL examples and best practices when using the standard can be found in Section 9 and Section 11, respectively.

7.1 Understanding the UML Diagrams

The types of biological design data modeled by SBOL are commonly referred to as *classes*, especially when discussing the details of software implementation. Each SBOL class can be instantiated by many SBOL objects. These objects MAY contain data that differ in content, but they MUST agree on the type and form of their data as dictated by their common class. Classes are represented in UML diagrams as rectangles labeled at the top with class names.

Classes can be connected to other classes by association properties, which are represented in UML diagrams as arrows. These arrows are labeled with data cardinalities in order to indicate how many values a given association property can possess (see below). The remaining (non-association) properties of a class are listed below its name. Each of the latter properties is labeled with its data type and cardinality.

In the case of an association property, the class from which the arrow originates is the owner of the association property. A diamond at the origin of the arrow indicates the type of association. Open-faced diamonds indicate shared aggregation, in which the owner of the association property exists independently of its value. In the SBOL data model, the value of an association property MUST be a `URI` or set of `URI`s that refer to SBOL objects belonging to the class at the tip of the arrow.

By contrast, filled diamonds indicate composite aggregation, also known as a part-whole relationship, in which the value of the association property MUST NOT exist independently of its owner. In addition, in the SBOL data model, it is REQUIRED that the value of each composite aggregation property is a unique SBOL object (that is, not the value for more than one such property). Note that in all cases, composite aggregation is used in such a way that there SHOULD NOT be duplication of such objects.

All SBOL properties are labeled with one of several restrictions on data cardinality. These are:

- 1 - REQUIRED, one: there MUST be exactly one value for this property.
- 0…1 - OPTIONAL: there MAY be a single value for this property, or it MAY be absent.
- 0…* - unbounded: there MAY be any number of values for this property, including none.
- 1…* - REQUIRED, unbounded: there MAY be any number of values for this property, as long as there is at least one.
- *n*…* - at least: there MUST be at least *n* values for this property.

Finally, classes can inherit the properties of other classes. Inheritance relationships are represented in UML diagrams as open-faced, triangular arrows that point from the inheriting class to the inherited class. Some classes in the SBOL data model cannot be instantiated as objects and exist only to group common properties for inheritance. These classes have italicized names and are known as abstract classes.

7.2 Naming and Font Conventions

SBOL classes are named using upper "camel case," meaning that each word is capitalized and all words are run together without spaces, e.g. `Identified`, `SequenceAnnotation`. Properties, on the other hand, are named using lower camel case, meaning that they begin lowercase (e.g., `identity`) but if they consist of multiple words, all words

doi:10.2390/biecoll-jib-2015-272

after the first begin with an uppercase letter (e.g., persistentIdentity).

Within the SBOL data model, each property is given a singular or plural name in accordance with its data cardinalities. The forms of these names follow the usual rules of English grammar. For example, sequenceAnnotation is the singular form of sequenceAnnotations.

SBOL properties are always given singular names, however, when SBOL objects are serialized (using *Resource Description Framework* (RDF) as described in Section 10). This is because the SBOL data model does not contain classes that correspond directly to the RDF elements that group other elements into ordered or unordered sets. Consequently, if an SBOL property has multiple values, then it is serialized as multiple property entries, each with a singular name and a single value. For example, if an SBOL property has five values, then its serialization contains five RDF triples, each with a singular predicate name and one of the five values as its object.

7.3 Data Types

When SBOL use simple "primitive" data types such as Strings or Integers, these are defined as the following specific formal types:

- String: http://www.w3.org/TR/xmlschema11-2/#string
  Example: "LacI coding sequence"
- Integer: http://www.w3.org/TR/xmlschema11-2/#integer
  Example: 3
- Double: http://www.w3.org/TR/xmlschema11-2/#double
  Example: 3.14159
- Boolean: http://www.w3.org/TR/xmlschema11-2/#boolean
  Example: `true`

The term literal is used to denote an object that can be any of the four types listed above. In addition to the simple types listed above, SBOL also uses objects with types *Uniform Resource Identifier* (URI) and *XML Qualified Name* (QName):

- URI: http://www.w3.org/TR/xmlschema11-2/#anyURI
  Example: `http://www.partsregistry.org/Part:BBa_J23119`
- QName: http://www.w3.org/TR/xmlschema11-2/#QName
  Example: `myapp:Datasheet` where `myapp="http://www.myapp.org/"`

Note that, in compliance with RDF standards, URIs are generally serialized using an `rdf:resource` property, e.g.: `rdf:resource="http://www.partsregistry.org/Part:BBa_J23119"`

It is important to realize that in RDF, a URI might or might not be a resolvable URL (web address). A URI is always a globally unique identifier within a structured namespace. In some cases, that name is also a reference to (or within) a document, and in some cases that document can also be retrieved (e.g., using a web browser).

7.4 Identified

All SBOL-defined classes are directly or indirectly derived from the Identified abstract class. This inheritance means that all SBOL objects are uniquely identified using URIs that uniquely refer to these objects within an SBOL document or at locations on the World Wide Web.

As shown in Figure 4, the Identified class includes the following properties: identity, persistentIdentity, version, wasDerivedFrom, name, description, and annotations. The latter property is described separately in Section 7.11.

doi:10.2390/biecoll-jib-2015-272

When an SBOL resource reference takes the form of a URI, that URI can either be the value of an identity property or the value of a persistentIdentity property. If the URI is equal to the value of an identity property, then it is guaranteed to be unique, and it refers to precisely one SBOL object with that URI. If the URI is equal to the value of a persistentIdentity property, then it MAY refer to multiple SBOL objects that are different "versions" of each other. These objects SHOULD be compared to one another to determine which single object the URI resolves to (normally the most recent version - see Section 7.4). Throughout this document, when a URI is used to refer to an SBOL object, it could fall into either of these cases.

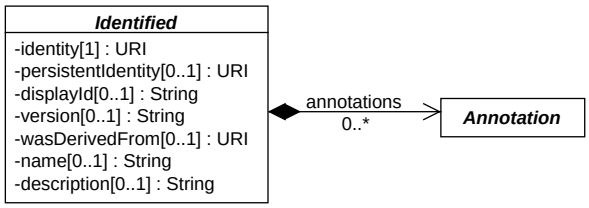


Figure 4: *Diagram of the Identified abstract class and its associated properties*

The identity property

The identity property is REQUIRED by all Identified objects and has a data type of URI. A given Identified object's identity URI MUST be globally unique among all other identity URIs. It is also highly RECOMMENDED that the URI structure follows the recommended best practices for compliant URIs specified in Section 11.2.

Although most SBOL properties are defined by SBOL and serialized with its namespace, the identity property is defined by the analogous RDF about property and is serialized with the RDF namespace as follows:

http://www.w3.org/1999/02/22-rdf-syntax-ns#about.

The use of about is expressly for the purpose of making SBOL compliant with pre-existing standards: when you see about in an SBOL document, you SHOULD interpret it as meaning identity.

The persistentIdentity property

The persistentIdentity property is OPTIONAL and has a data type of URI. This URI serves to uniquely refer to a set of SBOL objects that are different versions of each other.

An Identified object MUST be referred to using either its identity URI or its persistentIdentity URI.

The displayId property

The displayId property is an OPTIONAL identifier with a data type of String. This property is intended to be an intermediate between name and identity that is machine-readable, but more human-readable than the full URI of an identity.

If the displayId property is used, then its String value SHOULD be locally unique (global uniqueness is not necessary) and MUST be composed of only alphanumeric or underscore characters and MUST NOT begin with a digit.

The version property

The version property is OPTIONAL and has a data type of String. This property can be used to compare two SBOL objects with the same persistentIdentity.

doi:10.2390/biecoll-jib-2015-272

If the version property is used, then it is RECOMMENDED that version numbering follow the conventions of semantic versioning (http://semver.org/), particularly as implemented by Maven (http://maven.apache.org/). This convention represents versions as sequences of numbers and qualifiers that are separated by the characters "." and "-" and are compared in lexicographical order (for example, 1 < 1.3.1 < 2.0-beta). For a full explanation, see the linked resources.

The wasDerivedFrom property

The wasDerivedFrom property is OPTIONAL and has a data type of URI. An SBOL object with this property refers to another SBOL object or non-SBOL resource from which this object was derived.

If the wasDerivedFrom property of an SBOL object *A* that refers to an SBOL object *B* has an identical persistentIdentity, and both *A* and *B* have a version, then the version of *B* MUST precede that of *A*. In addition, an SBOL object MUST NOT refer to itself via its own wasDerivedFrom property or form a cyclical chain of references via its wasDerivedFrom property and those of other SBOL objects. For example, the reference chain "*A* was derived from *B* and *B* was derived from *A*" is cyclical.

The name property

The name property is OPTIONAL and has a data type of String. This property is intended to be displayed to a human when visualizing an Identified object.

If an Identified object lacks a name, then software tools SHOULD instead display the object's displayId or identity. It is RECOMMENDED that software tools give users the ability to switch perspectives between name properties that are human-readable and displayId properties that are less human-readable, but are more likely to be unique.

The description property

The description property is OPTIONAL and has a data type of String. This property is intended to contain a more thorough text description of an Identified object.

The annotations property

The annotations property is OPTIONAL and MAY specify a set of Annotation objects that are contained by the Identified object. The Annotation class is described in more detail in Section Section 7.11.1.

Serialization

No complete serialization is defined for Identified, since this class is only used indirectly through its child classes. Any such child class, however, has the following form for serializing properties inherited from Identified, where CLASS_NAME is replaced by the name of the class:

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:pr="http://partsregistry.org" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.
      org/dc/terms/" xmlns:prov="http://www.w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:CLASS_NAME rdf:about="...">
    zero or one <sbol:persistentIdentity rdf:resource="..."/> element
    zero or one <sbol:displayId>...</sbol:displayId> element
    zero or one <sbol:version>...</sbol:version> element
    zero or one <prov:wasDerivedFrom rdf:resource="..."/> element
    zero or one <dcterms:title>...</dcterms:title> element
    zero or one <dcterms:description>...</dcterms:description> element
              ...
  </sbol:CLASS_NAME>
  ...
</rdf:RDF>
```


doi:10.2390/biecoll-jib-2015-272

Note that several of the properties are not in the `sbol` namespace, but are mapped to standardized terms defined elsewhere:

- `identity` is serialized as `rdf:about`
- `wasDerivedFrom` is serialized as `prov:wasDerivedFrom`
- `name` is serialized as `dcterms:title`
- `description` is serialized as `dcterms:description`

7.5 TopLevel

`TopLevel` is an abstract class that is extended by any `Identified` class that can be found at the top level of an SBOL document or file. In other words, `TopLevel` objects are not nested inside any other object via a composite aggregation or black diamond arrow association property. Instead of nesting, composite `TopLevel` objects refer to subordinate `TopLevel` objects by their `URI`s using shared aggregation or white diamond arrow association properties. The `TopLevel` classes defined in this specification are `Sequence`, `ComponentDefinition`, `Model`, `ModuleDefinition`, `Collection`, and `GenericTopLevel` (Figure 5).

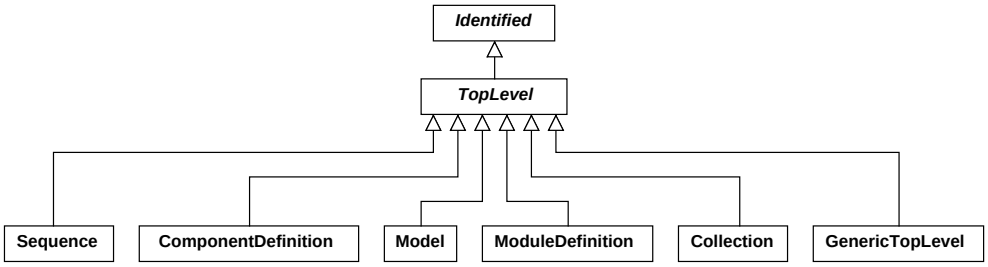


Figure 5: *Classes that inherit from the `TopLevel` abstract class.*

Serialization

No serialization is defined for `TopLevel`, since this class has no properties of its own and is only used indirectly through its child classes. All `TopLevel` classes are serialized one level beneath the RDF document root.

7.6 Sequence

The purpose of the `Sequence` class is to represent the primary structure of a `ComponentDefinition` object and the manner in which it is encoded. This representation is accomplished by means of the `elements` property and `encoding` property (Figure 6).

The `elements` property

The `elements` property is a REQUIRED `String` of characters that represents the constituents of a biological or chemical molecule. For example, these characters could represent the nucleotide bases of a molecule of DNA, the amino acid residues of a protein, or the atoms and chemical bonds of a small molecule.

doi:10.2390/biecoll-jib-2015-272

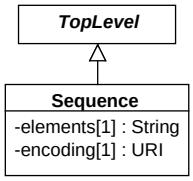


Figure 6: *Diagram of the Sequence class and its associated properties.*

The encoding property

The encoding property is REQUIRED and has a data type of URI. This property MUST indicate how the elements property of a Sequence MUST be formed and interpreted.

For example, the elements property of a Sequence with an IUPAC DNA encoding property MUST contain characters that represent nucleotide bases, such as a, t, c, and g. The elements property of a Sequence with a Simplified Molecular-Input Line-Entry System (SMILES) encoding, on the other hand, MUST contain characters that represent atoms and chemical bonds, such as C, N, O, and =.

Table 1 provides a list of possible URI values for the encoding property. The terms in Table 1 are organized by the type of ComponentDefinition (see Table 2) that typically refer to a Sequence with such an encoding. When the encoding of a Sequence is well described by one of the URIs in Table 1, it MUST contain that URI.

Encoding	URI	ComponentDefinition Type
IUPAC DNA, RNA	http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html	DNA, RNA
IUPAC Protein	http://www.chem.qmul.ac.uk/iupac/AminoAcid/	Protein
SMILES	http://www.opensmiles.org/opensmiles.html	SmallMolecule

Table 1: *URIs for specifying the encoding property of a Sequence, organized by the type of ComponentDefinition (see Table 2) that typically refer to a Sequence with such an encoding.*

Serialization

The serialization of a Sequence MUST have the following form:

```
<sbol:Sequence rdf:about="...">
  ... properties inherited from identified ...
  one <sbol:elements>...</sbol:elements> element
  one <sbol:encoding rdf:resource="..."/> element
</sbol:Sequence>
```

The example below shows the serialization of the Sequence for a promoter. The nucleotide bases of the Sequence are serialized as the String value of its elements property, while its IUPAC DNA encoding is serialized as the URI value of its encoding property.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://parts.igem.org/Part:BBa_J23119:Design"/>
    <sbol:elements>ttgacagctagctcagtcctaggtataatgctagc</sbol:elements>
```


doi:10.2390/biecoll-jib-2015-272

```
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
</rdf:RDF>
```

7.7 ComponentDefinition

The `ComponentDefinition` class represents the structural entities of a biological design. The primary usage of this class is to represent structural entities with designed sequences, such as DNA, RNA, and proteins, but it can also be used to represent any other entity that is part of a design, such as small molecules, molecular complexes, and light.

As shown in Figure 7, the `ComponentDefinition` class describes a structural design entity using the following properties: `types`, `roles`, and `sequences`. In addition, this class has properties for describing and organizing the substructure of said design entity, including `components`, `sequenceAnnotations`, and `sequenceConstraints`.

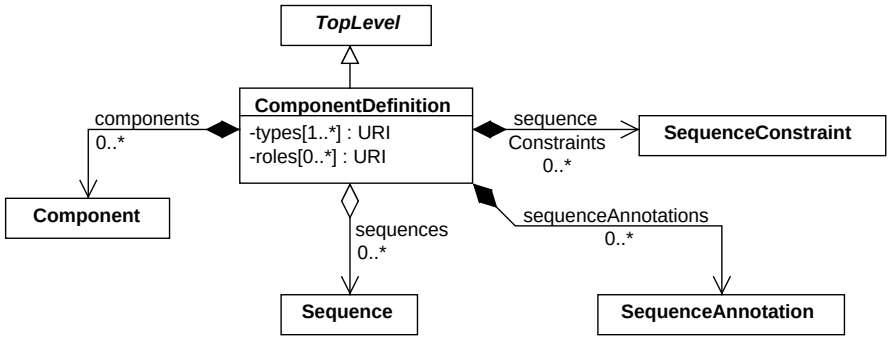


Figure 7: *Diagram of the `ComponentDefinition` class and its associated properties.*

The `types` property

The `types` property is a REQUIRED set of `URI`s that specifies the category of biochemical or physical entity (for example DNA, protein, or small molecule) that a `ComponentDefinition` object abstracts for the purpose of engineering design.

The `types` property of every `ComponentDefinition` MUST contain one or more `URI`s that MUST identify terms from appropriate ontologies, such as the BioPAX ontology or the ontology of Chemical Entities of Biological Interest (ChEBI). Table 2 provides a list of possible ontology terms for the `types` property and their `URI`s. In order to maximize the compatibility of designs, any `ComponentDefinition` that can be well-described by one of the terms in Table 2 MUST use the `URI` for that term as one of its `types`. Finally, if the `types` property contains multiple `URI`s, then they MUST identify non-conflicting terms (otherwise, it might not be clear how to interpret them). For example, the BioPAX terms provided by Table 2 would conflict because they specify classes of biochemical entities with different molecular structures.

The `roles` property

The `roles` property is an OPTIONAL set of `URI`s that clarifies the potential function of the entity represented by a `ComponentDefinition` in a biochemical or physical context.

The `roles` property of a `ComponentDefinition` MAY contain one or more `URI`s that MUST identify terms from ontologies that are consistent with the `types` property of the `ComponentDefinition`. For example, the `roles` property

doi:10.2390/biecoll-jib-2015-272

ComponentDefinition Type	URI for BioPAX Term
DNA	http://www.biopax.org/release/biopax-level3.owl#DnaRegion
RNA	http://www.biopax.org/release/biopax-level3.owl#RnaRegion
Protein	http://www.biopax.org/release/biopax-level3.owl#Protein
Small Molecule	http://www.biopax.org/release/biopax-level3.owl#SmallMolecule
Complex	http://www.biopax.org/release/biopax-level3.owl#Complex

Table 2: *BioPAX terms to specify the* `types` *property of a* `ComponentDefinition`*.*

of a DNA or RNA `ComponentDefinition` could contain `URI`s identifying terms from the Sequence Ontology (SO). Table 3 contains a list of possible ontology terms for the `roles` property and their `URI`s. These terms are organized by the type of `ComponentDefinition` to which they SHOULD apply (see Table 2). Any `ComponentDefinition` that can be well-described by one of the terms in Table 3 MUST use the `URI` for that term as one of its `roles`.

ComponentDefinition Role	URI for Ontology Term	ComponentDefinition Type
Promoter	http://identifiers.org/so/SO:0000167	DNA
RBS	http://identifiers.org/so/SO:0000139	DNA
CDS	http://identifiers.org/so/SO:0000316	DNA
Terminator	http://identifiers.org/so/SO:0000141	DNA
Gene	http://identifiers.org/so/SO:0000704	DNA
Operator	http://identifiers.org/so/SO:0000057	DNA
Engineered Gene	http://identifiers.org/so/SO:0000280	DNA
mRNA	http://identifiers.org/so/SO:0000234	RNA
Effector	http://identifiers.org/chebi/CHEBI:35224	Small Molecule

Table 3: *Ontology terms to specify the* `roles` *property of a* `ComponentDefinition`*, organized by the type of* `ComponentDefinition` *to which they are intended to apply (see Table 2).*

The `sequences` *property*

The `sequences` property is OPTIONAL and MAY include a set of `URI`s that refer to `Sequence` objects. These objects define the primary structure of the `ComponentDefinition`.

Many `ComponentDefinition` objects will refer to precisely one `Sequence` object. For certain use cases, however, it can be appropriate to refer to multiple `Sequence` objects. For example, a user might wish to provide two different representations of the structure of a DNA `ComponentDefinition`, one that represents its structure at the level of nucleotide bases and one that represents its structure at the level of atoms and bonds.

If a `ComponentDefinition` refers to more than one `Sequence` object, then these objects MUST be consistent with each other, such that well-defined mappings exist between their `elements` properties in accordance with their `encoding` properties. Furthermore, these objects MUST NOT have conflicting `encoding` properties. For example, the `IUPAC` `encoding` properties provided by Table 1 conflict with each other because they do not specify how to encode the same class of biochemical entity. The `SMILES` `encoding`, however, does not conflict with them because it specifies how to encode biochemical entities in general, which includes DNA, RNA, and proteins. If a `ComponentDefinition` refers to more than one `Sequence` with the same `encoding`, then the `elements` of these `Sequence` objects SHOULD have equal lengths. These requirements and best practices are intended to make it easier for software tools to locate any regions specified by the `SequenceAnnotation` objects of a `ComponentDefinition` on its associated `Sequence` objects, as well as validate whether its `Sequence` objects are consistent with those associated with any `ComponentDefinition` objects that it composes via its `Component` objects.

Finally, if a `ComponentDefinition` refers to one or more `Sequence` objects and its `types` property refers to a term from Table 2, then one of these `Sequence` objects MUST have the `encoding` that is cross-listed with this term in

doi:10.2390/biecoll-jib-2015-272

Table 1. Conversely, if a ComponentDefinition refers to a Sequence with an encoding from Table 1, then its types property MUST refer to the term from Table 2 that is cross-listed with this encoding in Table 1. For example, if the types property of a ComponentDefinition refers to the BioPAX term for DNA, then one of the Sequence objects to which it refers (if any) MUST have an IUPAC DNA encoding, and if a ComponentDefinition refers to a Sequence with an IUPAC DNA encoding, then its types property MUST refer to the BioPAX term for DNA. These requirements are meant to provide for some degree of consistency between the types property of a ComponentDefinition and the encoding properties of the Sequence objects to which the ComponentDefinition refers.

The components property

The components property is OPTIONAL and MAY specify a set of Component objects that are contained by the ComponentDefinition. The set of relations between Component and ComponentDefinition objects is strictly acyclic (see Section 7.7.1).

While the ComponentDefinition class is analogous to a blueprint or specification sheet for a biological part, the Component class represents the specific occurrence of a part within a design. Hence, this class allows a biological design to include multiple instances of a particular part (defined by reference to the same ComponentDefinition). For example, the ComponentDefinition of a polycistronic gene could contain two Component objects that refer to the same ComponentDefinition of a CDS.

The components properties of ComponentDefinition objects can be used to construct a hierarchy of Component and ComponentDefinition objects. If a ComponentDefinition in such a hierarchy refers to one or more Sequence objects, and there exist ComponentDefinition objects lower in the hierarchy that refer to Sequence objects with the same encoding, then the elements properties of these Sequence objects SHOULD be consistent with each other, such that well-defined mappings exist from the "lower level" elements to the "higher level" elements in accordance with their shared encoding properties. This mapping is also subject to any restrictions on the positions of the Component objects in the hierarchy that are imposed by the SequenceAnnotation or SequenceConstraint objects contained by the ComponentDefinition objects in the hierarchy.

A DNA ComponentDefinition, for example, could refer to a Sequence with an IUPAC DNA encoding and an elements String of "gattaca." In turn, this ComponentDefinition could contain a Component that refers to a "lower level" ComponentDefinition that also refers to a Sequence with an IUPAC DNA encoding. Consequently, a consistent elements String of this "lower level" Sequence could be "gatta," or perhaps "tgta" if the Component is positioned by a SequenceAnnotation that contains a Location with an orientation of "reverse complement" (see Section 7.7.5).

The sequenceAnnotations property

The sequenceAnnotations property is OPTIONAL and MAY contain a set of SequenceAnnotation objects. Each SequenceAnnotation specifies and describes a potentially discontiguous region on the Sequence objects referred to by the ComponentDefinition.

In addition, each SequenceAnnotation can position a Component of the ComponentDefinition at the region specified by its Location objects (see Section 7.7.5). The sequenceAnnotations property MUST NOT contain two or more SequenceAnnotation objects that refer to the same Component in this way.

Finally, as a best practice, if a ComponentDefinition refers to a Sequence with an IUPAC encoding from Table 1, then each of its SequenceAnnotation objects that contains a Range or Cut SHOULD specify a region on the elements of this Sequence. For example, the ComponentDefinition of a eukaryotic gene could refer to a Sequence with an IUPAC DNA encoding. In order to specify the discontiguous region occupied by its CDS, this gene ComponentDefinition would need a SequenceAnnotation that contains one or more Range objects, each one specifying start and end positions that correspond to indices of the elements of its DNA Sequence.

doi:10.2390/biecoll-jib-2015-272

The `sequenceConstraints` property

The `sequenceConstraints` property is OPTIONAL and MAY contain a set of `SequenceConstraint` objects. These objects describe any restrictions on the relative, sequence-based positions and/or orientations of the `Component` objects contained by the `ComponentDefinition`. For example, the `ComponentDefinition` of a gene might specify that the position of its promoter `Component` precedes that of its CDS `Component`. This is particularly useful when a `ComponentDefinition` lacks a `Sequence` and therefore cannot specify the precise, sequence-based positions of its `Component` objects using `SequenceAnnotation` objects.

Serialization

The serialization of a `ComponentDefinition` MUST have the form below. The `components`, `sequenceConstraints`, `sequenceAnnotations`, and `sequences` properties of a `ComponentDefinition` contain or reference objects belonging to the appropriate SBOL classes as their values, while the `types` and `roles` properties contain `URI`s that identify ontology terms as their values. As shown below, each of these objects and `URI`s is serialized as part of an implicit set of SBOL properties with singular rather then plural names. In particular, each object is serialized as a RDF/XML node nested within a property, while each `URI` (except the `identity`) is serialized as a `rdf:resource` on a property.

```
<sbol:ComponentDefinition rdf:about="...">
  ... properties inherited from identified ...
  zero or more  <sbol:sequence rdf:resource="..."/> element
  one or more   <sbol:type rdf:resource="..."/> elements
  zero or more  <sbol:role rdf:resource="..."/> elements
  zero or more  <sbol:component>
                  <sbol:Component rdf:about="...">...</sbol:Component>
                </sbol:component> elements
  zero or more <sbol:sequenceAnnotation>
                  <sbol:SequenceAnnotation rdf:about="...">...</sbol:SequenceAnnotation>
                </sbol:sequenceAnnotation> elements
  zero or more <sbol:sequenceConstraint>
                  <sbol:SequenceConstraint rdf:about="...">...</sbol:SequenceConstraint>
                </sbol:sequenceConstraint> elements
</sbol:ComponentDefinition>
```

The example below shows the serialization for the `ComponentDefinition` of a promoter. The BioPAX term `DnaRegion` and the ChEBI term `CHEBI:4705` (`double-stranded DNA`) are used to indicate that the type of biological entity represented by this `ComponentDefinition` is DNA. Its role is specified using the SO terms `SO:0000167` (`promoter`) and the more specific `SO:0000613` (`bacterial_RNApol_promoter`).

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
     w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://partsregistry.org/Part:BBa_J23119"/>
    <dcterms:title>J23119 promoter</dcterms:title>
    <dcterms:description>Constitutive promoter</dcterms:description>
    <sbol:type rdf:resource="http://identifiers.org/chebi/CHEBI:4705"/>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000613"/>
    <sbol:sequence rdf:resource="http://partsregistry.org/seq/BBa_J23119"/>
  </sbol:ComponentDefinition>
</rdf:RDF>
```

7.7.1 ComponentInstance

The `ComponentInstance` abstract class is inherited by SBOL classes that represent the usage or occurrence of a `ComponentDefinition` within a larger design (that is, another `ComponentDefinition` or `ModuleDefinition`).

doi:10.2390/biecoll-jib-2015-272

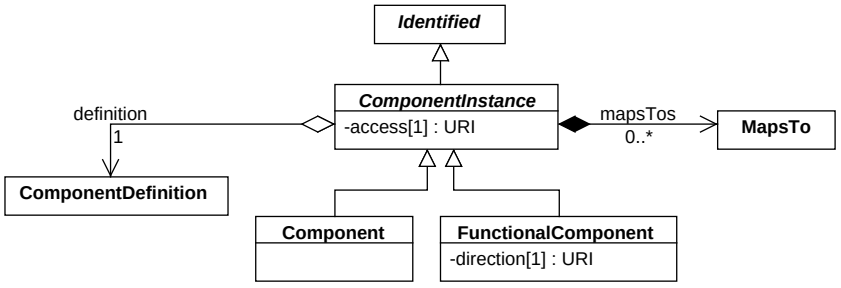


Figure 8: *Diagram of the* `ComponentInstance` *class and its associated properties.*

Currently, there are two subclasses of `ComponentInstance`:

- The `Component` class is used to specify the structural usage of a `ComponentDefinition` inside another `ComponentDefinition` via the `components` property.
- The `FunctionalComponent` class is used to specify the functional usage of a `ComponentDefinition` inside a `ModuleDefinition` via the `functionalComponents` property. This class is described in Section 7.9.2.

The `definition` property

The `definition` property is a REQUIRED `URI` that refers to the `ComponentDefinition` of the `ComponentInstance`. As described in the previous section, this `ComponentDefinition` effectively provides information about the `types` and `roles` of the `ComponentInstance`.

The `definition` property MUST NOT refer to the same `ComponentDefinition` as the one that contains the `ComponentInstance`. Furthermore, `ComponentInstance` objects MUST NOT form a cyclical chain of references via their `definition` properties and the `ComponentDefinition` objects that contain them. For example, consider the `ComponentInstance` objects A and B and the `ComponentDefinition` objects X and Y. The reference chain "X contains A, A is defined by Y, Y contains B, and B is defined by X" is cyclical.

The `mapsTos` property

The `mapsTos` property is OPTIONAL and MAY contain a set of `MapsTo` objects that refer to and link together `ComponentInstance` objects (both `Component` objects and `FunctionalComponent` objects) within a larger design.

Section 7.7.3 contains a more detailed description of the `MapsTo` class.

The `access` property

The `access` property is a REQUIRED `URI` that indicates whether the `ComponentInstance` can be referred to remotely by a `MapsTo` on another `ComponentInstance` or `Module` contained by a different parent `ComponentDefinition` or `ModuleDefinition` (one that does not contain this `ComponentInstance`).

Table 4 provides a list of REQUIRED `access` `URI`s. The value of the `access` property MUST be one of these `URI`s.

In some cases, a designer might want to set the `access` property of a `ComponentInstance` such that others cannot map to the `ComponentInstance` when they reuse its parent `ComponentDefinition`. For example, a designer who is concerned about retroactivity might set the `access` of the `ComponentInstance` to "private" in order to prevent its mapping to another `ComponentInstance` that participates in a new `Interaction` as part of a composite design.

doi:10.2390/biecoll-jib-2015-272

Access URI	Description
http://sbols.org/v2#public	The ComponentInstance MAY be referred to by remote MapsTo objects.
http://sbols.org/v2#private	The ComponentInstance MUST NOT be referred to by remote MapsTo objects.

Table 4: *REQUIRED URIs for the access property.*

Serialization

No serialization is defined for the ComponentInstance class, since this class is only used indirectly through the Component and FunctionalComponent subclasses.

7.7.2 Component

The Component class is used to compose ComponentDefinition objects into a structural hierarchy. For example, the ComponentDefinition of a gene could contain four Component objects: a promoter, RBS, CDS, and terminator. In turn, the ComponentDefinition of the promoter Component could contain Component objects defined as various operator sites.

Serialization

The serialization of a Component MUST have the following form:

```
<sbol:Component rdf:about="...">
  ... properties inherited from identified ...
  one          <sbol:access rdf:resource="..."/> element
  one          <sbol:definition rdf:resource="..."/> element
  zero or more <sbol:mapsTo rdf:resource="..."/> elements
</sbol:Component>
```

The example below shows the serialization of a Component that represents an instance of a promoter:

```
<sbol:Component rdf:about="http://partsregistry.org/cd/BBa_F2620/pLuxR">
  <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/pLuxR"/>
  <sbol:displayId>pLuxR</sbol:displayId>
  <sbol:access rdf:resource="http://sbols.org/v2#public"/>
  <sbol:definition rdf:resource="http://partsregistry.org/cd/BBa_R0062"/>
</sbol:Component>
```

7.7.3 MapsTo

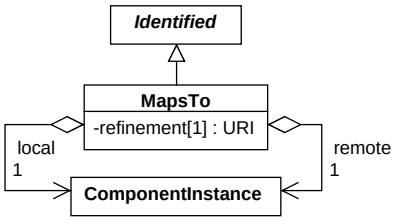


Figure 9: *Diagram of the MapsTo class and its associated properties.*

When ComponentDefinition and ModuleDefinition objects are composed into structural and functional hierar-

doi:10.2390/biecoll-jib-2015-272

chies using ComponentInstance and Module objects, it is often the case that some ComponentInstance objects are intended to represent the same entity in the overall design. The purpose of the MapsTo class is to make these identity relationships clear and explicit. For example, consider a ModuleDefinition for a genetic inverter that includes a FunctionalComponent for an abstract repressor protein. When this ModuleDefinition is instantiated within a "higher level" ModuleDefinition that includes a FunctionalComponent for a LacI protein, the MapsTo object can be used to indicate that the repressor protein in the first ModuleDefinition is LacI in the context of the composite design.

In particular, a MapsTo object provides two pieces of information:

- An identity relationship between two ComponentInstance objects, the first contained by the "lower level" definition of the ComponentInstance or Module that owns the MapsTo, and the second contained by the "higher level" definition that contains the ComponentInstance or Module that owns the MapsTo. The remote property of a MapsTo refers to the first "lower level" ComponentInstance, while the local property refers to the second "higher level" ComponentInstance.
- Instructions on how to interpret local and remote ComponentInstance objects that refer to different ComponentDefinition objects (that is, non-identical objects). These are specified using the refinement property of the MapsTo class.

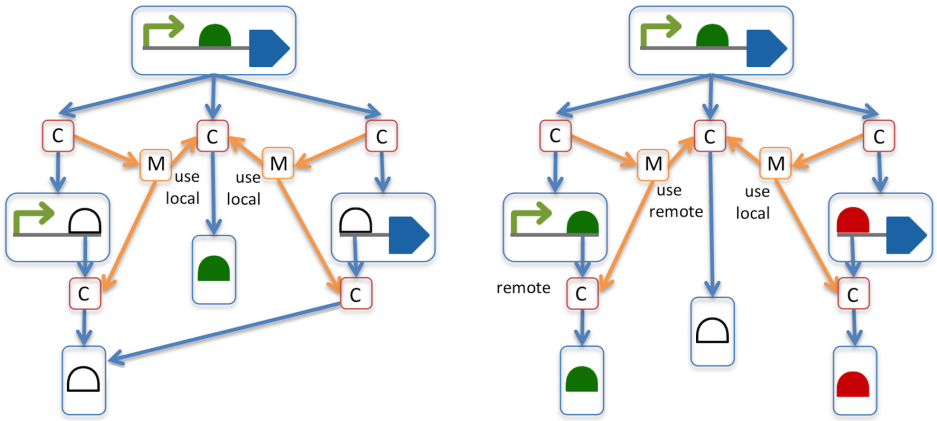


Figure 10: *Linking Component objects using MapsTo entities. Boxes with diagrams represent ComponentDefinition objects, boxes with the C label represent Component objects, and boxes with the M label represent MapsTo objects. In both diagrams, a promoter-RBS ComponentDefinition and a RBS-CDS ComponentDefinition are being composed to form the ComponentDefinition of a complete transcriptional unit. In the left-hand diagram, the two Component objects inside the promoter-RBS ComponentDefinition and RBS-CDS ComponentDefinition objects both refer to an abstract RBS ComponentDefinition that lacks a sequence (white semicircle). Through the use of MapsTo objects with refinement set to useLocal, these "lower level" ComponentDefinition objects are effectively overridden by that of the green RBS in the ComponentDefinition of the complete transcriptional unit. In the right-hand diagram, however, the two "lower level" RBS ComponentDefinition objects do not lack sequences and it is the "higher level" RBS ComponentDefinition that is abstract. In this case, one of the MapsTo objects has a useRemote refinement, resulting in the green RBS ComponentDefinition overriding that of the abstract RBS in the "higher level" ComponentDefinition.*

To illustrate this concept, two examples are provided in Figure 10, in which the ComponentDefinition of a transcriptional unit is specified by composing two "lower level" ComponentDefinition objects. In both examples, the two "lower level" ComponentDefinition objects each contain a RBS Component that is intended to represent the same design entity in the "higher level" ComponentDefinition of the transcriptional unit.

doi:10.2390/biecoll-jib-2015-272

In order to explicitly represent the identity relationships in this example, a new RBS Component needs to be created inside the "higher level" ComponentDefinition. This "higher level" Component then needs to be linked to the equivalent "lower level" Component objects by means of the MapsTo class, using one MapsTo object per link. For example, in order to link the "higher level" RBS Component to the "lower level" RBS Component of the promoter-RBS ComponentDefinition, a MapsTo has to be created on the "higher level" promoter-RBS Component. The local property of this MapsTo then has to refer to the "higher level" RBS Component, while its remote property has to refer to the "lower level" RBS Component. In this way, many "lower level" Component objects can be linked together at the "higher level" using as an equal number of MapsTo objects, each one referring to a different remote Component, but all referring to the same local Component.

The same types of identity relationships can also be declared between FunctionalComponent objects contained by ModuleDefinition objects, or between Component objects and FunctionalComponent objects contained by ComponentDefinition objects and ModuleDefinition objects, respectively. See Section 9 and Section B for additional examples using the MapsTo class.

The `local` property

This REQUIRED property has a data type of URI and is used to refer to the ComponentInstance contained by the "higher level" ComponentDefinition or ModuleDefinition. This local ComponentInstance MUST be contained by the ComponentDefinition or ModuleDefinition that contains the ComponentInstance or Module that owns the MapsTo.

The `remote` property

This REQUIRED property has a data type of URI and is used to refer to the ComponentInstance contained by the "lower level" ComponentDefinition or ModuleDefinition. This remote ComponentInstance MUST be contained by the ComponentDefinition or ModuleDefinition that is the definition of the ComponentInstance or Module that owns the MapsTo. Lastly, the access property of the remote ComponentInstance MUST be set to "public."

The `refinement` property

The refinement property is REQUIRED and has a data type of URI. Each MapsTo object MUST specify the relationship between its local and remote ComponentInstance objects using one of the REQUIRED refinement URIs provided in Table 5.

Refinement URI	Description
http://sbols.org/v2#useRemote	All references to the local ComponentInstance MUST dereference to the remote ComponentInstance instead.
http://sbols.org/v2#useLocal	In the context of the ComponentDefinition or ModuleDefinition that contains the owner of the MapsTo, all references to the remote ComponentInstance MUST dereference to the local ComponentInstance instead.
http://sbols.org/v2#verifyIdentical	The definition properties of the local and remote ComponentInstance objects MUST refer to the same ComponentDefinition.
http://sbols.org/v2#merge	In the context of the ComponentDefinition or ModuleDefinition that contains the owner of the MapsTo, all references to the local ComponentInstance or the remote ComponentInstance MUST dereference to both objects.

Table 5: *REQUIRED URIs for the refinement property.*

Serialization

The serialization of MapsTo MUST have the following form.

```
<sbol:MapsTo rdf:about="...">
```


doi:10.2390/biecoll-jib-2015-272

```
  ... properties inherited from identified ...
  one <sbol:refinement rdf:resource="..."/> element
  one <sbol:remote rdf:resource="..."/> element
  one <sbol:local rdf:resource="..."/> element
</sbol:MapsTo>
```

In the example below, a FunctionalComponent in a "higher level" ModuleDefinition of a genetic toggle switch is linked to a FunctionalComponent in a "lower level" LacI inverter ModuleDefinition. The full example can be found in Section B.2.2.

```
<sbol:MapsTo rdf:about="http://sbolstandard.org/example/toggle_switch/laci_inverter/LacI_mapping">
  <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toggle_switch/laci_inverter/LacI_mapping"/>
  <sbol:displayId>LacI_mapping</sbol:displayId>
  <sbol:refinement rdf:resource="http://sbols.org/v2#useRemote"/>
  <sbol:remote rdf:resource="http://sbolstandard.org/example/laci_inverter/TF"/>
  <sbol:local rdf:resource="http://sbolstandard.org/example/toggle_switch/LacI"/>
</sbol:MapsTo>
```

7.7.4 SequenceAnnotation

The SequenceAnnotation class describes one or more regions of interest on the Sequence objects referred to by its parent ComponentDefinition. In addition, SequenceAnnotation objects can describe the substructure of their parent ComponentDefinition through association with the Component objects contained by this ComponentDefinition.

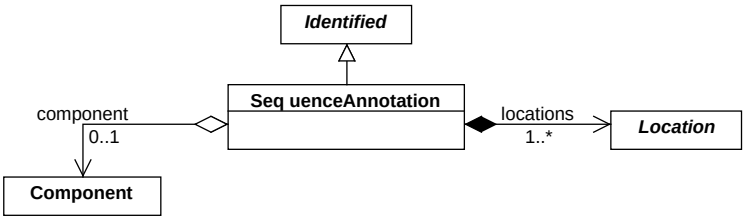


Figure 11: *Diagram of the SequenceAnnotation class and its associated properties.*

The `locations` property

The locations property is a REQUIRED set of one or more Location objects that indicate which elements of a Sequence are described by the SequenceAnnotation.

Allowing multiple Location objects on a single SequenceAnnotation is intended to enable representation of discontinuous regions (for example, a Component encoded across a set of exons with interspersed introns). As such, the Location objects of a single SequenceAnnotation SHOULD NOT specify overlapping regions, since it is not clear what this would mean. There is no such concern with different SequenceAnnotation objects, however, which can freely overlap in Location (for example, specifying overlapping linkers for sequence assembly).

The `component` property

The component property is OPTIONAL and has a data type of URI. This URI MUST refer to a Component that is contained by the same parent ComponentDefinition that contains the SequenceAnnotation. In this way, the properties of the SequenceAnnotation, such as its description and locations, are associated with part of the substructure of its parent ComponentDefinition.

doi:10.2390/biecoll-jib-2015-272

Serialization

The serialization of a `SequenceAnnotation` MUST have the form below. In this template, `A_LOCATION_SUBCLASS` represents one of the `Location` subclasses.

```
<sbol:SequenceAnnotation rdf:about="...">
  ... properties inherited from identified ...
  zero or one <sbol:component rdf:resource="..."/> element
  one or more <sbol:location>
                <sbol:A_LOCATION_SUBCLASS rdf:about="...">...</sbol:A_LOCATION_SUBCLASS>
              </sbol:location> elements
</sbol:SequenceAnnotation>
```

The example below shows the serialization of a `SequenceAnnotation` object. It specifies the region occupied by a `Component` named BBa_F2620.

```
<sbol:SequenceAnnotation rdf:about="http://partsregistry.org/cd/BBa_F2620/anno2">
  <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno2"/>
  <sbol:displayId>anno2</sbol:displayId>
  <sbol:location>
    <sbol:Range rdf:about="http://partsregistry.org/cd/BBa_F2620/anno2/range">
      <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno2/range"/>
      <sbol:displayId>range</sbol:displayId>
      <sbol:start>56</sbol:start>
      <sbol:end>68</sbol:end>
      <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
    </sbol:Range>
  </sbol:location>
  <sbol:component rdf:resource="http://partsregistry.org/cd/BBa_F2620/rbs"/>
</sbol:SequenceAnnotation>
```

7.7.5 Location

The `Location` class is extended by the `Range`, `Cut`, and `GenericLocation` classes.

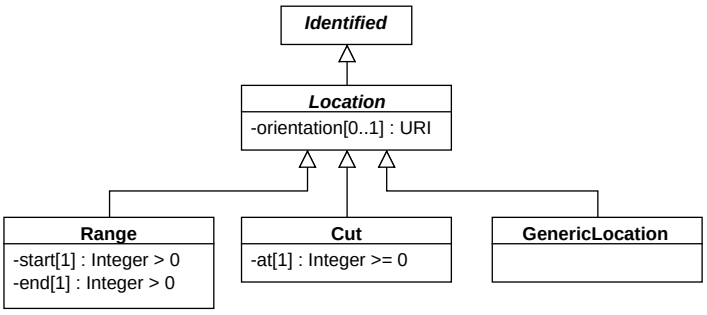


Figure 12: *Diagram of the `Location` class and its associated properties.*

The `orientation` property

The `orientation` property is OPTIONAL and has a data type of `URI`. All subclasses of `Location` share this property, which can be used to indicate how the region specified by the `SequenceAnnotation` and any associated double-stranded `Component` is oriented on the `elements` of a `Sequence` from their parent `ComponentDefinition`. Table 6

doi:10.2390/biecoll-jib-2015-272

provides a list of REQUIRED orientation URIs. If a Location object has an orientation, then it MUST come from Table 6.

Orientation URI	Description
http://sbols.org/v2#inline	The region specified by this Location is on the elements of a Sequence.
http://sbols.org/v2#reverseComplement	The region specified by this Location is on the reverse-complement translation of the elements of a Sequence. The exact nature of this translation depends on the encoding of the Sequence.

Table 6: *REQUIRED URIs for the orientation property*

Range

A Range object specifies a region via discrete, inclusive start and end positions that correspond to indices for characters in the elements String of a Sequence.

Note that the index of the first location is 1, as is typical practice in biology, rather than 0, as is typical practice in computer science.

The start property

The start property specifies the inclusive starting position of the Range. This property is REQUIRED and MUST contain an Integer value greater than zero.

The end property

The end property specifies the inclusive ending position of the Range. This property is REQUIRED and MUST contain an Integer value greater than zero. In addition, this Integer value MUST be greater than or equal to that of the start property.

Serialization

The serialization of a Range MUST have the following form:

```
<sbol:Range rdf:about="...">
  ... properties inherited from identified ...
  one         <sbol:start>...</sbol:start> element
  one         <sbol:end>...</sbol:end> element
  zero or one <sbol:orientation rdf:resource="..."/> element
</sbol:Range>
```

The example below shows the serialization of a Range object. It specifies the region between the inclusive positions 56 and 68, with an orientation of "inline."

```
<sbol:Range rdf:about="http://partsregistry.org/cd/BBa_F2620/anno2/range">
  <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno2/range"/>
  <sbol:displayId>range</sbol:displayId>
  <sbol:start>56</sbol:start>
  <sbol:end>68</sbol:end>
  <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
</sbol:Range>
```

Cut

The Cut class has been introduced to enable the specification of a region between two discrete positions. This specification is accomplished using the at property, which specifies a discrete position that that corresponds to the index of a character in the elements String of a Sequence (except in the case when at is equal to zero—see below).

doi:10.2390/biecoll-jib-2015-272

The `at` property

The `at` property is REQUIRED and MUST contain an `Integer` value greater than or equal to zero. The region specified by the `Cut` is between the position specified by this property and the position that immediately follows it. When the `at` property is equal to zero, the specified region is immediately before the first discrete position or character in the `elements String` of a `Sequence`.

Serialization

The serialization of a `Cut` MUST have the following form:

```
<sbol:Cut rdf:about="...">
  ... properties inherited from identified ...
  one         <sbol:at>...</sbol:at> element
  zero or one <sbol:orientation rdf:resource="..."/> element
</sbol:Cut>
```

The example below shows the serialization of a `Cut` object. It specifies a region in between positions 10 and 11, with an `orientation` of "inline."

```
<sbol:Cut rdf:about="http://partsregistry.org/cd/BBa_J23119/cutat10/cut">
  <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119/cutat10/cut"/>
  <sbol:displayId>cut</sbol:displayId>
  <sbol:at>10</sbol:at>
  <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
</sbol:Cut>
```

GenericLocation

While the `Range` and `Cut` classes are best suited to specifying regions on `Sequence` objects with `IUPAC` encodings, the `GenericLocation` class is included as a starting point for specifying regions on `Sequence` objects with different `encoding` properties and potentially nonlinear structure. This class can also be used to set the `orientation` of a `SequenceAnnotation` and any associated `Component` when their parent `ComponentDefinition` is a partial design that lacks a `Sequence`.

Serialization

The serialization of a `GenericLocation` MUST have the following form:

```
<sbol:GenericLocation rdf:about="...">
  ... properties inherited from identified ...
  zero or one <sbol:orientation rdf:resource="..."/> element
</sbol:GenericLocation>
```

The example below shows the serialization of a `GenericLocation` object with an `orientation` of "reverse complement":

```
<sbol:GenericLocation rdf:about="http://www.partsregistry.org/Part:BBa_F2620/anno5/location">
  <sbol:orientation rdf:resource="http://sbols.org/v2#reverseComplement"/>
</sbol:GenericLocation>
```

7.7.6 SequenceConstraint

The `SequenceConstraint` class can be used to assert restrictions on the relative, sequence-based positions of pairs of `Component` objects contained by the same parent `ComponentDefinition`. The primary purpose of this class is to enable the specification of partially designed `ComponentDefinition` objects, for which the precise positions or orientations of their contained `Component` objects are not yet fully determined. Each `SequenceConstraint` includes the `restriction`, `subject`, and `object` properties.

doi:10.2390/biecoll-jib-2015-272

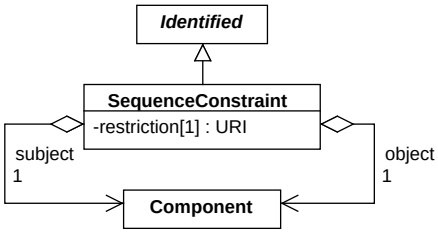


Figure 13: *Diagram of the* `SequenceConstraint` *class and its associated properties.*

The `subject` *property*

The `subject` property is REQUIRED and MUST contain a `URI` that refers to a `Component` contained by the same parent `ComponentDefinition` that contains the `SequenceConstraint`.

The `object` *property*

The `object` property is REQUIRED and MUST contain a `URI` that refers to a `Component` contained by the same parent `ComponentDefinition` that contains the `SequenceConstraint`. This `Component` MUST NOT be the same `Component` that the `SequenceConstraint` refers to via its `subject` property.

The `restriction` *property*

The `restriction` property is REQUIRED and has a data type of `URI`. This property MUST indicate the type of structural restriction on the relative, sequence-based positions or orientations of the `subject` and `object` `Component` objects. The `URI` value of this property SHOULD come from the RECOMMENDED `URI`s in Table 7.

Restriction URI	Description
`http://sbols.org/v2#precedes`	The position of the `subject` `Component` MUST precede that of the `object` `Component`. If each one is associated with a `SequenceAnnotation`, then the `SequenceAnnotation` associated with the `subject` `Component` MUST specify a region that starts before the region specified by the `SequenceAnnotation` associated with the `object` `Component`.
`http://sbols.org/v2#sameOrientationAs`	The `subject` and `object` `Component` objects MUST have the same orientation. If each one is associated with a `SequenceAnnotation`, then the `orientation` `URI`s of the `Location` objects of the first `SequenceAnnotation` MUST be among those of the second `SequenceAnnotation`, and vice versa.
`http://sbols.org/v2#oppositeOrientationAs`	The `subject` and `object` `Component` objects MUST have opposite orientations. If each one is associated with a `SequenceAnnotation`, then the `orientation` `URI`s of the `Location` objects of one `SequenceAnnotation` MUST NOT be among those of the other `SequenceAnnotation`.

Table 7: *RECOMMENDED* `URI`*s for the* `restriction` *property.*

Serialization

The serialization of a `SequenceConstraint` MUST have the following form:

```
<sbol:SequenceConstraint rdf:about="...">
  ... properties inherited from identified ...
  one <sbol:restriction rdf:resource="..."/> element
```


doi:10.2390/biecoll-jib-2015-272

```
  one <sbol:subject rdf:resource="..."/> element
  one <sbol:object rdf:resource="..."/> element
</sbol:SequenceConstraint>
```

The example below shows the serialization of a `SequenceConstraint` belonging to the `ComponentDefinition` of a LacI-repressible promoter. This `SequenceConstraint` has a "precedes" `restriction` that indicates that the `subject` `Component`, which represents the core of the promoter, is positioned before the `object` `Component`, which represents the LacI operator of the promoter.

```
<sbol:SequenceConstraint rdf:about="http://partsregistry.org/cd/BBa_K174004/r1">
  <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_K174004/r1"/>
  <sbol:displayId>r1</sbol:displayId>
  <sbol:restriction rdf:resource="http://sbols.org/v2#precedes"/>
  <sbol:subject rdf:resource="http://partsregistry.org/cd/pspac"/>
  <sbol:object rdf:resource="http://partsregistry.org/cd/LacI_operator"/>
</sbol:SequenceConstraint>
```

7.8 Model

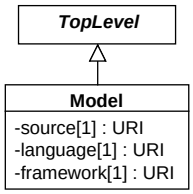


Figure 14: *Diagram of the `Model` class and its associated properties.*

The purpose of the `Model` class is to serve as a placeholder for an external computational model and provide additional meta-data to enable better reasoning about the contents of this model. In this way, there is minimal duplication of standardization efforts and users of SBOL can formalize the function of a `ModuleDefinition` in the language of their choice.

The meta-data provided by the `Model` class include the following properties: the `source` or location of the actual content of the model, the `language` in which the model is implemented, and the model's `framework`.

The `source` property

The `source` property is REQUIRED and MUST contain a `URI` reference to the source file for a model.

The `language` property

The `language` property is REQUIRED and MUST contain a `URI` that specifies the language in which the model is implemented. It is RECOMMENDED that this `URI` refer to a term from the EMBRACE Data and Methods (EDAM) ontology. Table 8 provides a list of terms from this ontology and their `URI`s. If the `language` property of a `Model` is well-described by one these terms, then it MUST contain the `URI` for this term as its value.

The `framework` property

The `framework` property is REQUIRED and MUST contain a `URI` that specifies the framework in which the model is implemented. It is RECOMMENDED this `URI` refer to a term from the modeling framework branch of the SBO when possible. A few suggested modeling frameworks and their corresponding `URI`s are shown in Table 9. If the

doi:10.2390/biecoll-jib-2015-272

Model Language	URI for EDAM Term
SBML	http://identifiers.org/edam/format_2585
CellML	http://identifiers.org/edam/format_3240
BioPAX	http://identifiers.org/edam/format_3156

Table 8: *Terms from the EDAM ontology to specify the* `language` *property of a* `Model`*.*

`framework` property of a `Model` is well-described by one these terms, then it MUST contain the `URI` for this term as its value.

Framework	URI for SBO Term
Continuous	http://identifiers.org/biomodels.sbo/SBO:0000062
Discrete	http://identifiers.org/biomodels.sbo/SBO:0000063

Table 9: *SBO terms to specify the* `framework` *property of a* `Model`*.*

Serialization

The serialization of a `Model` MUST have the following form:

```
<sbol:Model rdf:about="http://www.sbolstandard.org/examples/toggleswitch">
  ... properties inherited from identified ...
  one <sbol:source rdf:resource="..."/> element
  one <sbol:language rdf:resource="..."/> element
  one <sbol:framework rdf:resource="..."/> element
</sbol:Model>
```

The example below shows the serialization of a `Model` object that refers to a quantitative model of a genetic toggle switch. The model is implemented in the SBML `language` and adheres to a continuous modeling `framework`. Lastly, the model can be retrieved from a model repository via its `source` `URI`, which is a URL.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
    w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:Model rdf:about="http://www.sbolstandard.org/examples/pIKE_Toggle_1">
    <sbol:persistentIdentity rdf:resource="http://www.sbolstandard.org/examples/pIKE_Toggle_1"/>
    <sbol:displayId>pIKE_Toggle_1</sbol:displayId>
    <dcterms:title>pIKE_Toggle_1 toggle switch</dcterms:title>
    <sbol:source rdf:resource="http://virtualparts.org/part/pIKE_Toggle_1"/>
    <sbol:language rdf:resource="http://identifiers.org/edam/format_2585"/>
    <sbol:framework rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000062"/>
  </sbol:Model>
</rdf:RDF>
```

7.9 ModuleDefinition

The `ModuleDefinition` class represents a grouping of structural and functional entities in a biological design. The primary usage of this class is to assert the molecular interactions and abstract function of its child entities.

As shown in Figure 15, these child entities are aggregated via the `functionalComponents` `modules` properties, while representation of their abstract function is accomplished via the `roles` property. More detailed descriptions of the function of a `ModuleDefinition` are provided by its `interactions` and `models` properties. Lastly, since `ModuleDefinition` objects can be more abstract and represent entities of engineering design rather than biology, they can have designated "inputs" and "outputs" expressed by the `direction` properties on its `FunctionalComponent`

doi:10.2390/biecoll-jib-2015-272

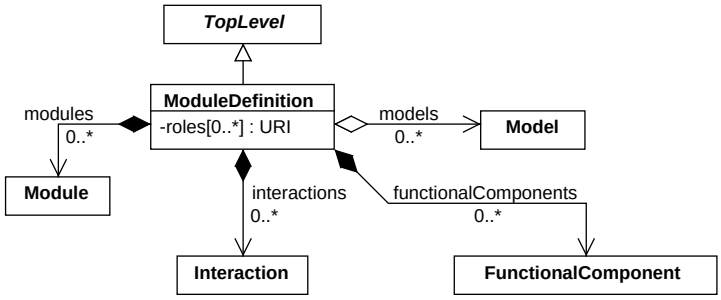


Figure 15: *Diagram of the* `ModuleDefinition` *class and its associated properties.*

objects.

The `roles` *property*

The `roles` property is an OPTIONAL set of `URI`s that clarifies the intended function of a `ModuleDefinition`.

These `URI`s might identify descriptive biological roles, such as "metabolic pathway" and "signaling cascade," but they can also identify identify "logical" roles, such as "inverter" or "AND gate", or other abstract roles for describing the function of design. Interpretation of the meaning of such roles currently depends on the software tools that read and write them.

The `modules` *property*

The `modules` property is OPTIONAL and MAY specify a set of `Module` objects contained by the `ModuleDefinition`. Note that the set of relations between `Module` and `ModuleDefinition` objects is strictly acyclic.

While the `ModuleDefinition` class is analogous to a specification sheet for a system of interacting biological elements, the `Module` class represents the occurrence of a particular subsystem within the system. Hence, this class allows a system design to include multiple instances of a subsystem, all defined by reference to the same `ModuleDefinition`. For example, consider the `ModuleDefinition` for a network of two-input repressor devices in which the particular repressors have not been chosen yet. This `ModuleDefinition` could contain multiple `Module` objects that refer to the same `ModuleDefinition` of an abstract two-input repressor device.

The `functionalComponents` *property*

The `functionalComponents` property is OPTIONAL and MAY specify a set of `FunctionalComponent` objects contained by the `ModuleDefinition`.

Just as a `Module` represents an instance of a subsystem in the overall system represented by a `ModuleDefinition`, a `FunctionalComponent` represents an instance of a structural entity (represented by a `ComponentDefinition`) in the system. This concept allows a `ModuleDefinition` to assert different interactions for separate copies of the same structural entity if needed. For example, a `ModuleDefinition` might contain multiple `FunctionalComponent` objects that refer to the same promoter `ComponentDefinition`, but assert different interactions for these promoter copies based on their separate positions in another `ComponentDefinition` that represents the structure of the entire system.

The `interactions` *property*

The `interactions` property is OPTIONAL and MAY specify a set of `Interaction` objects within the `ModuleDefinition`.

doi:10.2390/biecoll-jib-2015-272

The Interaction class provides an abstract, machine-readable representation of entity behavior within a ModuleDefinition (whereas a more detailed model of the system might not be suited to machine reasoning, depending on its implementation). Each Interaction contains Participation objects that indicate the roles of the FunctionalComponent objects involved in the Interaction.

The `models` property

The models property is OPTIONAL and MAY specify a set of URI references to Model objects.

Model objects are placeholders that link ModuleDefinition objects to computational models of any format. A ModuleDefinition object can link to more than one Model since each might encode system behavior in a different way or at a different level of detail.

Serialization

The serialization of ModuleDefinition has the following form:

```
<sbol:ModuleDefinition rdf:about="...">
  ... properties inherited from identified ...
  zero or more  <sbol:role rdf:resource="..."/> elements
  zero or more  <sbol:model rdf:resource="..."/> elements
  zero or more <sbol:functionalComponent>
                 <sbol:FunctionalComponent rdf:about="...">...</sbol:FunctionalComponent >
               </sbol:functionalComponent> elements
  zero or more <sbol:module>
                 <sbol:Module rdf:about="...">...</sbol:Module>
               </sbol:module> elements
  zero or more <sbol:interaction>
                 <sbol:Interaction rdf:about="...">...</sbol:Interaction>
               </sbol:interaction> elements
</sbol:ModuleDefinition>
```

The example below shows a simple ModuleDefinition containing two components, a FunctionalComponent for a DNA sequence encoding constitutive expression of GFP and another for the GFP protein expressed from this sequence, plus an interaction describing that relation.

```
<sbol:ModuleDefinition rdf:about="http://sbolstandard.org/example/md/GFP_expression">
  <sbol:functionalComponent>
    <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/md/GFP_expression/GFP_protein">
      <sbol:definition rdf:resource="http://sbolstandard.org/example/GFP"/>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:direction rdf:resource="http://sbols.org/v2#output"/>
    </sbol:FunctionalComponent>
  </sbol:functionalComponent>
  <sbol:functionalComponent>
    <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/md/GFP_expression/Constitutive_GFP">
      <sbol:definition rdf:resource="http://sbolstandard.org/example/GFP_generator"/>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:direction rdf:resource="http://sbols.org/v2#none"/>
    </sbol:FunctionalComponent>
  </sbol:functionalComponent>
  <sbol:interaction>
    <sbol:Interaction rdf:about="http://sbolstandard.org/example/md/GFP_expression/express_GFP">
      ...
    </sbol:Interaction>
  </sbol:interaction>
</sbol:ModuleDefinition>
```

7.9.1 Module

The Module class represents the usage or occurrence of a ModuleDefinition within a larger design (that is, another ModuleDefinition).

doi:10.2390/biecoll-jib-2015-272

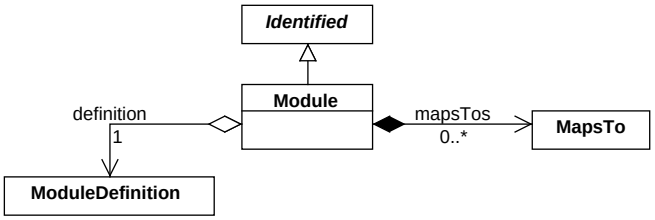


Figure 16: *Diagram of the* `Module` *class and its associated properties.*

The `definition` *property*

The `definition` property is a REQUIRED `URI` that refers to the `ModuleDefinition` for the `Module`.

The `definition` property MUST NOT refer to the same `ModuleDefinition` as that which contains the `Module`. Furthermore, `Module` objects MUST NOT form a cyclical chain of references via their `definition` properties and the `ModuleDefinition` objects that contain them. For example, consider the `Module` objects A and B and the `ModuleDefinition` objects X and Y. The reference chain "X contains A, A is defined by Y, Y contains B, and B is defined by X" is cyclical.

The `mapsTo` *property*

The `mapsTos` property is an OPTIONAL set of `MapsTo` objects that refer to and link `ComponentInstance` objects together within the heterarchy of `Module`, `ModuleDefinition`, `ComponentInstance`, and `ComponentDefinition` objects.

Section 7.7.3 contains a detailed description of the `MapsTo` class.

Serialization

The serialization of `Module`s has the following form.

```
<sbol:Module rdf:about="...">
  ... properties inherited from identified ...
  one          <sbol:definition rdf:resource="..."/> element
  zero or more <sbol:mapsTo>
                 <sbol:MapsTo rdf:about="...">...</sbol:MapsTo>
               </sbol:mapsTo> element
</sbol:Module>
```

The example below specifies a TetR inverter that is being used as a part of a genetic toggle switch:

```
<sbol:Module rdf:about="http://sbolstandard.org/example/toggle_switch/tetr_inverter">
  <sbol:definition rdf:resource="http://sbolstandard.org/example/tetr_inverter"/>
  ...
</sbol:Module>
```

7.9.2 FunctionalComponent

A `FunctionalComponent` is an instance of a `ComponentDefinition` being used as part of a `ModuleDefinition`. The `ModuleDefinition` describes how the that describes how the `FunctionalComponent` interacts with others and summarizes their aggregate function.

The `FunctionalComponent` class inherits from the `ComponentInstance` class and therefore has the `definition`, `access`, and `mapsTos` properties. In addition, it has a `direction` property that specifies whether it serves as an

doi:10.2390/biecoll-jib-2015-272

input, output, both, or neither with regards to the ModuleDefinition that contains it.

The direction property

Each FunctionalComponent MUST specify via the direction property whether it serves as an input, output, both, or neither for its parent ModuleDefinition object. The value for this property MUST be one of the URIs given in Table 10.

Direction URI	Description
http://sbols.org/v2#in	Indicates that the FunctionalComponent is an input.
http://sbols.org/v2#out	Indicates that the FunctionalComponent is an output.
http://sbols.org/v2#inout	Indicates that the FunctionalComponent is both an input and output
http://sbols.org/v2#none	Indicates that the FunctionalComponent is neither an input or output.

Table 10: *REQUIRED URIs for the direction property.*

The direction property is a means to encode how a designer thinks about the "purpose" of a connection in a system. In SBOL, such a connection is represented with a FunctionalComponent, and a system is represented as with a ModuleDefinition. For example, consider a system that has been designed to sense the concentration of the cell-to-cell signaling molecule $3OC_6HSL$ and report it via the concentration of another gene product. In this system, the concentration of $3OC_6HSL$ is being sensed by the system, so the FunctionalComponent for $3OC_6HSL$ would have a direction of "input." In turn, the concentration of the reporter gene product is intended to be read/consumed by other biological systems, so the FunctionalComponent for this product would have a direction of "output." The CDS encoding the product, however, is not intended to directly transfer information into or out of the ModuleDefinition for the system, so its FunctionalComponent would have a direction of "neither."

Serialization

The serialization of a FunctionalComponent has the following form.

```
<sbol:FunctionalComponent rdf:about="...">
  ... properties inherited from identified ...
  one          <sbol:definition rdf:resource="..."/> element
  one          <sbol:access rdf:resource="..."/> element
  one          <sbol:direction rdf:resource="..."/> element
  zero or more <sbol:mapsTo rdf:resource="..."/> elements
</sbol:FunctionalComponent>
```

In the example below, the functional component is defined as a public input/output. The component refers to the `Part:BBa_R0010` promoter from the iGEM Parts Registry.

```
<sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/laci_inverter/promoter">
  <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_R0010"/>
  <sbol:access rdf:resource="http://sbols.org/v2#public"/>
  <sbol:direction rdf:resource="http://sbols.org/v2#inout"/>
</sbol:FunctionalComponent>
```

7.9.3 Interaction

The Interaction class provides more detailed description of how the FunctionalComponent objects of a ModuleDefinition are intended to work together. For example, this class can be used to represent different forms of genetic regulation (e.g., transcriptional activation or repression), processes from the central dogma of biology (e.g. transcription and translation), and other basic molecular interactions (e.g., non-covalent binding or enzymatic phosphorylation). Each Interaction includes a types property that refers to descriptive ontology terms and a participations property that describes which FunctionalComponent objects participate in the Interaction.

doi:10.2390/biecoll-jib-2015-272

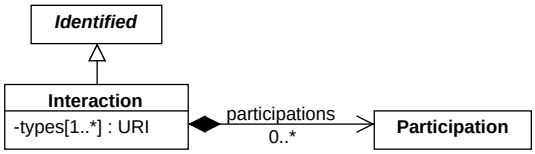


Figure 17: *Diagram of the* `Interaction` *class and its associated properties.*

The `types` *property*

The `types` property is a REQUIRED set of `URI`s that describes the behavior represented by an `Interaction`.

The `types` property MUST contain one or more `URI`s that MUST identify terms from appropriate ontologies. It is RECOMMENDED that at least one of the `URI`s contained by the `types` property refer to a term from the occurring entity branch of the Systems Biology Ontology (SBO). (See `http://www.ebi.ac.uk/sbo/main/`) Table 11 provides a list of possible SBO terms for the `types` property and their corresponding `URI`s.

Interaction Type	URI for SBO Term
Inhibition	`http://identifiers.org/biomodels.sbo/SBO:0000169`
Stimulation	`http://identifiers.org/biomodels.sbo/SBO:0000170`
Genetic Production	`http://identifiers.org/biomodels.sbo/SBO:0000589`
Non-Covalent Binding	`http://identifiers.org/biomodels.sbo/SBO:0000177`

Table 11: *SBO terms to specify the* `types` *property of an* `Interaction`*.*

If an `Interaction` is well described by one of the terms from Table 11, then its `types` property MUST contain the `URI` that identifies this term. Lastly, if the `types` property of an `Interaction` contains multiple `URI`s, then they MUST identify non-conflicting terms. For example, the SBO terms “stimulation” and “inhibition” would conflict.

The `participations` *property*

The `participations` property is an OPTIONAL and MAY contain a set of `Participation` objects, each of which identifies the `roles` that its referenced `FunctionalComponent` plays in the `Interaction`.

Even though an `Interaction` generally contains at least one `Participation`, the case of zero `Participation` objects is allowed because it is plausible that a designer might want to specify that an `Interaction` will exist, even if its `participant`s have not yet been determined.

Serialization

The serialization of an `Interaction` has the following form.

```
<sbol:Interaction rdf:about="...">
  ... properties inherited from identified ...
  one or more  <sbol:type rdf:resource="..."/> elements
  zero or more <sbol:participation>
                 <sbol:Participation rdf:about="...">...</sbol:Participation>
               </sbol:participation> elements
</sbol:Interaction>
```

The example below shows an `Interaction` representing an inhibition relationship (SBO:0000169) between a repressor (SBO:0000020, full `Participation` details shown) and a promoter:

doi:10.2390/biecoll-jib-2015-272

```
<sbol:Interaction rdf:about="http://sbolstandard.org/example/laci_inverter/LacI_pLacI">
  <sbol:type rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000169"/>
  <sbol:participation>
    <sbol:Participation rdf:about="http://sbolstandard.org/example/laci_inverter/LacI_pLacI/P03023">
      <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000020"/>
      <sbol:participant rdf:resource="http://sbolstandard.org/example/laci_inverter/TF"/>
    </sbol:Participation>
  </sbol:participation>
  <sbol:participation>
    <sbol:Participation rdf:about="...">
      ...
    </sbol:Participation>
  </sbol:participation>
</sbol:Interaction>
```

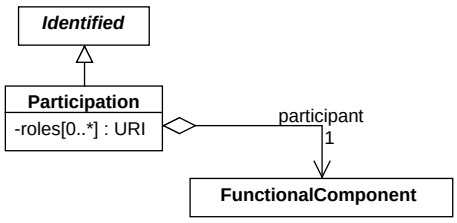


Figure 18: *Diagram of the* `Participation` *class and its associated properties.*

7.9.4 Participation

Each `Participation` represents how a particular `FunctionalComponent` behaves in its parent `Interaction`.

The `roles` *property*

The `roles` property is an OPTIONAL set of `URI`s that describes the behavior of a `Participation` (and by extension its referenced `FunctionalComponent`) in the context of its parent `Interaction`.

The `roles` property MAY contain one or more `URI`s that MUST identify terms from appropriate ontologies. It is RECOMMENDED that at least one of the `URI`s contained by the `types` property refer to a term from the participant role branch of the SBO. Table 12 provides a list of possible SBO terms for the `roles` property and their corresponding `URI`s.

Participation Role	URI for SBO Term
Inhibitor	`http://identifiers.org/biomodels.sbo/SBO:0000020`
Stimulator	`http://identifiers.org/biomodels.sbo/SBO:0000459`
Reactant	`http://identifiers.org/biomodels.sbo/SBO:0000010`
Product	`http://identifiers.org/biomodels.sbo/SBO:0000011`
Ligand	`http://identifiers.org/biomodels.sbo/SBO:0000280`
Non-Covalent Complex	`http://identifiers.org/biomodels.sbo/SBO:0000253`

Table 12: *SBO terms to specify the* `roles` *property of a* `Participation`.

If a `Participation` is well described by one of the terms from Table 12, then its `roles` property MUST contain the `URI` that identifies this term. Lastly, if the `roles` property of a `Participation` contains multiple `URI`s, then they

doi:10.2390/biecoll-jib-2015-272

MUST identify non-conflicting terms. For example, the SBO terms “stimulator” and “inhibitor” would conflict.

The `participant` property

The `participant` property MUST specify precisely one `FunctionalComponent` object that plays the designated role in its parent `Interaction` object.

Serialization

The serialization of `Participation` objects has the following form.

```
<sbol:Participation rdf:about="...">
  ... properties inherited from identified ...
  zero or more <sbol:role rdf:resource="..."/> elements
  one          <sbol:participant rdf:resource="..."/> element
</sbol:Participation>
```

In the example below, the role of participating `FunctionalComponent` is defined to be `inhibitor`, using the `SBO:0000020` term. This component is specified using the `participant` property of the `Participation` entity.

```
<sbol:Participation rdf:about="http://sbolstandard.org/example/laci_inverter/LacI_pLacI/P03023">
  <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000020"/>
  <sbol:participant rdf:resource="http://sbolstandard.org/example/laci_inverter/TF"/>
</sbol:Participation>
```

7.10 Collection

The `Collection` class is a class that groups together a set of `TopLevel` objects that have something in common. Some examples of `Collection` objects:

- Results of a query to find all `ComponentDefinition` objects in a repository that function as promoters.
- A set of `ModuleDefinition` objects representing a library of genetic logic gates.
- A `ModuleDefinition` for a complex design, and all of the `ModuleDefinition`, `ComponentDefinition`, `Sequence`, and `Model` objects used to provide its full specification.

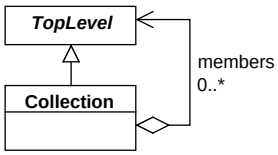


Figure 19: *Diagram of the `Collection` class and its associated properties.*

The `members` property

The `members` property of a `Collection` is OPTIONAL and MAY contain a set of `URI` references to zero or more `TopLevel` objects.

doi:10.2390/biecoll-jib-2015-272

Serialization

The serialization of a `Collection` has the following form:

```
<sbol:Collection rdf:about="...">
  ... properties inherited from identified ...
  zero or more <sbol:member rdf:resource="..."/> element
</sbol:Collection>
```

The example below shows the serialization of a `Collection` object grouping together a library of constitutive promoters.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
     w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:Collection rdf:about="http://parts.igem.org/Promoters/Catalog/Anderson">
    <sbol:persistentIdentity rdf:resource="http://parts.igem.org/Promoters/Catalog/Anderson"/>
    <sbol:displayId>Anderson</sbol:displayId>
    <dcterms:title>Anderson promoters</dcterms:title>
    <dcterms:description>The Anderson promoter collection</dcterms:description>
    <sbol:member rdf:resource="http://partsregistry.org/Part:BBa_J23119"/>
    ...
    <sbol:member rdf:resource="http://partsregistry.org/Part:BBa_J23118"/>
  </sbol:Collection>
</rdf:RDF>
```

7.11 Annotation and Extension of SBOL

SBOL does not currently represent all types of biological design data, since many of these data types (e.g., biological context and design performance metrics) lack a clear consensus on their proper representation. In addition, some types of biological data are not directly relevant to design and are therefore outside of the scope of SBOL.

To enable representation of these data, SBOL allows developers to embed custom data within SBOL objects and documents, such that these data can be exchanged without being damaged or lost. This annotation and extension mechanism is designed to enable new types of data to be easily incorporated into the SBOL standard once there is community consensus on their proper representation.

Several methods are supported for connecting the SBOL data model with other types of application-specific data:

- Custom data can be added to an SBOL object by annotating that object with non-conflicting properties. These properties could contain `literal` data types such as `String`s or `URI`s that require a resolution mechanism to obtain external data. An example is annotating a `ComponentDefinition` with a property that contains a `String` description and `URI` for the parts registry from which its source data was originally imported.
- Custom data in the form of independent objects can be added to an SBOL document by creating `GenericTopLevel` objects and annotating them as described above. An example is a `GenericTopLevel` object that is annotated such that it represents a data sheet that describes the performance of a `ModuleDefinition` in a particular context.
- Finally, just as custom objects can be embedded in an SBOL document, external documents can embed or refer to SBOL objects. Support for this last case is not explicitly provided in this specification. Rather, this case depends on the external non-SBOL system managing its relationship to SBOL and data serialized in RDF/XML, and is included here for completeness.

7.11.1 Annotating SBOL objects

Each `Identified` object MAY contain any number of `Annotation` objects that store data in the form of name/value property pairs. The `qName` is REQUIRED and MUST contain a `QName`, which is composed of a namespace, an

doi:10.2390/biecoll-jib-2015-272

OPTIONAL prefix, and a local name. The qName property MUST be stored in the data model to allow for proper serialization as described below. The value property is also REQUIRED and MUST contain a literal (i.e., a String, Integer, Double, Boolean), URI, or NestedAnnotations object. A NestedAnnotations object MUST contain nestedQName and nestedURI properties, and it MAY contain an annotations property that contains zero or more Annotation objects.

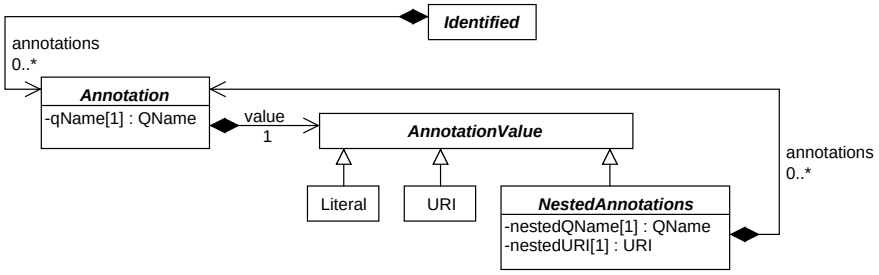


Figure 20: *Diagram of the Annotation class and its association with Identified and AnnotationValue objects, which is used for annotating SBOL entities with application specific data.*

Serialization

The serialization of an Annotation has the following form:

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:sbol="http://sbols.org/v2#"
  xmlns:prefix1="NAMESPACE_1"
  xmlns:prefix2="NAMESPACE_2"
  xmlns:nestedObjectPrefix="A_NESTED_OBJECT_NAMESPACE"
  ...
 >
<sbol:A_TOPLEVELOBJECT rdf:about="...">
                ...
  zero or more <prefix1:LOCAL_NAME_1>A_LITERAL</prefix1:LOCAL_NAME_1> elements
  zero or more <prefix1:LOCAL_NAME_2 rdf:resource=URI/> elements
  zero or more <prefix2:LOCAL_NAME_3>
                 <nestedObjectPrefix:NESTED_LOCAL_NAME rdf:about="...">
                   ...
                 </nestedObjectPrefix:NESTED_LOCAL_NAME>
               </prefix2:LOCAL_NAME_3> elements
</sbol:TOPLEVELOBJECT>
```

The qName property specifies a namespace, prefix, and local part/name. The use of such qualified names is described in detail by the W3C (http://www.w3.org/TR/1999/REC-xml-names-19990114/#ns-using). Essentially, the "xmlns" property defines the prefix String to use as an alias for the namespace. The prefix can be any String. Its use is OPTIONAL, since it simply replaces the full namespace, thereby making the serialization easier for a human to read.

The first form of Annotation shown above is for an Annotation that contains a literal as its value. The second form is for an Annotation that contains a URI as its value. Finally, the third form is for an Annotation that contains a NestedAnnotations object as its value. In the last case, the nestedQName property specifies the nested namespace, nested prefix, and nested local part/name, while the nestedURI property species the URI for the

doi:10.2390/biecoll-jib-2015-272

NestedAnnotations object.

The example below shows how the serialization for a promoter ComponentDefinition can be annotated with custom data. Annotations are added containing the relevant information from the iGEM Parts Registry. Each property serialization of an Annotation is qualified with the `http://www.partsregistry.org/` namespace, which is prefixed using `pr`. The first Annotation is named `pr:group`. It specifies the iGEM group that has designed the promoter and has a String value. The second Annotation is named `pr:experience`. It contains a URI value that is serialized as an RDF resource and can be resolved to the information Web page on the Parts Registry for the promoter. Finally, the third Annotation is named `pr:information`. It contains a NestedAnnotations object that is serialized as shown and includes information about the regulatory details of the promoter using Annotations that correspond to Parts Registry categories.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:pr="http://partsregistry.org" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.
     org/dc/terms/" xmlns:prov="http://www.w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <pr:group>iGEM2006_Berkeley</pr:group>
    <pr:experience rdf:resource="http://parts.igem.org/cgi/partsdb/part_info.cgi?part_name=BBa_J23119"/>
    <pr:information>
      <pr:Information rdf:about="http://parts.igem.org/cgi/partsdb/part_info.cgi?part_name=BBa_J23119">
        <pr:sigmafactor>//rnap/prokaryote/ecoli/sigma70</pr:sigmafactor>
        <pr:regulation>//regulation/constitutive</pr:regulation>
      </pr:Information>
    </pr:information>
    <dcterms:title>J23119</dcterms:title>
    <dcterms:description>Constitutive promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
  </sbol:ComponentDefinition>
</rdf:RDF>
```

7.11.2 GenericTopLevel

Custom data can also be embedded at the top level of an SBOL document. The GenericTopLevel class is used to represent top-level entities whose purpose is to contain a set of annotations that are independent of any other class of SBOL object. Entities that have independent existence and are not recognized by the SBOL standard are deserialized to GenericTopLevel objects. These GenericTopLevel objects can be safely used by tools to exchange non-SBOL data.

As with other TopLevel objects, GenericTopLevel objects MAY include the properties displayId, name, description, etc. The type of data annotating a GenericTopLevel object is indicated using the REQUIRED rdfType property, which MUST contain a QName. As before with the qName property, the rdfType property is used to set the namespace, prefix, and local part/name during serialization.

Serialization

The serialization of the GenericTopLevel class has the following form, where the prefix, namespace, and local part/name are defined by the rdfType property:

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:sbol="http://sbols.org/v2#"
  xmlns:applicationPrefix="APPLICATION_NAMESPACE"
  ...
 >
```


doi:10.2390/biecoll-jib-2015-272

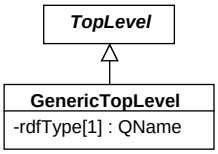


Figure 21: *Diagram of the* `GenericTopLevel` *class and its associated properties, which is used for embedding externally defined entities into SBOL documents.*

```
<applicationPrefix:APPLICATION_OBJECT_NAME rdf:about="...">
  ... properties inherited from identified ...
  ... any non-conflicting application-specific properties ...
</applicationPrefix:APPLICATION_OBJECT_NAME>
```

The example below shows how a datasheet object can be added to an SBOL document using the `GenericTopLevel` class. The J23119 promoter `ComponentDefinition` is annotated with the `myapp:datasheet` property that contains the `URI` of a `TopLevel` Datasheet object. The Datasheet object is further annotated with the transcription rate and `URI` for the actual characterization data using the `myapp:transcriptionRate` and `myapp:characterizationData` properties, respectively. As specified by their `rdfType` and `qName` properties, the `TopLevel` Datasheet object and all `Annotation` objects in this example are serialized with the custom `http://www.myapp.org/` namespace and `myapp` prefix.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:myapp="http://www.myapp.org/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.
      org/dc/terms/" xmlns:prov="http://www.w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/cd/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/cd/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://www.partsregistry.org/Part:BBa_J23119"/>
    <myapp:datasheet rdf:resource="http://www.partsregistry.org/gen/datasheet1"/>
    <dcterms:title>J23119</dcterms:title>
    <dcterms:description>Constitutive promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
  </sbol:ComponentDefinition>
  <myapp:Datasheet rdf:about="http://www.partsregistry.org/gen/datasheet1">
    <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/gen/datasheet1"/>
    <sbol:displayId>datasheet1</sbol:displayId>
    <myapp:characterizationData rdf:resource="http://www.myapp.org/measurement/1"/>
    <myapp:transcriptionRate>1</myapp:transcriptionRate>
    <dcterms:title>Datasheet 1</dcterms:title>
  </myapp:Datasheet>
</rdf:RDF>
```


doi:10.2390/biecoll-jib-2015-272

8 Mapping Between SBOL 1.1 and SBOL 2.0

Figure 22 depicts the mapping of SBOL 1.1 classes to SBOL 2.0 classes, indicating corresponding classes/properties by color. The SBOL 2.0 `Model` and `ModuleDefinition` classes have no SBOL 1.1 equivalent, and thus are not shown. In particular:

- SBOL 1.1 `Collection` objects containing `DnaComponent` objects map to SBOL 2.0 `Collection` objects that contain `ComponentDefinition` objects with DNA `types` properties.
- SBOL 1.1 `DnaComponent` objects maps to SBOL 2.0 `ComponentDefinition` objects with DNA `types` properties.
- SBOL 1.1 `DnaSequence` objects maps to an SBOL 2.0 `Sequence` objects with `IUPAC DNA` `encoding` properties.
- SBOL 1.1 `SequenceAnnotation` objects with `bioStart` and `bioEnd` properties map to SBOL 2.0 `SequenceAnnotation` objects that contain `Range` objects.
- SBOL 1.1 `SequenceAnnotation` objects that lack `bioStart` and `bioEnd` properties map to an SBOL 2.0 `SequenceAnnotation` objects that contain `GenericLocation` objects.
- Each SBOL 1.1 `SequenceAnnotation` also maps to an SBOL 2.0 `Component`, which represents the instantiation or usage of the appropriate `ComponentDefinition`.
- Each SBOL 1.1 `precedes` property maps to an SBOL 2.0 `SequenceConstraint` that specifies a precedes `restriction` property.

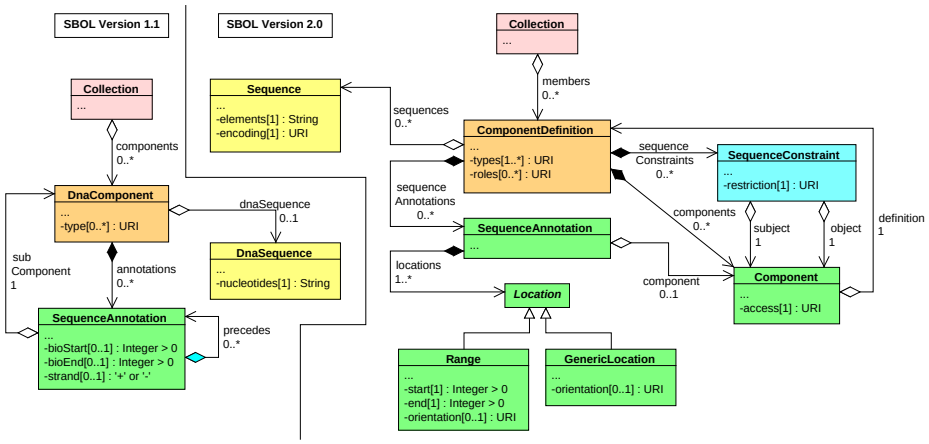


Figure 22: *The mapping from the SBOL 1.1 data model to the SBOL 2.0 data model, indicating corresponding classes/properties by color.*

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

9 Data Model Examples

This section illustrates how to use the SBOL data model by specifying the design of a LacI/TetR toggle switch similar to those constructed in Gardner et al. (2000). This design is visualized conceptually in Figure 23 and in detail in Figure 24.

Conceptually, the toggle switch is constructed from two mutually repressing genes. With repressors LacI and TetR, this results in a bi-stable system that will tend to settle into a state where precisely one of the two repressors is strongly expressed, repressing the other. Each of these repressors can have its activity disrupted by a small molecule (IPTG for LacI, aTc for TetR), which enables the system to be "toggled" from one state to the other by dosing it with the appropriate small molecule.

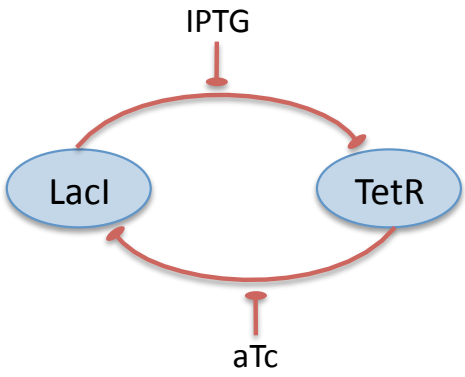


Figure 23: *Conceptual diagram of LacI/TetR toggle switch: the LacI and TetR transcription factors are arranged to mutually repress each other's expression, creating a bi-stable system. Transition between the two states is triggered by the small-molecule signals aTc (which disrupts TetR repression) and IPTG (which disrupts LacI repression).*

The LacI/TetR toggle switch is modeled in SBOL as two parallel hierarchies of structure and function. The structural hierarchy of the toggle switch is represented using `ComponentDefinition`s:

- The base elements of the hierarchy are DNA components, transcription factor proteins, and small molecules. As an example, Figure 25 is a UML diagram of the `ComponentDefinition` objects that represent these elements.
- Base elements are composed to form more complex structures at the top of the hierarchy, including genes and non-covalent complexes between transcription factor proteins and small molecules. As an example, Figure 26 is a UML diagram of the composite `ComponentDefinition` objects that represent the TetR gene and IPTG-LacI complex.

The functional hierarchy of the toggle switch is represented using `ModuleDefinition`s:

- The base elements of the hierarchy are LacI-dependent repression of TetR expression (the LacI inverter) and TetR-dependent repression of LacI (the TetR inverter). As an example, Figure 27 is a UML diagram of the `ModuleDefinition` that represents the LacI inverter.
- Base elements are composed to form the toggle switch at the top of the hierarchy. As an example, Figure 28 is a UML diagram of the `ModuleDefinition` that represents the toggle switch.

doi:10.2390/biecoll-jib-2015-272

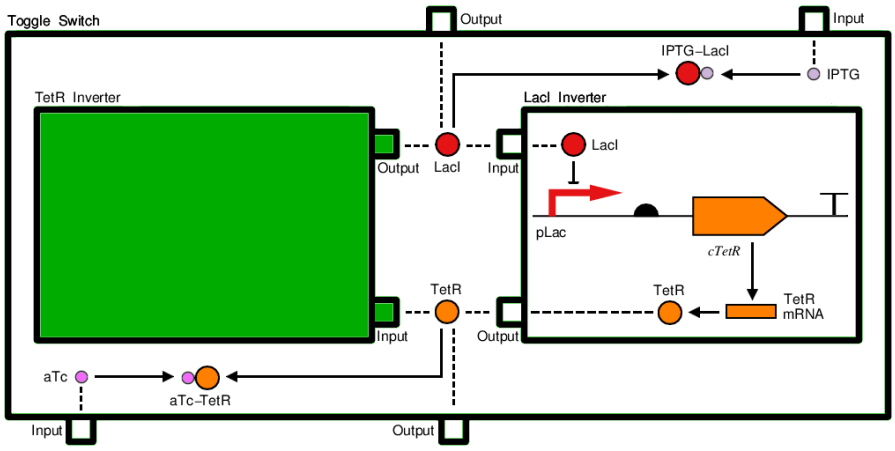


Figure 24: *Design of a LacI/TetR toggle switch. This design is composed of two inverter sub-designs, each containing a single gene. These genes mutually repress each other's expression via their encoded protein transcription factors, LacI and TetR. Furthermore, both LacI and TetR are bound by specific small molecules that sequester them and prevent them from acting as repressors. In this design, arrows represent different molecular interactions, including the repression of pLac via LacI, the non-covalent binding of IPTG to LacI, the transcription of TetR mRNA, and the translation of TetR. Dashed lines serve to map between transcription factors in the inverter sub-designs and those in the overall toggle switch design.*

doi:10.2390/biecoll-jib-2015-272

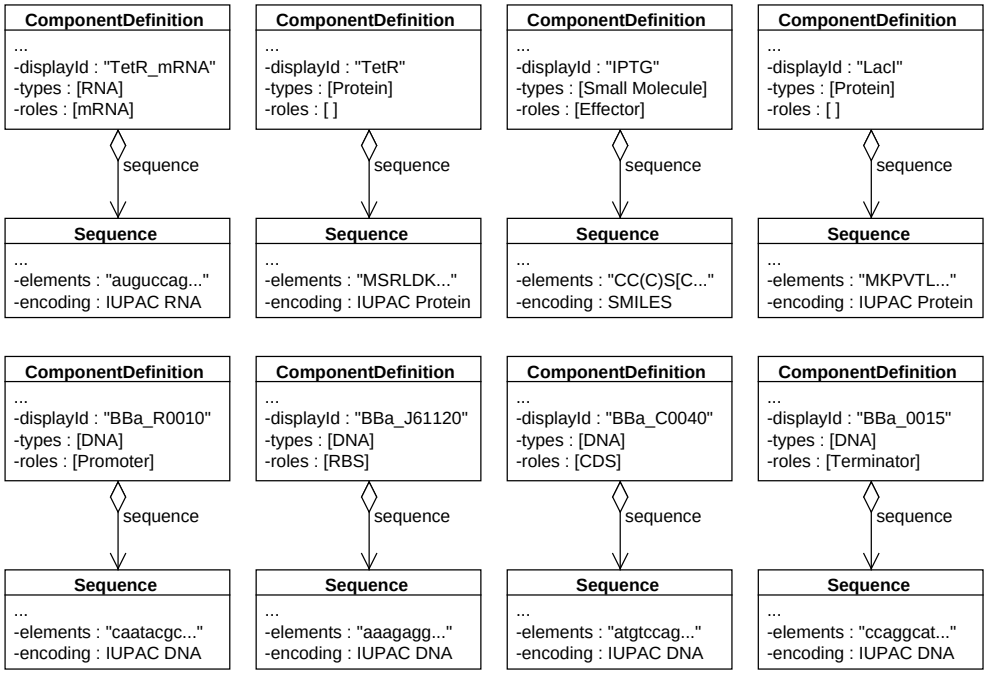


Figure 25: *`ComponentDefinition` objects for the LacI inverter. These include `ComponentDefinition` objects based on DNA parts from the iGEM Parts Registry and `ComponentDefinition` objects that represent TetR mRNA, TetR, LacI, and IPTG. Each `ComponentDefinition` is associated with a `Sequence` that has an `IUPAC DNA/RNA` or `IUPAC protein` `encoding`, except the `ComponentDefinition` of IPTG, which is associated with a `Sequence` that has a `SMILES` `encoding`.*

doi:10.2390/biecoll-jib-2015-272

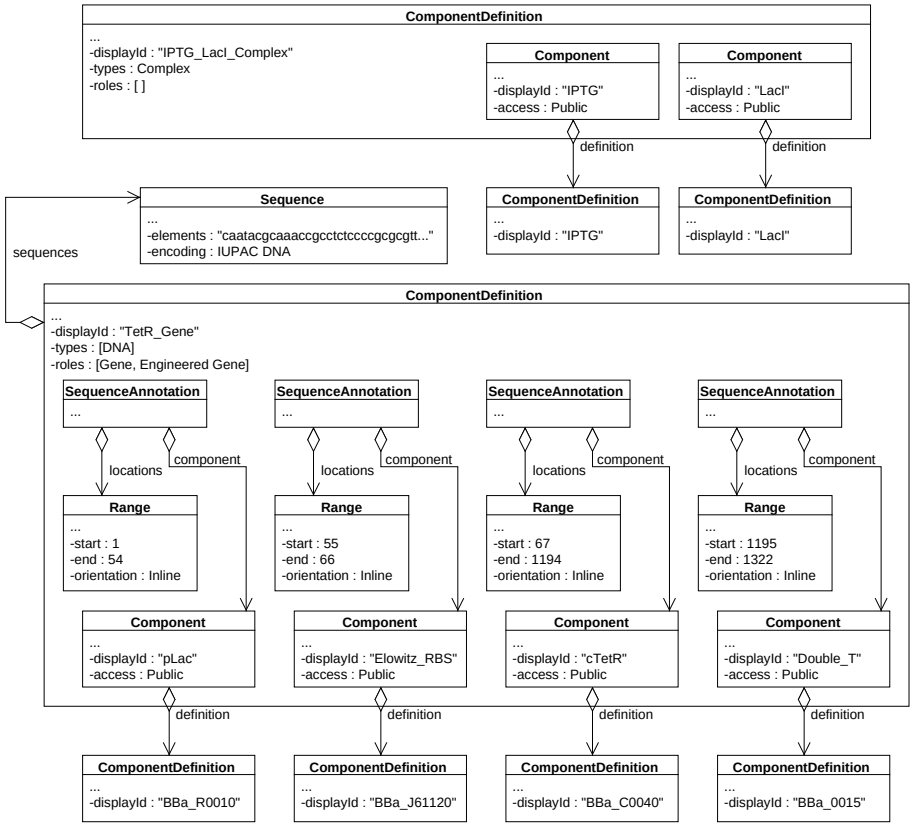


Figure 26: *Composite `ComponentDefinition` objects for the LacI inverter. In the case of the `ComponentDefinition` that represents the TetR gene, its sub-`Component` objects are located as `Range`s along its `Sequence` using `SequenceAnnotation` objects. The `ComponentDefinition` that represents the IPTG-LacI complex, however, has no `Sequence` and its sub-`Component` objects are composed without any data about their relative positions.*

doi:10.2390/biecoll-jib-2015-272

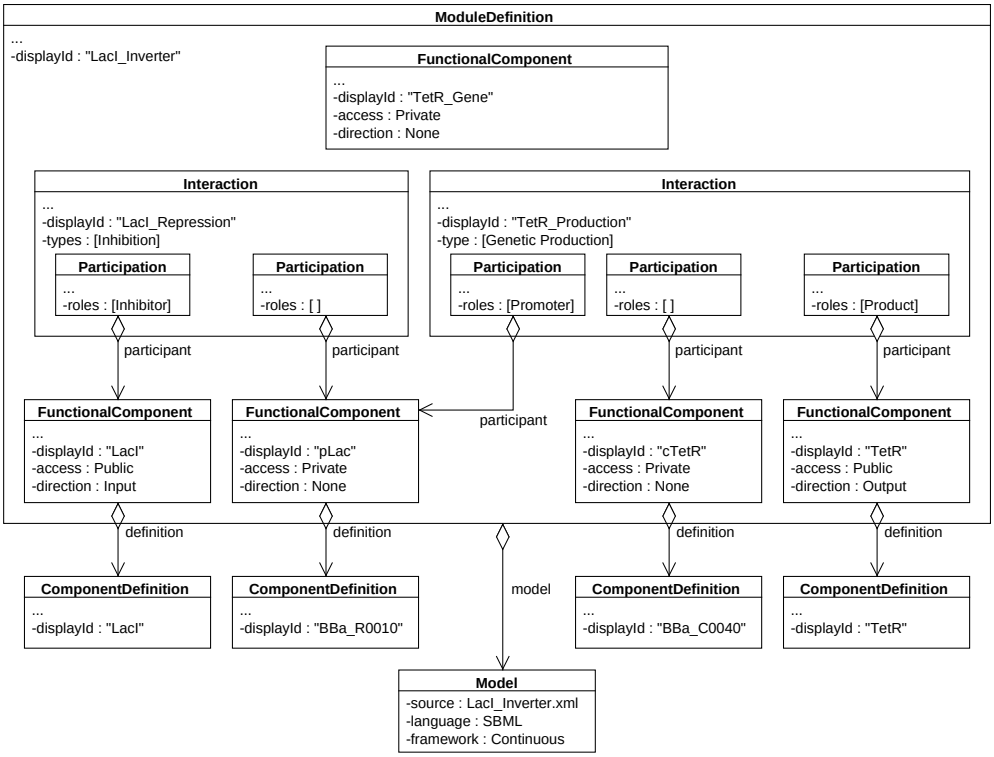


Figure 27: `ModuleDefinition` *of the LacI inverter. This* `ModuleDefinition` *contains* `FunctionalComponent` *objects that instantiate the* `ComponentDefinition` *objects for the LacI/TetR transcription factors and TetR gene. The* `FunctionalComponent` *for the TetR gene as a whole does not participate in any* `Interaction` *and merely indicates that the function of the gene is described by the LacI inverter* `ModuleDefinition`*. The remaining* `FunctionalComponent` *objects participate in a repression* `Interaction` *and a genetic production* `Interaction`*, thereby indicating which biological structures carry out the function of the LacI inverter* `ModuleDefinition`*. In this case, the transcription and translation of TetR are represented as a single genetic production* `Interaction` *that abstracts away the presence of the intermediate TetR mRNA. In addition, this* `ModuleDefinition` *is also associated with a continuous* `Model` *written in the SBML source file "LacI_Inverter.xml."*

doi:10.2390/biecoll-jib-2015-272

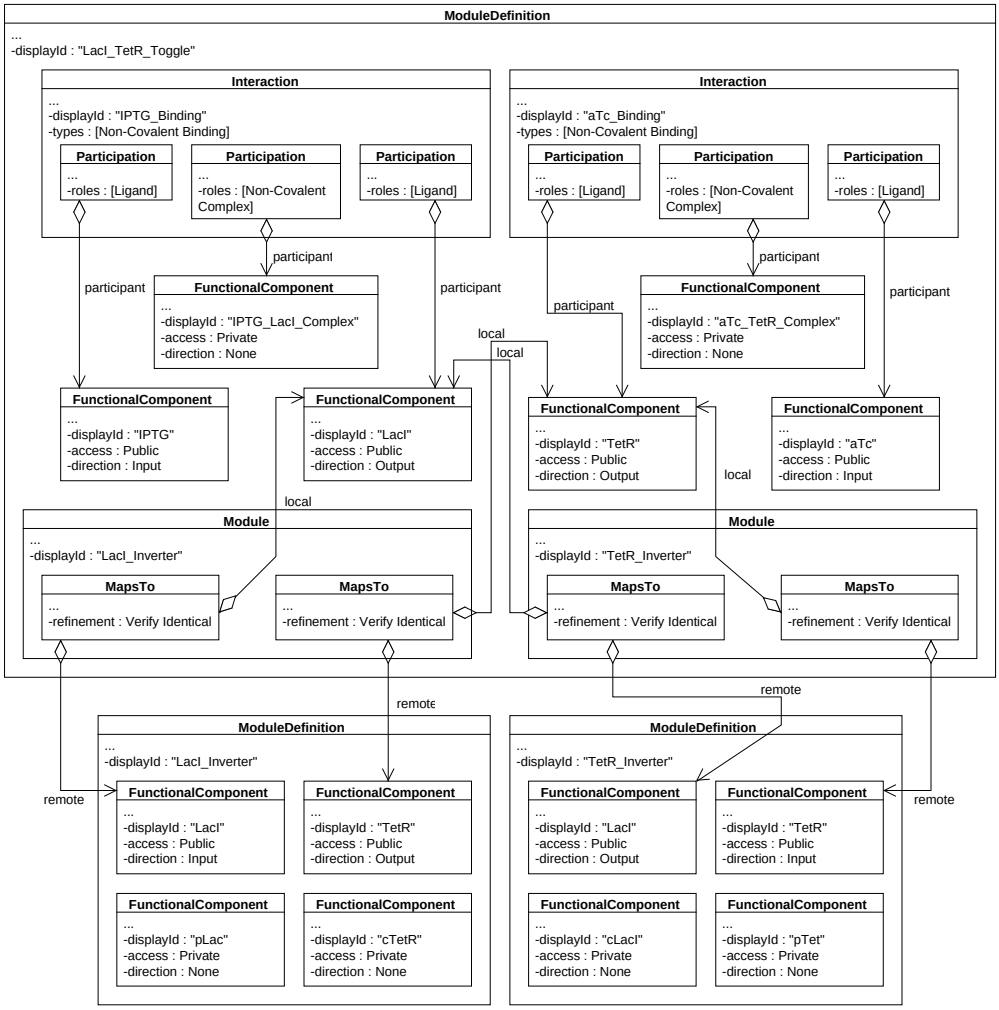


Figure 28: *Composite* `ModuleDefinition` *of the LacI/TetR toggle switch. This* `ModuleDefinition` *contains the* `Module` *objects that instantiate LacI and TetR inverter* `ModuleDefinition` *objects. It also contains* `FunctionalComponent` *objects that instantiate the* `ComponentDefinition` *objects for the LacI/TetR transcription factors and IPTG/aTc small molecules. These* `FunctionalComponent` *objects each participate in a non-covalent binding* `Interaction`*. To complete the composition of the toggle switch,* `MapsTo` *objects are used to indicate that the output of the LacI inverter* `ModuleDefinition` *is identical to the input of the TetR inverter* `ModuleDefinition` *and vice versa.*

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

10 SBOL RDF Serialization

In order for SBOL objects to be readily stored and exchanged, it is important that they are able to be *serialized*, i.e., converted to a sequence of bytes that can be stored in a file or exchanged over a network. The serialization format for SBOL is designed to meet several competing requirements. First, SBOL needs to support ad-hoc annotations and extensions. Second, SBOL needs to support processing by general database and semantic web software tools that have little or no knowledge of the SBOL data model. Finally, it ought to be relatively simple to write a new software implementation, so that SBOL can be readily used even in software environments where community-maintained implementations are not available.

To meet these goals, the canonical serialization of SBOL has been selected to be a strict dialect of RDF/XML Beckett and McBride (2004), a syntactic standard defined for Semantic Web data exchange. This serialization provides a standard base from which to meet further requirements. Moreover, it allows any RDF/XML-aware software tool to consume and operate on an SBOL file without needing any customization to support SBOL. Where possible, we have re-used predicates from widely-used terminologies (such as Dublin Core DCMI Usage Board (2012)) to expose as much of the data as practical to such standard RDF tooling.

Arbitrary RDF/XML, however, provides a sometimes problematically large amount of flexibility in how equivalent data can be serialized. This flexibility can result in different serializations when processing RDF/XML files using standard off-the-shelf XML tools, such as DOM-OO mappings. To address this issue, we define a canonical association between the nesting of data structures within the SBOL UML data model and the RDF/XML file. For all ownership associations (filled diamonds), the RDF/XML for the owned entity is embedded within the owner's RDF/XML (note, however, that the property values MAY be listed in any order). For all associations that are by reference (open diamonds), the RDF/XML for the referenced property is linked via a resource `URI`. For example, the serialization of a `ComponentDefinition` embeds the serializations of the `SequenceConstraint` and `Component` objects associated with it. Those `SequenceConstraint` objects, however, link to the `Component` objects with a `URI` rather than embedding another copy.

Every SBOL document MUST be a valid RDF/XML document. Accordingly, each SBOL document starts with an XML declaration that has its XML version set to "1.0." As shown in the example below, this declaration is then followed by an `rdf:RDF` XML element that includes the namespace declarations for RDF, Dublin Core, PROV, and SBOL. The SBOL namespace, which is `http://sbols.org/v2#`, is used to indicate which entities and properties in the SBOL document are defined by SBOL, and MUST NOT be used for any entities or properties not defined in this specification.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
...
</rdf:RDF>
```

All first-class SBOL data types (i.e., those enumerated in Section 7) have an associated identifying `URI`. In the RDF, this is the resource `URI` used by instances of that type. For example, `ComponentDefinition` has the type `URI` `sbol:ComponentDefinition`. Properties and associations are then asserted as nested RDF/XML assertions. Section 7 provides the serialization template and an example at the end of its description of each data type. All of the data types that are `TopLevel` are so named because they always appear at the top-most level of the RDF/XML serialization. All other data types will always appear nested within their parent container and, ultimately, some `TopLevel` object. For example, a `ComponentDefinition` is a `TopLevel` and therefore listed at the top-most level of the RDF/XML serialization, and contains its `SequenceConstraint` objects, since they are not `TopLevel`. Its `Sequence`, however, is also `TopLevel` and is therefore not nested within and instead linked via a `URI`.

Each instance of a first-class SBOL data type MAY have annotations attached, as described in Section 7.11. These annotations are composed of a name and a value. They are serialized to RDF as a conceptual triple with the subject being the identity of the instance they annotate, the predicate being the name of the annotation, and the object

doi:10.2390/biecoll-jib-2015-272

being the value of that annotation. Annotation values are always nested within the RDF/XML serialization of the instance that they annotate. For example, a `ModuleDefinition` might add a DOI annotation that links to the scientific article that first described the system that it represents.

SBOL also supports top-level, user-defined data, again as described in Section 7.11. This is to allow non-standardized but necessary information to be carried around as part of a design. For example, a particular sub-community might have an internal standard for genetic device characterization data sheets. Such data can be represented as a `GenericTopLevel` object with internal structured annotations. For example, each individual data sheet might be contained in its own `GenericTopLevel` instance. This custom data is serialized into the RDF/XML in the usual way, as an RDF/XML block at the top level of the file. Other objects can refer to this entity through their annotations by reference, and this generic top-level entity can refer to other entities via references. For example, a `ModuleDefinition` might use an annotation to refer to the data sheet `GenericTopLevel` that documents its properties.

By adopting this paradigm of RDF/XML serialization, SBOL is able to adapt to future changes in the standard without requiring large-scale alterations to the RDF files. Since exactly the same scheme is used to serialize annotations as is used to serialize specification-defined properties and associations, it is possible to update the SBOL standard to recognize a different range of properties and associations. Those properties not recognized by the specification will always be available through the API as annotations. Similarly, by allowing arbitrary top-level entities in a SBOL file, we enable future specifications or extensions to ratify the structure of other top-level objects. These entities would then become part of the explicit data model, but the identical RDF serialization would be used. Applications lacking support for a given extension can safely read in, manipulate, and write out the top-level data that is not understood, treating it as a top-level structured annotation, without data loss or corruption. Finally, the very regimented control of nesting versus referencing also allows the XML structure to be very predictable, enabling XML/DOM-based tooling to work with SBOL RDF/XML files safely.

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

11 Recommended Best Practices

11.1 Use of the Version Property

Once an SBOL object has been published where others might have access it (e.g., to an online repository), it might be the case that other objects come to depend on the particular contents of the published object. Thus, in order to avoid creating conflicting data, if a person wants to change the properties of a published object, they SHOULD do so by making a new copy of object that incorporates the change and has an `identity` property that contains a new `URI`.

The relationship between the old and new objects (i.e., that the new object was derived from the old object), however, is not visible unless it is explicitly declared. This is RECOMMENDED to be done using the `persistentIdentity`, and `version` properties. The preferred practice for declaring such a relationship is to use the same `persistentIdentity` for both objects, but give the newer object a later `version`. Then, when the new object is published, it can be clear to both humans and machines that this object is intended to update the previously published object. In this way, when a user wants the latest version of an object, they can obtain it by referencing the object via its `persistentIdentity` and rely on a tool to find the object with that `persistentIdentity` and the latest `version`.

As stated in Section 7.4, it is RECOMMENDED that version numbering follow the conventions of semantic versions (`http://semver.org/`), particularly as implemented by Maven (`http://maven.apache.org/`). This convention represents versions as sequences of numbers and qualifiers separated by the characters `.` and `-` and compared in lexicographical order (for example, 1 < 1.3.1 < 2.0-beta). For a full explanation, see the linked resources.

11.2 Compliant SBOL Objects

Maintaining unique `identity` `URI`s for all SBOL objects can be a very challenging implementation task. To reduce this burden, users of SBOL 2.0 are encouraged to follow a few simple rules when constructing the `identity` properties and related properties for SBOL objects. When these rules are followed in constructing an SBOL object, we say that this object is *compliant*. These rules are as follows:

1. The `identity` of a compliant SBOL object MUST begin with a *URI prefix* that maps to a domain over which the user has control. Namely, the user can guarantee uniqueness of identities within this domain.
2. The `persistentIdentity` and `displayId` properties are REQUIRED of a compliant SBOL object.
3. The `persistentIdentity` of a compliant `TopLevel` object MUST end with a delimiter ('/', '#', or ':') followed by the `displayId` of the object.
4. The `persistentIdentity` of a compliant SBOL object that is not also a `TopLevel` object MUST begin with the `persistentIdentity` of its parent object and be immediately followed by a delimiter ('/', '#', or ':') and the `displayId` of the compliant object.
5. If a compliant SBOL object is not given a `version`, then its `identity` and `persistentIdentity` properties MUST contain the same `URI`.
6. If a compliant SBOL object has a `version`, then its `identity` property MUST contain a `URI` of the form "⟨persistentIdentity⟩/⟨version⟩".
7. The `version` of a compliant SBOL object that is not also a `TopLevel` object MUST contain the same `String` as the `version` property of the compliant object's parent object.
8. The `identity`, `persistentIdentity`, `displayId`, and `version` properties of a compliant SBOL object MUST NOT be changed once set.

All examples in this specification use compliant `URI`s.

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

11.3 Annotations: Embedded Objects vs. External References

When annotating an SBOL document with additional information, there are two general methods that can be used:

- Embed the information in the SBOL document, either as non-SBOL properties or `GenericTopLevel` objects.
- Store the information separately and annotate the SBOL document with `URI`s that point to it.

In theory, either method can be used in any case. (Note that a third case not discussed here is to use SBOL to annotate external objects with linking to SBOL documents, rather than annotate SBOL documents with links external objects.)

In practice, embedding massive amounts of non-SBOL data into SBOL documents is likely to cause problems for people and software tools trying to manage and exchange such documents. Therefore, it is RECOMMENDED that small amounts of information (e.g., design notes or preferred graphical layout) be embedded in the SBOL model, while large amounts of information (e.g., the contents of the scientific publication from which a model was derived or flow cytometry data that characterizes performance) be linked with URIs pointing to external resources. The boundary between "small" and "large" is left deliberately vague, recognizing that it will likely depend on the particulars of a given SBOL application.

11.4 Completeness and Validation

RDF documents containing serialized SBOL objects might or might not be entirely self-contained. A SBOL document is self-contained or "complete" if every SBOL object referred to in the document is contained in the document. It is RECOMMENDED that serializations be complete whenever practical. In order words, when serializing an SBOL object, serialize all of the other objects that it points to, then serialize all of the other objects that these objects point to, etc., until the document is complete.

It is important to note that there is no guarantee that an RDF document contains valid SBOL. When an RDF document is deserialized to SBOL objects, the program doing so SHOULD verify that all of the property values encoded therein have the right data type (e.g., that the object pointed to by the `sequences` property of a `ComponentDefinition` is really a `Sequence`). For complete files, this validation can be carried out entirely locally. For files that are not complete, an implementation either needs to have a means of validating those external references (e.g., by retrieving them from various repositories), or it needs to mark them as unverified and not depend on their correctness.

11.5 Recommended Ontologies for External Terms

External ontologies and controlled vocabularies are an integral part of SBOL. SBOL uses `URI`s to access existing biological information through these resources. New SBOL specific terms are defined only when necessary. For example, `ComponentDefinition types`, such as DNA or protein, are described using BioPAX terms. Similarly, the roles of a DNA `ComponentDefinition` are described via SO terms. Although RECOMMENDED ontologies have been indicated in relevant sections where possible, other resources providing similar terms can also be used. A summary of these external sources can be found in Table 13.

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

SBOL Entity	Property	Preferred External Resource	More Information
ComponentDefinition	types	BioPAX	http://www.biopax.org
	roles	SO (*DNA* or *RNA*)	http://www.sequenceontology.org
	roles	CHEBI (*small molecule*)	https://www.ebi.ac.uk/chebi/
Interaction	types	SBO (occurring entity branch)	http://www.ebi.ac.uk/sbo/main/
Participation	roles	SBO (participant roles branch)	http://www.ebi.ac.uk/sbo/main/
Model	language	EDAM	http://bioportal.bioontology.org/ontologies/EDAM
	framework	SBO (modeling framework branch)	http://www.ebi.ac.uk/sbo/main/

Table 13: *Preferred external resources from which to draw values for various SBOL properties.*

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

References

Beckett, D. and McBride, B. (2004). RDF/XML syntax specification (revised). *W3C recommendation*, 10.

Canton, B., Labno, A., and Endy, D. (2008). Refinement and standardization of synthetic biological parts and devices. *Nature Biotechnology*, 26(7):787–793.

DCMI Usage Board (2012). DCMI metadata terms. DCMI recommendation, Dublin Core Metadata Initiative.

Finney, A., Hucka, M., Bornstein, B. J., Keating, S. M., Shapiro, B. M., Matthews, J., Kovitz, B. K., Schilstra, M. J., Funahashi, A., Doyle, J., and Kitano, H. (2006). Software infrastructure for effective communication and reuse of computational models. In Szallasi, Z., Stelling, J., and Periwal, V., editors, *System Modeling in Cell Biology: From Concepts to Nuts and Bolts*. MIT Press.

Galdzicki, M., Clancy, K. P., Oberortner, E., Pocock, M., Quinn, J. Y., Rodriguez, C. A., Roehner, N., Wilson, M. L., Adam, L., Anderson, J. C., et al. (2014). The synthetic biology open language (SBOL) provides a community standard for communicating designs in synthetic biology. *Nature biotechnology*, 32(6):545–550.

Gardner, T. S., Cantor, C. R., and Collins, J. J. (2000). Construction of a genetic toggle switch in escherichia coli. *Nature*, 403:339–342.

Norrander, J., Kempe, T., and Messing, J. (1983). Construction of improved m13 vectors using oligodeoxynucleotide-directed mutagenesis. *Gene*, 26:101–106.

Roehner, N., Oberortner, E., Pocock, M., Beal, J., Clancy, K., Madsen, C., Misirli, G., Wipat, A., Sauro, H., and Myers, C. J. (2015). Proposed data model for the next version of the synthetic biology open language. *ACS synthetic biology*, 4(1):57–71.

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

A Validation Rules

This section summarizes all the conditions that either MUST be or are RECOMMENDED to be true of an SBOL Version 2.0 document. There are different degrees of rule strictness. Rules of the former kind are strict SBOL validation rules—data encoded in SBOL MUST conform to all of them in order to be considered valid. Rules of the latter kind are consistency rules that SBOL data are RECOMMENDED to adhere to as a best practice. To help highlight these differences, we use the following symbols next to the rule numbers:

- ☑ A checked box indicates a strong REQUIRED condition for SBOL conformance. If a SBOL document does not follow this rule, it does not conform to the SBOL specification. (Mnemonic intention behind the choice of symbol: "This MUST be checked.")
- ▲ A triangle indicates a weak REQUIRED condition for SBOL conformance. While this rule MUST be followed, it can be difficult, if not impossible, for a machine to automatically check whether the rule has been followed. (Mnemonic intention behind the choice of symbol: "This is a cause for warning.")
- ★ A star indicates a RECOMMENDED condition for following best practices. This rule is not strictly a matter of SBOL conformance, but its recommendation comes from logical reasoning. If an SBOL document does not follow this rule, it is still valid SBOL, but it might have degraded functionality in some tools. (Mnemonic intention behind the choice of symbol: "You're a star if you heed this.")

The validation rules listed in the following subsections are all believed to be stated or implied in the rest of this specification document. They are enumerated here for convenience and to provide a "master checklist" for SBOL validation. In case of a conflict between this section and other portions of the specification (though there are believed to be none), this section is considered authoritative for the purpose of determining the validity of an SBOL document.

☞ For convenience and brevity, we use the shorthand "`sbol:x`" to stand for an attribute or element name `x` in the namespace for the SBOL specification, using the namespace prefix `sbol`. In reality, the prefix string can be different from the literal "`sbol`" used here (and indeed, it can be any valid XML namespace prefix that the software chooses). We use "`sbol:x`" because it is shorter than to write a full explanation everywhere we refer to an attribute or element in the SBOL specification namespace.

General rules for an SBOL document

sbol-10101 ☑ An SBOL document MUST declare the use of the following XML namespace: "`http://sbols.org/v2#`".
Reference: Section 10 on page 54

sbol-10102 ☑ An SBOL document MUST declare the use of the following XML namespace: "`http://www.w3.org/1999/02/22-rdf-syntax-ns#`".
Reference: Section 10 on page 54

sbol-10103 ☑ If an SBOL document includes any `name` or `description` properties, then it MUST declare the use of the following XML namespace: "`http://purl.org/dc/terms/`".
Reference: Section 10 on page 54

sbol-10104 ☑ If an SBOL document includes any `wasDerivedFrom` properties, then it MUST declare the use of the following XML namespace: "`http://www.w3.org/ns/prov#`".
Reference: Section 10 on page 54

doi:10.2390/biecoll-jib-2015-272

Rules for the Identified class

sbol-10201 ☑ The `identity` property of an `Identified` object is REQUIRED and MUST contain a `URI` that adheres to the syntax defined by:
"`http://www.w3.org/1999/02/22-rdf-syntax#about`"
Reference: Section 7.4 on page 16

sbol-10202 ▲ The `identity` property of an `Identified` object MUST be globally unique.
Reference: Section 7.4 on page 16

sbol-10203 ☑ The `persistentIdentity` property of an `Identified` object is OPTIONAL and MAY contain a `URI` that MUST adhere to the syntax defined by:
"`http://www.w3.org/1999/02/22-rdf-syntax#about`"
Reference: Section 7.4 on page 16

sbol-10204 ☑ The `displayId` property of an `Identified` object is OPTIONAL and MAY contain a `String` that MUST be composed of only alphanumeric or underscore characters and MUST NOT begin with a digit.
Reference: Section 7.4 on page 16

sbol-10205 ★ The `displayId` property of an `Identified` object SHOULD be locally unique.
Reference: Section 7.4 on page 16

sbol-10206 ☑ The `version` property of an `Identified` object is OPTIONAL and MAY contain a `String` that MUST be composed of only alphanumeric characters, underscores, hyphens, or periods and MUST begin with a digit.
Reference: Section 7.4 on page 16

sbol-10207 ★ The `version` property of an `Identified` object SHOULD follow the conventions of semantic versioning as implemented by Maven.
Reference: Section 7.4 on page 16

sbol-10208 ☑ The `wasDerivedFrom` property of an `Identified` object is OPTIONAL and MAY contain a `URI`.
Reference: Section 7.4 on page 16

sbol-10209 ☑ The `wasDerivedFrom` property of an `Identified` object MUST NOT contain a `URI` reference to the `Identified` object itself.
Reference: Section 7.4 on page 16

sbol-10210 ▲ `Identified` objects MUST NOT form circular reference chains via their `wasDerivedFrom` properties.
Reference: Section 7.4 on page 16

sbol-10211 ▲ If the `wasDerivedFrom` property of one `Identified` object refers to another `Identified` object with the same `persistentIdentity` property, then `version` property of the second `Identified` object MUST precede that of the first if both objects have a `version`.
Reference: Section 7.4 on page 16

sbol-10212 ☑ The `name` property of an `Identified` object is OPTIONAL and MAY contain a `String`.
Reference: Section 7.4 on page 16

sbol-10213 ☑ The `description` property of an `Identified` object is OPTIONAL and MAY contain a `String`.
Reference: Section 7.4 on page 16

sbol-10214 ☑ The `annotations` property of an `Identified` object is OPTIONAL and MAY contain a set of `Annotation` objects.
Reference: Section 7.4 on page 16

doi:10.2390/biecoll-jib-2015-272

Section

sbol-10215 ★ The `displayId` property of a compliant `Identified` object is REQUIRED.
Reference: Section 11.2 on page 56

sbol-10216 ★ The `persistentIdentity` property of a compliant `TopLevel` object is REQUIRED and MUST contain a `URI` that ends with a delimiter ('/', '#', or ':') followed by the `displayId` of the `TopLevel` object.
Reference: Section 11.2 on page 56

sbol-10217 ★ The `persistentIdentity` property of a compliant `Identified` object that is not also a `TopLevel` object is REQUIRED and MUST contain a `URI` that begins with the `persistentIdentity` of the compliant object's parent and is immediately followed by a delimiter ('/', '#', or ':') and the `displayId` of the compliant object.
Reference: Section 11.2 on page 56

sbol-10218 ★ If a compliant `Identified` object has no `version` property, then its `identity` property MUST contain the same `URI` as its `persistentIdentity` property. Otherwise, the compliant object's `identity` property MUST contain a `URI` that begins with its `persistentIdentity` and is immediately followed by a delimiter ('/', '#', or ':') and its `version`.
Reference: Section 11.2 on page 56

sbol-10219 ★ The `version` property of a compliant `Identified` object that is not also a `TopLevel` object is REQUIRED to contain the same `String` as the `version` property of the compliant object's parent.
Reference: Section 11.2 on page 56

Rules for the TopLevel class

sbol-10301 ☑ A `TopLevel` object MUST inherit all properties of the `Identified` class.
Reference: Section 7.5 on page 19

Rules for the Sequence class

sbol-10401 ☑ A `Sequence` MUST inherit all properties of the `TopLevel` class.
Reference: Section 7.6 on page 19

sbol-10402 ☑ The `elements` property of a `Sequence` is REQUIRED and MUST contain a `String`.
Reference: Section 7.6 on page 19

sbol-10403 ☑ The `encoding` property of `Sequence` is REQUIRED and MUST contain a `URI`.
Reference: Section 7.6 on page 19

sbol-10404 ▲ The `encoding` property of a `Sequence` MUST indicate how the `elements` property of the `Sequence` is to be formed and interpreted.
Reference: Section 7.6 on page 19

sbol-10405 ▲ The `elements` property of a `Sequence` MUST be consistent with its `encoding` property.
Reference: Section 7.6 on page 19

sbol-10406 ▲ The `encoding` property of a `Sequence` MUST contain a `URI` from Table 1 if it is well-described by this `URI`.
Reference: Section 7.6 on page 19

Rules for the ComponentDefinition class

sbol-10501 ☑ A `ComponentDefinition` MUST inherit all properties of the `TopLevel` class.
Reference: Section 7.7 on page 21

doi:10.2390/biecoll-jib-2015-272

Section

sbol-10502 ☑ The `types` property of a `ComponentDefinition` is REQUIRED and MUST contain a non-empty set of `URI`s.
Reference: Section 7.7 on page 21

sbol-10503 ☑ The `types` property of a `ComponentDefinition` MUST NOT contain more than one `URI` from Table 2.
Reference: Section 7.7 on page 21

sbol-10504 ▲ Each `URI` contained by the `types` property of a `ComponentDefinition` MUST refer to an ontology term that describes the category of biochemical or physical entity that is represented by the `ComponentDefinition`.
Reference: Section 7.7 on page 21

sbol-10505 ▲ The `types` property of a `ComponentDefinition` MUST contain a `URI` from Table 2 if it is well-described by this `URI`.
Reference: Section 7.7 on page 21

sbol-10506 ▲ All `URI`s contained by the `types` property of a `ComponentDefinition` MUST refer to non-conflicting ontology terms.
Reference: Section 7.7 on page 21

sbol-10507 ☑ The `roles` property of a `ComponentDefinition` is OPTIONAL and MAY contain a set of `URI`s.
Reference: Section 7.7 on page 21

sbol-10508 ▲ Each `URI` contained by the `roles` property of a `ComponentDefinition` MUST refer to an ontology term that clarifies the potential function of the `ComponentDefinition` in a biochemical or physical context.
Reference: Section 7.7 on page 21

sbol-10509 ▲ Each `URI` contained by the `roles` property of a `ComponentDefinition` MUST refer to an ontology term that is consistent with its `types` property.
Reference: Section 7.7 on page 21

sbol-10510 ▲ The `roles` property of a `ComponentDefinition` MUST contain a `URI` from Table 3 if it is well-described by this `URI`.
Reference: Section 7.7 on page 21

sbol-10511 ★ The `roles` property of a `ComponentDefinition` SHOULD only contain a `URI` provided in Table 3 if one of its `types` is cross-listed with this `URI`.
Reference: Section 7.7 on page 21

sbol-10512 ☑ The `sequences` property of a `ComponentDefinition` is OPTIONAL and MAY contain a set of `URI`s.
Reference: Section 7.7 on page 21

sbol-10513 ▲ Each `URI` contained by the `sequences` property of a `ComponentDefinition` MUST refer to a `Sequence` object.
Reference: Section 7.7 on page 21

sbol-10514 ▲ The `Sequence` objects referred to by the `sequences` property of a `ComponentDefinition` MUST be consistent with each other, such that well-defined mappings exist between their `elements` properties in accordance with their `encoding` properties.
Reference: Section 7.7 on page 21

sbol-10515 ▲ The `sequences` property of a `ComponentDefinition` MUST NOT refer to `Sequence` objects with conflicting `encoding` properties.
Reference: Section 7.7 on page 21

doi:10.2390/biecoll-jib-2015-272

sbol-10516 ▲ If the `sequences` property of a `ComponentDefinition` refers to one or more `Sequence` objects, and one of the `types` of this `ComponentDefinition` comes from Table 2, then one of the `Sequence` objects MUST have the `encoding` that is cross-listed with this type in Table 1.
Reference: Section 7.7 on page 21

sbol-10517 ▲ If the `sequences` property of a `ComponentDefinition` refers to a `Sequence` with an `encoding` from Table 1, then the `types` property of the `ComponentDefinition` MUST contain the type from Table 2 that is cross-listed with this `encoding` in Table 1.
Reference: Section 7.7 on page 21

sbol-10518 ★ If a `ComponentDefinition` refers to more than one `Sequence` with the same `encoding`, then the `elements` of these `Sequence` objects SHOULD have equal lengths.
Reference: Section 7.7 on page 21

sbol-10519 ☑ The `components` property of a `ComponentDefinition` is OPTIONAL and MAY contain a set of `Component` objects.
Reference: Section 7.7 on page 21

sbol-10520 ★ If a `ComponentDefinition` in a `ComponentDefinition-Component` hierarchy refers to one or more `Sequence` objects, and there exist `ComponentDefinition` objects lower in the hierarchy that refer to `Sequence` objects with the same `encoding`, then the `elements` properties of these `Sequence` objects SHOULD be consistent with each other, such that well-defined mappings exist from the "lower level" `elements` to the "higher level" `elements` in accordance with their shared `encoding` properties. This mapping is also subject to any restrictions on the positions of the `Component` objects in the hierarchy that are imposed by the `SequenceAnnotation` or `SequenceConstraint` objects contained by the `ComponentDefinition` objects in the hierarchy.
Reference: Section 7.7 on page 21

sbol-10521 ☑ The `sequenceAnnotations` property of a `ComponentDefinition` is OPTIONAL and MAY contain a set of `SequenceAnnotation` objects.
Reference: Section 7.7 on page 21

sbol-10522 ☑ The `sequenceAnnotations` property of a `ComponentDefinition` MUST NOT contain two or more `SequenceAnnotation` objects that refer to the same `Component`.
Reference: Section 7.7 on page 21

sbol-10523 ★ If the `sequences` property of a `ComponentDefinition` refers to a `Sequence` with an `IUPAC` `encoding` from Table 1, then each `SequenceAnnotation` that includes a `Range` and/or `Cut` in the `sequenceAnnotations` property of the `ComponentDefinition` SHOULD specify a region on the `elements` of this `Sequence`.
Reference: Section 7.7 on page 21

sbol-10524 ☑ The `sequenceConstraints` property of a `ComponentDefinition` is OPTIONAL and MAY contain a set of `SequenceConstraint` objects.
Reference: Section 7.7 on page 21

Rules for the ComponentInstance class

sbol-10601 ☑ A `ComponentInstance` MUST inherit all properties of the `Identified` class.
Reference: Section 7.7.1 on page 24

sbol-10602 ☑ The `definition` property of a `ComponentInstance` is REQUIRED and MUST contain a `URI`.
Reference: Section 7.7.1 on page 24

doi:10.2390/biecoll-jib-2015-272

sbol-10603 ☑ The `definition` property of a `ComponentInstance` MUST NOT contain a `URI` reference to the `ComponentDefinition` that contains the `ComponentInstance`.
Reference: Section 7.7.1 on page 24

sbol-10604 ▲ The `URI` contained by the `definition` property MUST refer to a `ComponentDefinition` object.
Reference: Section 7.7.1 on page 24

sbol-10605 ▲ `ComponentInstance` objects MUST NOT form circular reference chains via their `definition` properties and parent `ComponentDefinition` objects.
Reference: Section 7.7.1 on page 24

sbol-10606 ☑ The `mapsTos` property of a `ComponentInstance` is OPTIONAL and MAY contain a set of `MapsTo` objects.
Reference: Section 7.7.1 on page 24

sbol-10607 ☑ The `access` property of a `ComponentInstance` is REQUIRED and MUST contain a `URI` from Table 4
Reference: Section 7.7.1 on page 24

Rules for the Component class

sbol-10701 ☑ A `Component` MUST inherit all properties of the `ComponentInstance` class.
Reference: Section 7.7.1 on page 24

Rules for the MapsTo class

sbol-10801 ☑ A `MapsTo` MUST inherit all properties of the `Identified` class.
Reference: Section 7.7.3 on page 26

sbol-10802 ☑ The `local` property of a `MapsTo` is REQUIRED and MUST contain a `URI`.
Reference: Section 7.7.3 on page 26

sbol-10803 ☑ If a `MapsTo` is contained by a `Component` in a `ComponentDefinition`, then the `local` property of the `MapsTo` MUST refer to another `Component` in the `ComponentDefinition`.
Reference: Section 7.7.3 on page 26

sbol-10804 ☑ If a `MapsTo` is contained by a `FunctionalComponent` or `Module` in a `ModuleDefinition`, then the `local` property of the `MapsTo` MUST refer to another `FunctionalComponent` in the `ModuleDefinition`.
Reference: Section 7.7.3 on page 26

sbol-10805 ☑ The `remote` property of a `MapsTo` is REQUIRED and MUST contain a `URI`.
Reference: Section 7.7.3 on page 26

sbol-10806 ▲ The `remote` property of a `MapsTo` MUST refer to a `ComponentInstance`.
Reference: Section 7.7.3 on page 26

sbol-10807 ▲ The `ComponentInstance` referred to by the `remote` property of a `MapsTo` MUST have an `access` property that contains the `URI` `http://sbols.org/v2#public`.
Reference: Section 7.7.3 on page 26

sbol-10808 ▲ If a `MapsTo` is contained by a `ComponentInstance`, then the `remote` property of the `MapsTo` MUST refer to a `Component` in the `ComponentDefinition` that is referenced by the `definition` property of the `ComponentInstance`.
Reference: Section 7.7.3 on page 26

doi:10.2390/biecoll-jib-2015-272

Section

sbol-10809 ▲ If a `MapsTo` is contained by a `Module`, then the `remote` property of the `MapsTo` MUST refer to a `FunctionalComponent` in the `ModuleDefinition` that is referenced by the `definition` property of the `Module`.
Reference: Section 7.7.3 on page 26

sbol-10810 ☑ The `refinement` property of a `MapsTo` is REQUIRED and MUST contain a `URI` from Table 5.
Reference: Section 7.7.3 on page 26

sbol-10811 ▲ If the `refinement` property of a `MapsTo` contains the `URI` `http://sbols.org/v2#verifyIdentical`, then the `ComponentInstance` objects referred to by `local` and `remote` properties of the `MapsTo` MUST refer to the same `ComponentDefinition` via their `definition` properties.
Reference: Section 7.7.3 on page 26

Rules for the SequenceAnnotation class

sbol-10901 ☑ A `SequenceAnnotation` MUST inherit all properties of the `Identified` class.
Reference: Section 7.7.4 on page 29

sbol-10902 ☑ The `locations` property of a `SequenceAnnotation` is REQUIRED and MUST contain a non-empty set of `Location` objects.
Reference: Section 7.7.4 on page 29

sbol-10903 ★ The `Location` objects contained by the `locations` property of a single `SequenceAnnotation` SHOULD NOT specify overlapping regions.
Reference: Section 7.7.4 on page 29

sbol-10904 ☑ The `component` property is OPTIONAL and MAY contain a `URI` reference to a `Component`.
Reference: Section 7.7.4 on page 29

sbol-10905 ☑ The `Component` referenced by the `component` property of a `SequenceAnnotation` MUST be contained by the `ComponentDefinition` that contains the `SequenceAnnotation`.
Reference: Section 7.7.4 on page 29

Rules for the Location class

sbol-11001 ☑ A `Location` MUST inherit all properties of the `Identified` class.
Reference: Section 7.7.5 on page 30

sbol-11002 ☑ The `orientation` property of a `Location` is OPTIONAL and MAY contain a `URI` from Table 6.
Reference: Section 7.7.5 on page 32

Rules for the Range class

sbol-11101 ☑ A `Range` MUST inherit all properties of the `Location` class.
Reference: Section 7.7.5 on page 31

sbol-11102 ☑ The `start` property of a `Range` is REQUIRED and MUST contain an `Integer` greater than zero.
Reference: Section 7.7.5 on page 31

sbol-11103 ☑ The `end` property of a `Range` is REQUIRED and MUST contain an `Integer` greater than zero.
Reference: Section 7.7.5 on page 31

sbol-11104 ☑ The value of the `end` property of a `Range` MUST be greater than or equal to the value of its `start` property.
Reference: Section 7.7.5 on page 31

doi:10.2390/biecoll-jib-2015-272

Rules for the Cut class

sbol-11201 ☑ A `Cut` MUST inherit all properties of the `Location` class.
Reference: Section 7.7.5 on page 31

sbol-11202 ☑ The `at` property is REQUIRED and MUST contain an `Integer` greater than or equal to zero.
Reference: Section 7.7.5 on page 31

Rules for the GenericLocation class

sbol-11301 ☑ A `GenericLocation` MUST inherit all properties of the `Location` class.
Reference: Section 7.7.5 on page 32

Rules for the SequenceConstraint class

sbol-11401 ☑ A `SequenceConstraint` MUST inherit all properties of the `Identified` class.
Reference: Section 7.7.6 on page 32

sbol-11402 ☑ The `subject` property is REQUIRED and MUST contain a `URI` reference to a `Component`.
Reference: Section 7.7.6 on page 32

sbol-11403 ☑ The `Component` referenced by the `subject` property of a `SequenceConstraint` MUST be contained by the `ComponentDefinition` that contains the `SequenceConstraint`.
Reference: Section 7.7.6 on page 32

sbol-11404 ☑ The `object` property is REQUIRED and MUST contain a `URI` reference to a `Component`.
Reference: Section 7.7.6 on page 32

sbol-11405 ☑ The `Component` referenced by the `object` property of a `SequenceConstraint` MUST be contained by the `ComponentDefinition` that contains the `SequenceConstraint`.
Reference: Section 7.7.6 on page 32

sbol-11406 ☑ The `object` property of a `SequenceConstraint` MUST NOT refer to the same `Component` as the `subject` property of the `SequenceConstraint`.
Reference: Section 7.7.6 on page 32

sbol-11407 ☑ The `restriction` property is REQUIRED and MUST contain a `URI`.
Reference: Section 7.7.6 on page 32

sbol-11408 ▲ The `URI` contained by the `restriction` property of a `SequenceConstraint` MUST indicate the type of structural restriction on the relative, sequence-based positions or orientations of the `Component` objects referred to by the `subject` and `object` properties of the `SequenceConstraint`.
Reference: Section 7.7.6 on page 32

sbol-11409 ▲ If the `restriction` property of a `SequenceConstraint` contains the `URI` `http://sbols.org/v2#precedes`, then the position of the `Component` referred to by the `subject` property of the `SequenceConstraint` MUST precede that of the `Component` referred to by its `object` property.
Reference: Section 7.7.6 on page 32

sbol-11410 ▲ If the `restriction` property of a `SequenceConstraint` contains the `URI` `http://sbols.org/v2#sameOrientationAs`, then the orientation of the `Component` referred to by the `subject` property of the `SequenceConstraint` MUST be the same as that of the `Component` referred to by its `object` property.
Reference: Section 7.7.6 on page 32

sbol-11411 ▲ If the `restriction` property of a `SequenceConstraint` contains the `URI` `http://sbols.org/v2#oppositeOrientationAs`, then the orientation of the `Component` referred to by the

doi:10.2390/biecoll-jib-2015-272

`subject` property of the `SequenceConstraint` MUST be opposite that of the `Component` referred to by its `object` property.
Reference: Section 7.7.6 on page 32

sbol-11412 ★ The `URI` contained by the `restriction` property SHOULD come from Table 7.
Reference: Section 7.7.6 on page 32

Rules for the Model class

sbol-11501 ☑ A `Model` MUST inherit all properties of the `TopLevel` class.
Reference: Section 7.8 on page 34

sbol-11502 ☑ The `source` property is a REQUIRED and MUST contain a `URI`.
Reference: Section 7.8 on page 34

sbol-11503 ▲ The `URI` contained by the `source` property of a `Model` MUST specify the location of the model's source file.
Reference: Section 7.8 on page 34

sbol-11504 ☑ The `language` property is REQUIRED and MUST contain a `URI`.
Reference: Section 7.8 on page 34

sbol-11505 ▲ The `URI` contained by the `language` property of a `Model` MUST specify the language in which the model is encoded.
Reference: Section 7.8 on page 34

sbol-11506 ▲ The `language` property of a `Model` MUST contain a `URI` from Table 8 if it is well-described by this `URI`.
Reference: Section 7.8 on page 34

sbol-11507 ★ The `language` property of a `Model` SHOULD contain a `URI` that refers to a term from the EDAM ontology.
Reference: Section 7.8 on page 34

sbol-11508 ☑ The `framework` property is REQUIRED and MUST contain a `URI`.
Reference: Section 7.8 on page 34

sbol-11509 ▲ The `URI` contained by the `framework` property of a `Model` MUST specify the modeling framework of the model.
Reference: Section 7.8 on page 34

sbol-11510 ▲ The `framework` property of a `Model` MUST contain a `URI` from Table 9 if it is well-described by this `URI`.
Reference: Section 7.8 on page 34

sbol-11511 ★ The `framework` property SHOULD contain a `URI` that refers to a term from the modeling framework branch of the SBO.
Reference: Section 7.8 on page 34

Rules for the ModuleDefinition class

sbol-11601 ☑ A `ModuleDefinition` MUST inherit all properties of the `TopLevel` class.
Reference: Section 7.9 on page 35

sbol-11602 ☑ The `roles` property is OPTIONAL and MAY contain a set of `URI`s.
Reference: Section 7.9 on page 35

doi:10.2390/biecoll-jib-2015-272

Section

sbol-11603 ▲ Each `URI` contained by `roles` property of a `ModuleDefinition` MUST refer to a resource that clarifies the intended function of the `ModuleDefinition`.
Reference: Section 7.9 on page 35

sbol-11604 ☑ The `modules` property OPTIONAL and MAY contain a set of `Module` objects.
Reference: Section 7.9 on page 35

sbol-11605 ☑ The `interactions` property is OPTIONAL and MAY contain a set of `Interaction` objects.
Reference: Section 7.9 on page 35

sbol-11606 ☑ The `functionalComponents` property is OPTIONAL and MAY contain a set of `FunctionalComponent` objects.
Reference: Section 7.9 on page 35

sbol-11607 ☑ The `models` property is OPTIONAL and MAY contain a set of `URI`s.
Reference: Section 7.9 on page 35

sbol-11608 ▲ Each `URI` contained by the `models` property of a `ModuleDefinition` MUST refer to a `Model`.
Reference: Section 7.9 on page 35

Rules for the Module class

sbol-11701 ☑ A `Module` MUST inherit all properties of the `Identified` class.
Reference: Section 7.9.1 on page 37

sbol-11702 ☑ The `definition` property of a `Module` is REQUIRED and MUST contain a `URI`.
Reference: Section 7.9.1 on page 37

sbol-11703 ☑ The `URI` contained by the `definition` property of `Module` MUST refer to a `ModuleDefinition`.
Reference: Section 7.9.1 on page 37

sbol-11704 ☑ The `definition` property of a `Module` MUST NOT contain a `URI` reference to the `ModuleDefinition` that contains the `Module`.
Reference: Section 7.9.1 on page 37

sbol-11705 ▲ `Module` objects MUST NOT form circular reference chains via their `definition` properties and parent `ModuleDefinition` objects.
Reference: Section 7.9.1 on page 37

sbol-11706 ☑ The `mapsTos` property is OPTIONAL and MAY contain a set of `MapsTo` objects.
Reference: Section 7.9.1 on page 37

Rules for the FunctionalComponent class

sbol-11801 ☑ A `FunctionalComponent` MUST inherit all properties of the `ComponentInstance` class.
Reference: Section 7.7.1 on page 24

sbol-11802 ☑ The `direction` property of a `FunctionalComponent` is REQUIRED and MUST contain a `URI` from Table 10.
Reference: Section 7.9.2 on page 38

Rules for the Interaction class

sbol-11901 ☑ An `Interaction` MUST inherit all properties of the `Identified` class.
Reference: Section 7.9.3 on page 39

sbol-11902 ☑ The `types` property of an `Interaction` is REQUIRED and MUST contain a non-empty set of `URI`s.
Reference: Section 7.9.3 on page 39

doi:10.2390/biecoll-jib-2015-272

Section

sbol-11903 ▲ Each URI contained by the types property of an Interaction MUST refer to an ontology term that describes the behavior represented by the Interaction.
Reference: Section 7.9.3 on page 39

sbol-11904 ▲ All URIs contained by the types property of an Interaction MUST refer to non-conflicting ontology terms.
Reference: Section 7.9.3 on page 39

sbol-11905 ★ A least one URI contained by the types property of an Interaction SHOULD refer to a term from the occurring entity relationship branch of the SBO.
Reference: Section 7.9.3 on page 39

sbol-11906 ☑ The participations property of an Interaction is OPTIONAL and MAY contain a set of Participation objects.
Reference: Section 7.9.3 on page 39

Rules for the Participation class

sbol-12001 ☑ A Participation MUST inherit all properties of the Identified class.
Reference: Section 7.9.4 on page 41

sbol-12002 ☑ The participant property of a Participation is REQUIRED and MUST contain a URI reference to a FunctionalComponent.
Reference: Section 7.9.4 on page 41

sbol-12003 ☑ The FunctionalComponent referenced by the participant property of a Participation MUST be contained by the ModuleDefinition that contains the Interaction which contains the Participation.
Reference: Section 7.9.4 on page 41

sbol-12004 ☑ The roles property of an Participation is OPTIONAL and MAY contain a set of URIs.
Reference: Section 7.9.4 on page 41

sbol-12005 ▲ Each URI contained by the roles property of an Participation MUST refer to an ontology term that describes the behavior represented by the Participation.
Reference: Section 7.9.4 on page 41

sbol-12006 ▲ All URIs contained by the roles property of an Participation MUST refer to non-conflicting ontology terms.
Reference: Section 7.9.4 on page 41

sbol-12007 ★ A least one role in the set of roles SHOULD be a URI from the participant role branch of the SBO.
Reference: Section 7.9.4 on page 41

Rules for the Collection class

sbol-12101 ☑ A Collection MUST inherit all properties of the TopLevel class.
Reference: Section 7.10 on page 42

sbol-12102 ☑ The members property of a Collection is OPTIONAL and MAY contain a set of URIs.
Reference: Section 7.10 on page 42

sbol-12103 ▲ Each URI contained by the members property of a Collection MUST reference a TopLevel object.
Reference: Section 7.10 on page 42

doi:10.2390/biecoll-jib-2015-272

Rules for the Annotation class

sbol-12201 ☑ The `name` property of an `Annotation` is REQUIRED and MUST contain a `QName`.
Reference: Section 7.11 on page 43

sbol-12202 ☑ The `value` property of an `Annotation` is REQUIRED and MUST contain an `AnnotationValue`.
Reference: Section 7.11 on page 43

sbol-12203 ☑ An `AnnotationValue` MUST be a `literal` (a `String`, `Integer`, `Double`, or `Boolean`), `URI`, or a `NestedAnnotations` object.
Reference: Section 7.11 on page 43

sbol-12204 ☑ The `nestedQName` property of a `NestedAnnotations` object is REQUIRED and MUST contain a `QName`.
Reference: Section 7.11 on page 43

sbol-12205 ☑ The `nestedURI` property of a `NestedAnnotations` object is REQUIRED and MUST contain a `URI`.
Reference: Section 7.11 on page 43

sbol-12206 ☑ The `annotations` property of a `NestedAnnotations` object is OPTIONAL and MAY contain a set of `Annotation` objects.
Reference: Section 7.11 on page 43

Rules for the GenericTopLevel class

sbol-12301 ☑ A `GenericTopLevel` object MUST inherit all properties of the `TopLevel` class.
Reference: Section 7.11.2 on page 45

sbol-12302 ☑ The `rdfType` property of a `GenericTopLevel` object is REQUIRED and MUST contain a `QName`.
Reference: Section 7.11.2 on page 45

doi:10.2390/biecoll-jib-2015-272

Journal of Integrative Bioinformatics, 12(2):272, 2015 http://journal.imbio.de

B Examples of Serialization

B.1 Simple Examples

B.1.1 Serializing Sequence Objects

This example shows the serialization of a `Sequence`.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://parts.igem.org/Part:BBa_J23119:Design"/>
    <sbol:elements>ttgacagctagctcagtcctaggtataatgctagc</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
</rdf:RDF>
```

B.1.2 Serializing ComponentDefinition Objects

This example shows the serialization of a simple promoter `ComponentDefinition` and the `Sequence` to which it refers.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://partsregistry.org/Part:BBa_J23119"/>
    <dcterms:title>J23119 promoter</dcterms:title>
    <dcterms:description>Constitutive promoter</dcterms:description>
    <sbol:type rdf:resource="http://identifiers.org/chebi/CHEBI:4705"/>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000613"/>
    <sbol:sequence rdf:resource="http://partsregistry.org/seq/BBa_J23119"/>
  </sbol:ComponentDefinition>
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://parts.igem.org/Part:BBa_J23119:Design"/>
    <sbol:elements>ttgacagctagctcagtcctaggtataatgctagc</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
</rdf:RDF>
```

B.1.3 Serializing SequenceConstraint Objects

This example shows the serialization of `SequenceConstraint` between two `Component` objects in a composite promoter `ComponentDefinition`. In the example, the promoter `ComponentDefinition` has two sub-`Component`s that instantiate the `ComponentDefinition` objects for a core promoter region and and a binding site. The `SequenceConstraint` specifies that the core promoter region precedes the binding site.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_K174004">
```


doi:10.2390/biecoll-jib-2015-272

```
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_K174004"/>
    <sbol:displayId>BBa_K174004</sbol:displayId>
    <dcterms:title>pspac promoter</dcterms:title>
    <dcterms:description>LacI repressible promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
    <sbol:sequenceConstraint>
      <sbol:SequenceConstraint rdf:about="http://partsregistry.org/cd/BBa_K174004/r1">
        <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_K174004/r1"/>
        <sbol:displayId>r1</sbol:displayId>
        <sbol:restriction rdf:resource="http://sbols.org/v2#precedes"/>
        <sbol:subject rdf:resource="http://partsregistry.org/cd/pspac"/>
        <sbol:object rdf:resource="http://partsregistry.org/cd/LacI_operator"/>
      </sbol:SequenceConstraint>
    </sbol:sequenceConstraint>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/pspac">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/pspac"/>
    <sbol:displayId>pspac</sbol:displayId>
    <dcterms:title>constitutive promoter</dcterms:title>
    <dcterms:description>pspac core promoter region</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/LacI_operator">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/LacI_operator"/>
    <sbol:displayId>LacI_operator</sbol:displayId>
    <dcterms:title>LacI operator</dcterms:title>
    <dcterms:description>LacI binding site</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000057"/>
  </sbol:ComponentDefinition>
</rdf:RDF>
```

B.1.4 Serializing Cut Location Objects

This example shows the serialization of a `Cut Location`.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://partsregistry.org/Part:BBa_J23119"/>
    <dcterms:title>J23119 promoter</dcterms:title>
    <dcterms:description>Constitutive promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000613"/>
    <sbol:sequenceAnnotation>
      <sbol:SequenceAnnotation rdf:about="http://partsregistry.org/cd/BBa_J23119/cutat10">
        <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119/cutat10"/>
        <sbol:displayId>cutat10</sbol:displayId>
        <sbol:location>
          <sbol:Cut rdf:about="http://partsregistry.org/cd/BBa_J23119/cutat10/cut1">
            <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119/cutat10/cut1"/>
            <sbol:displayId>cut1</sbol:displayId>
            <sbol:at>10</sbol:at>
            <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
          </sbol:Cut>
        </sbol:location>
      </sbol:SequenceAnnotation>
    </sbol:sequenceAnnotation>
    <sbol:sequenceAnnotation>
      <sbol:SequenceAnnotation rdf:about="http://partsregistry.org/cd/BBa_J23119/cutat12">
        <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119/cutat12"/>
```


doi:10.2390/biecoll-jib-2015-272

```
        <sbol:displayId>cutat12</sbol:displayId>
        <sbol:location>
          <sbol:Cut rdf:about="http://partsregistry.org/cd/BBa_J23119/cutat12/cut2">
            <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119/cutat12/cut2"/>
            <sbol:displayId>cut2</sbol:displayId>
            <sbol:at>12</sbol:at>
            <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
          </sbol:Cut>
        </sbol:location>
      </sbol:SequenceAnnotation>
    </sbol:sequenceAnnotation>
    <sbol:sequence rdf:resource="http://partsregistry.org/seq/BBa_J23119"/>
  </sbol:ComponentDefinition>
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://parts.igem.org/Part:BBa_J23119:Design"/>
    <sbol:elements>ttgacagctagctcagtcctaggtataatgctagc</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
</rdf:RDF>
```

B.1.5 Serializing Model Objects

This example shows the serialization of a `Model`. In this example, the `Model` refers to an ODE model written in SBML that can be accessed the identified source repository.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:Model rdf:about="http://www.sbolstandard.org/examples/pIKE_Toggle_1">
    <sbol:persistentIdentity rdf:resource="http://www.sbolstandard.org/examples/pIKE_Toggle_1"/>
    <sbol:displayId>pIKE_Toggle_1</sbol:displayId>
    <dcterms:title>pIKE_Toggle_1 toggle switch</dcterms:title>
    <sbol:source rdf:resource="http://virtualparts.org/part/pIKE_Toggle_1"/>
    <sbol:language rdf:resource="http://identifiers.org/edam/format_2585"/>
    <sbol:framework rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000062"/>
  </sbol:Model>
</rdf:RDF>
```

B.1.6 Serializing ModuleDefinition Objects

This example shows the serialization of a simple `ModuleDefinition`. This `ModuleDefinition` includes an `Interaction` that represents the translation of a protein.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ModuleDefinition rdf:about="http://sbolstandard.org/example/md/GFP_expression">
    <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/md/GFP_expression"/>
    <sbol:displayId>GFP_expression</sbol:displayId>
    <sbol:functionalComponent>
      <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/md/GFP_expression/Constitutive_GFP">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/md/GFP_expression/Constitutive_GFP"/>
        <sbol:displayId>Constitutive_GFP</sbol:displayId>
        <sbol:definition rdf:resource="http://sbolstandard.org/example/GFP_generator"/>
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:direction rdf:resource="http://sbols.org/v2#in"/>
      </sbol:FunctionalComponent>
    </sbol:functionalComponent>
    <sbol:functionalComponent>
      <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/md/GFP_expression/GFP_protein">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/md/GFP_expression/GFP_protein"/>
        <sbol:displayId>GFP_protein</sbol:displayId>
```


doi:10.2390/biecoll-jib-2015-272

```
      <sbol:definition rdf:resource="http://sbolstandard.org/example/GFP"/>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:direction rdf:resource="http://sbols.org/v2#out"/>
    </sbol:FunctionalComponent>
  </sbol:functionalComponent>
  <sbol:interaction>
    <sbol:Interaction rdf:about="http://sbolstandard.org/example/md/GFP_expression/express_GFP">
      <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/md/GFP_expression/express_GFP"/>
      <sbol:displayId>express_GFP</sbol:displayId>
      <sbol:type rdf:resource="Transcription"/>
      <sbol:participation>
        <sbol:Participation rdf:about="http://sbolstandard.org/example/md/GFP_expression/express_GFP/Protein">
          <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/md/GFP_expression/express_GFP/Protein"/>
          <sbol:displayId>Protein</sbol:displayId>
          <sbol:participant rdf:resource="http://sbolstandard.org/example/md/GFP_expression/GFP_protein"/>
        </sbol:Participation>
      </sbol:participation>
      <sbol:participation>
        <sbol:Participation rdf:about="http://sbolstandard.org/example/md/GFP_expression/express_GFP/CDS">
          <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/md/GFP_expression/express_GFP/CDS"/>
          <sbol:displayId>CDS</sbol:displayId>
          <sbol:participant rdf:resource="http://sbolstandard.org/example/md/GFP_expression/Constitutive_GFP"/>
        </sbol:Participation>
      </sbol:participation>
    </sbol:Interaction>
  </sbol:interaction>
 </sbol:ModuleDefinition>
</rdf:RDF>
</rdf:RDF>
```

B.1.7 Serializing Application Specific Data Within SBOL Objects

This example shows the serialization of application-specific data from `Annotation` objects. A `ComponentDefinition` that represents a promoter is annotated with custom data on the promoter's sigma factor and how it is regulated.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:pr="http://partsregistry.org/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.
     org/dc/terms/" xmlns:prov="http://www.w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <pr:group>iGEM2006_Berkeley</pr:group>
    <pr:experience rdf:resource="http://parts.igem.org/cgi/partsdb/part_info.cgi?part_name=BBa_J23119"/>
    <pr:information>
      <pr:Information rdf:about="http://parts.igem.org/cgi/partsdb/part_info.cgi?part_name=BBa_J23119">
        <pr:sigmafactor>//rnap/prokaryote/ecoli/sigma70</pr:sigmafactor>
        <pr:regulation>//regulation/constitutive</pr:regulation>
      </pr:Information>
    </pr:information>
    <dcterms:title>J23119</dcterms:title>
    <dcterms:description>Constitutive promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
  </sbol:ComponentDefinition>
</rdf:RDF>
```

B.1.8 Serializing Application-Specific Data Outside SBOL Objects

This example shows the serialization of application-specific data from `GenericTopLevel` objects. Such data can be referenced by other SBOL objects via `Annotation` objects.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:myapp="http://www.myapp.org/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.
     org/dc/terms/" xmlns:prov="http://www.w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
```


doi:10.2390/biecoll-jib-2015-272

```
  <sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/cd/BBa_J23119">
    <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/cd/BBa_J23119"/>
    <sbol:displayId>BBa_J23119</sbol:displayId>
    <prov:wasDerivedFrom rdf:resource="http://www.partsregistry.org/Part:BBa_J23119"/>
    <myapp:datasheet rdf:resource="http://www.partsregistry.org/gen/datasheet1"/>
    <dcterms:title>J23119</dcterms:title>
    <dcterms:description>Constitutive promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
  </sbol:ComponentDefinition>
  <myapp:Datasheet rdf:about="http://www.partsregistry.org/gen/datasheet1">
    <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/gen/datasheet1"/>
    <sbol:displayId>datasheet1</sbol:displayId>
    <myapp:characterizationData rdf:resource="http://www.myapp.org/measurement/1"/>
    <myapp:transcriptionRate>1</myapp:transcriptionRate>
    <dcterms:title>Datasheet 1</dcterms:title>
  </myapp:Datasheet>
</rdf:RDF>
```

B.1.9 Serializing Collection Objects

This example shows the serialization of a `Collection`. This `Collection` represents a library of promoters.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
      w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:Collection rdf:about="http://parts.igem.org/Promoters/Catalog/Anderson">
    <sbol:persistentIdentity rdf:resource="http://parts.igem.org/Promoters/Catalog/Anderson"/>
    <sbol:displayId>Anderson</sbol:displayId>
    <dcterms:title>Anderson promoters</dcterms:title>
    <dcterms:description>The Anderson promoter collection</dcterms:description>
    <sbol:member rdf:resource="http://partsregistry.org/Part:BBa_J23119"/>
    <sbol:member rdf:resource="http://partsregistry.org/Part:BBa_J23118"/>
  </sbol:Collection>
</rdf:RDF>
```

B.2 Complex Examples

B.2.1 PoPS Receiver

This example shows the serialization of the PoPS Receiver device designed by Canton and co-workers Canton et al. (2008). In particular, this example includes the serialization of a `ComponentDefinition` that composes five `Component` objects to define the structure of a detector for the cell-cell signaling molecule $3OC_6HSL$. The five components are arranged in a sequence: first come four `Component` objects that together implement constitutive expression of the LuxR protein, which in turn responds to $3OC_6HSL$: a constitutive promoter, 5'UTR, coding sequence for LuxR, and terminator. Finally, after these objects comes the `Component` object for the pLuxR promoter, which is activated in the presence of LuxR and $3OC_6HSL$. Complete details of the device can be found in the cited paper and also at `http://parts.igem.org/Part:BBa_F2620`.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:pr="http://partsregistry.org" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.
      org/dc/terms/" xmlns:prov="http://www.w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_F2620">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620"/>
    <sbol:displayId>BBa_F2620</sbol:displayId>
    <dcterms:title>BBa_F2620</dcterms:title>
    <dcterms:description>3OC6HSL -&gt; PoPS Receiver</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:00001411"/>
    <sbol:component>
      <sbol:Component rdf:about="http://partsregistry.org/cd/BBa_F2620/pLuxR">
        <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/pLuxR"/>
```


doi:10.2390/biecoll-jib-2015-272

```
      <sbol:displayId>pLuxR</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://partsregistry.org/cd/BBa_R0062"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://partsregistry.org/cd/BBa_F2620/luxR">
      <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/luxR"/>
      <sbol:displayId>luxR</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://partsregistry.org/cd/BBa_C0062"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://partsregistry.org/cd/BBa_F2620/pTetR">
      <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/pTetR"/>
      <sbol:displayId>pTetR</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://partsregistry.org/cd/BBa_R0040"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://partsregistry.org/cd/BBa_F2620/ter">
      <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/ter"/>
      <sbol:displayId>ter</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://partsregistry.org/cd/BBa_B0015"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://partsregistry.org/cd/BBa_F2620/rbs">
      <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/rbs"/>
      <sbol:displayId>rbs</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://partsregistry.org/cd/BBa_B0034"/>
    </sbol:Component>
  </sbol:component>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://partsregistry.org/cd/BBa_F2620/anno3">
      <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno3"/>
      <sbol:displayId>anno3</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://partsregistry.org/cd/BBa_F2620/anno3/location3">
          <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno3/location3"/>
          <sbol:displayId>location3</sbol:displayId>
          <sbol:start>69</sbol:start>
          <sbol:end>770</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://partsregistry.org/cd/BBa_F2620/luxR"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://partsregistry.org/cd/BBa_F2620/anno5">
      <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno5"/>
      <sbol:displayId>anno5</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://partsregistry.org/cd/BBa_F2620/anno5/location5">
          <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno5/location5"/>
          <sbol:displayId>location5</sbol:displayId>
          <sbol:start>901</sbol:start>
          <sbol:end>956</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://partsregistry.org/cd/BBa_F2620/pLuxR"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
```


doi:10.2390/biecoll-jib-2015-272

```
    <sbol:sequenceAnnotation>
      <sbol:SequenceAnnotation rdf:about="http://partsregistry.org/cd/BBa_F2620/anno1">
        <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno1"/>
        <sbol:displayId>anno1</sbol:displayId>
        <sbol:location>
          <sbol:Range rdf:about="http://partsregistry.org/cd/BBa_F2620/anno1/location1">
            <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno1/location1"/>
            <sbol:displayId>location1</sbol:displayId>
            <sbol:start>1</sbol:start>
            <sbol:end>55</sbol:end>
            <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
          </sbol:Range>
        </sbol:location>
        <sbol:component rdf:resource="http://partsregistry.org/cd/BBa_F2620/pTetR"/>
      </sbol:SequenceAnnotation>
    </sbol:sequenceAnnotation>
    <sbol:sequenceAnnotation>
      <sbol:SequenceAnnotation rdf:about="http://partsregistry.org/cd/BBa_F2620/anno4">
        <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno4"/>
        <sbol:displayId>anno4</sbol:displayId>
        <sbol:location>
          <sbol:Range rdf:about="http://partsregistry.org/cd/BBa_F2620/anno4/location4">
            <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno4/location4"/>
            <sbol:displayId>location4</sbol:displayId>
            <sbol:start>771</sbol:start>
            <sbol:end>900</sbol:end>
            <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
          </sbol:Range>
        </sbol:location>
        <sbol:component rdf:resource="http://partsregistry.org/cd/BBa_F2620/ter"/>
      </sbol:SequenceAnnotation>
    </sbol:sequenceAnnotation>
    <sbol:sequenceAnnotation>
      <sbol:SequenceAnnotation rdf:about="http://partsregistry.org/cd/BBa_F2620/anno2">
        <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno2"/>
        <sbol:displayId>anno2</sbol:displayId>
        <sbol:location>
          <sbol:Range rdf:about="http://partsregistry.org/cd/BBa_F2620/anno2/location2">
            <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_F2620/anno2/location2"/>
            <sbol:displayId>location2</sbol:displayId>
            <sbol:start>56</sbol:start>
            <sbol:end>68</sbol:end>
            <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
          </sbol:Range>
        </sbol:location>
        <sbol:component rdf:resource="http://partsregistry.org/cd/BBa_F2620/rbs"/>
      </sbol:SequenceAnnotation>
    </sbol:sequenceAnnotation>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_R0062">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_R0062"/>
    <sbol:displayId>BBa_R0062</sbol:displayId>
    <dcterms:title>pLuxR</dcterms:title>
    <dcterms:description>LuxR inducible promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
    <sbol:sequence rdf:resource="http://partsregistry.org/seq/BBa_R0062"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_B0015">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_B0015"/>
    <sbol:displayId>BBa_B0015</sbol:displayId>
    <dcterms:title>BBa_B0015</dcterms:title>
    <dcterms:description>Double terminator</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000141"/>
    <sbol:sequence rdf:resource="http://partsregistry.org/seq/BBa_B0015"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_C0062">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_C0062"/>
```


doi:10.2390/biecoll-jib-2015-272

```
    <sbol:displayId>BBa_C0062</sbol:displayId>
    <dcterms:title>luxR</dcterms:title>
    <dcterms:description>luxR coding sequence</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000316"/>
    <sbol:sequence rdf:resource="http://partsregistry.org/seq/BBa_C0062"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_R0040">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_R0040"/>
    <sbol:displayId>BBa_R0040</sbol:displayId>
    <dcterms:title>pTetR</dcterms:title>
    <dcterms:description>TetR repressible promoter</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
    <sbol:sequence rdf:resource="http://partsregistry.org/seq/BBa_R0040"/>
  </sbol:ComponentDefinition>
  <sbol:ComponentDefinition rdf:about="http://partsregistry.org/cd/BBa_B0034">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/cd/BBa_B0034"/>
    <sbol:displayId>BBa_B0034</sbol:displayId>
    <dcterms:title>BBa_B0034</dcterms:title>
    <dcterms:description>RBS based on Elowitz repressilator</dcterms:description>
    <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
    <sbol:role rdf:resource="http://identifiers.org/so/SO:0000139"/>
    <sbol:sequence rdf:resource="http://partsregistry.org/seq/BBa_B0034"/>
  </sbol:ComponentDefinition>
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_B0034">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_B0034"/>
    <sbol:displayId>BBa_B0034</sbol:displayId>
    <sbol:elements>aaagaggagaaa</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_R0040">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_R0040"/>
    <sbol:displayId>BBa_R0040</sbol:displayId>
    <sbol:elements>tccctatcagtgatagagattgacatccctatcagtgatagagatactgagcac</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_C0062">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_C0062"/>
    <sbol:displayId>BBa_C0062</sbol:displayId>
    <sbol:elements>atgcttatctgatatgactaaaatggtacattgtgaatattatttactcgcgatcatttatcctcattcta
    tggttaaatctgatatttcaatcctagataattaccctaaaaaatggaggcaatattatgatgacgctaatttaataaaatatgat
    cctatagtagattattctaactccaatcattcaccaattaattggaatatatttgaaaacaatgctgtaaataaaaaatctccaaa
    tgtaattaaagaagcgaaaacatcaggtcttatcactgggtttagtttccctattcatacggctaacaatggcttcggaatgctta
    gttttgcacattcagaaaaagacaactatatagatagtttatttttacatgcgtgtatgaacataccattaattgttccttctcta
    gttgataattatcgaaaaataaatatagcaaataataaatcaaacaacgatttaaccaaaagagaaaaagaatgtttagcgtgggc
    atgcgaaggaaaaagctcttgggatatttcaaaaatattaggttgcagtgagcgtactgtcactttccatttaaccaatgcgcaaa
    tgaaactcaatacaacaaaccgctgccaaagtatttctaaagcaattttaacaggagcaattgattgcccatactttaaaaattaa
    taacactgatagtgctagtgtagatcac</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_R0062">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_R0062"/>
    <sbol:displayId>BBa_R0062</sbol:displayId>
    <sbol:elements>acctgtaggatcgtacaggtttacgcaagaaaatggtttgttatagtcgaataaa</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://partsregistry.org/seq/BBa_B0015">
    <sbol:persistentIdentity rdf:resource="http://partsregistry.org/seq/BBa_B0015"/>
    <sbol:displayId>BBa_B0015</sbol:displayId>
    <sbol:elements>ccaggcatcaaataaaacgaaaggctcagtcgaaagactgggcctttcgttttatctgttgtttgtcggtg
    aacgctctctactagagtcacactggctcaccttcgggtgggcctttctgcgtttata</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
</rdf:RDF>
```


doi:10.2390/biecoll-jib-2015-272

B.2.2 Toggle Switch

This example shows the serialization of the ComponentDefinition and ModuleDefinition objects for a LacI/TetR toggle switch similar to those constructed in Gardner et al. (2000). This design is essentially similar to the one presented in Section 9, except that it uses some alternate groupings in how the total design is built up out of smaller entities.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dcterms="http://purl.org/dc/terms/" xmlns:prov="http://www.
     w3.org/ns/prov#" xmlns:sbol="http://sbols.org/v2#">
  <sbol:ModuleDefinition rdf:about="http://sbolstandard.org/example/toggle_switch">
    <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toggle_switch"/>
    <sbol:displayId>toggle_switch</sbol:displayId>
    <sbol:role rdf:resource="http://sbolstandard.org/example/module_role/toggle_switch"/>
    <sbol:functionalComponent>
      <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/toggle_switch/TetR">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toggle_switch/TetR"/>
        <sbol:displayId>TetR</sbol:displayId>
        <sbol:definition rdf:resource="http://identifiers.org/uniprot/Q6QR72"/>
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:direction rdf:resource="http://sbols.org/v2#inout"/>
      </sbol:FunctionalComponent>
    </sbol:functionalComponent>
    <sbol:functionalComponent>
      <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/toggle_switch/LacI">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toggle_switch/LacI"/>
        <sbol:displayId>LacI</sbol:displayId>
        <sbol:definition rdf:resource="http://identifiers.org/uniprot/P03023"/>
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:direction rdf:resource="http://sbols.org/v2#inout"/>
      </sbol:FunctionalComponent>
    </sbol:functionalComponent>
    <sbol:model rdf:resource="http://sbolstandard.org/example/toogleswitch"/>
    <sbol:module>
      <sbol:Module rdf:about="http://sbolstandard.org/example/toggle_switch/tetr_inverter">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toggle_switch/tetr_inverter"/>
        <sbol:displayId>tetr_inverter</sbol:displayId>
        <sbol:definition rdf:resource="http://sbolstandard.org/example/tetr_inverter"/>
        <sbol:mapsTo>
          <sbol:MapsTo rdf:about="http://sbolstandard.org/example/toggle_switch/tetr_inverter/TetR_mapping">
            <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toggle_switch/tetr_inverter/TetR_mapping"/>
            <sbol:displayId>TetR_mapping</sbol:displayId>
            <sbol:refinement rdf:resource="http://sbols.org/v2#useRemote"/>
            <sbol:remote rdf:resource="http://sbolstandard.org/example/tetr_inverter/TF"/>
            <sbol:local rdf:resource="http://sbolstandard.org/example/toggle_switch/TetR"/>
          </sbol:MapsTo>
        </sbol:mapsTo>
      </sbol:Module>
    </sbol:module>
    <sbol:module>
      <sbol:Module rdf:about="http://sbolstandard.org/example/toggle_switch/laci_inverter">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toggle_switch/laci_inverter"/>
        <sbol:displayId>laci_inverter</sbol:displayId>
        <sbol:definition rdf:resource="http://sbolstandard.org/example/laci_inverter"/>
        <sbol:mapsTo>
          <sbol:MapsTo rdf:about="http://sbolstandard.org/example/toggle_switch/laci_inverter/LacI_mapping">
            <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toggle_switch/laci_inverter/LacI_mapping"/>
            <sbol:displayId>LacI_mapping</sbol:displayId>
            <sbol:refinement rdf:resource="http://sbols.org/v2#useRemote"/>
            <sbol:remote rdf:resource="http://sbolstandard.org/example/laci_inverter/TF"/>
            <sbol:local rdf:resource="http://sbolstandard.org/example/toggle_switch/LacI"/>
          </sbol:MapsTo>
        </sbol:mapsTo>
      </sbol:Module>
    </sbol:module>
  </sbol:ModuleDefinition>
  <sbol:ModuleDefinition rdf:about="http://sbolstandard.org/example/laci_inverter">
```


doi:10.2390/biecoll-jib-2015-272

```
    <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/laci_inverter"/>
    <sbol:displayId>laci_inverter</sbol:displayId>
    <sbol:role rdf:resource="http://parts.igem.org/cgi/partsdb/pgroup.cgi?pgroup=inverter"/>
    <sbol:functionalComponent>
      <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/laci_inverter/TF">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/laci_inverter/TF"/>
        <sbol:displayId>TF</sbol:displayId>
        <sbol:definition rdf:resource="http://identifiers.org/uniprot/P03023"/>
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:direction rdf:resource="http://sbols.org/v2#inout"/>
      </sbol:FunctionalComponent>
    </sbol:functionalComponent>
    <sbol:functionalComponent>
      <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/laci_inverter/promoter">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/laci_inverter/promoter"/>
        <sbol:displayId>promoter</sbol:displayId>
        <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_R0010"/>
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:direction rdf:resource="http://sbols.org/v2#inout"/>
      </sbol:FunctionalComponent>
    </sbol:functionalComponent>
    <sbol:interaction>
      <sbol:Interaction rdf:about="http://sbolstandard.org/example/laci_inverter/LacI_pLacI">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/laci_inverter/LacI_pLacI"/>
        <sbol:displayId>LacI_pLacI</sbol:displayId>
        <sbol:type rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000169"/>
        <sbol:participation>
          <sbol:Participation rdf:about="http://sbolstandard.org/example/laci_inverter/LacI_pLacI/BBa_R0010">
            <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/laci_inverter/LacI_pLacI/BBa_R0010"/>
            <sbol:displayId>BBa_R0010</sbol:displayId>
            <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000598"/>
            <sbol:participant rdf:resource="http://sbolstandard.org/example/laci_inverter/promoter"/>
          </sbol:Participation>
        </sbol:participation>
        <sbol:participation>
          <sbol:Participation rdf:about="http://sbolstandard.org/example/laci_inverter/LacI_pLacI/P03023">
            <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/laci_inverter/LacI_pLacI/P03023"/>
            <sbol:displayId>P03023</sbol:displayId>
            <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000020"/>
            <sbol:participant rdf:resource="http://sbolstandard.org/example/laci_inverter/TF"/>
          </sbol:Participation>
        </sbol:participation>
      </sbol:Interaction>
    </sbol:interaction>
  </sbol:ModuleDefinition>
  <sbol:ModuleDefinition rdf:about="http://sbolstandard.org/example/tetr_inverter">
    <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/tetr_inverter"/>
    <sbol:displayId>tetr_inverter</sbol:displayId>
    <sbol:role rdf:resource="http://parts.igem.org/cgi/partsdb/pgroup.cgi?pgroup=inverter"/>
    <sbol:functionalComponent>
      <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/tetr_inverter/TF">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/tetr_inverter/TF"/>
        <sbol:displayId>TF</sbol:displayId>
        <sbol:definition rdf:resource="http://identifiers.org/uniprot/Q6QR72"/>
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:direction rdf:resource="http://sbols.org/v2#inout"/>
      </sbol:FunctionalComponent>
    </sbol:functionalComponent>
    <sbol:functionalComponent>
      <sbol:FunctionalComponent rdf:about="http://sbolstandard.org/example/tetr_inverter/promoter">
        <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/tetr_inverter/promoter"/>
        <sbol:displayId>promoter</sbol:displayId>
        <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_R0040"/>
        <sbol:access rdf:resource="http://sbols.org/v2#public"/>
        <sbol:direction rdf:resource="http://sbols.org/v2#inout"/>
      </sbol:FunctionalComponent>
    </sbol:functionalComponent>
    <sbol:interaction>
      <sbol:Interaction rdf:about="http://sbolstandard.org/example/tetr_inverter/LacI_pLacI">
```


doi:10.2390/biecoll-jib-2015-272

```
      <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/tetr_inverter/LacI_pLacI"/>
      <sbol:displayId>LacI_pLacI</sbol:displayId>
      <sbol:type rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000169"/>
      <sbol:participation>
        <sbol:Participation rdf:about="http://sbolstandard.org/example/tetr_inverter/LacI_pLacI/Q6QR72">
          <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/tetr_inverter/LacI_pLacI/Q6QR72"/>
          <sbol:displayId>Q6QR72</sbol:displayId>
          <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000020"/>
          <sbol:participant rdf:resource="http://sbolstandard.org/example/tetr_inverter/TF"/>
        </sbol:Participation>
      </sbol:participation>
      <sbol:participation>
        <sbol:Participation rdf:about="http://sbolstandard.org/example/tetr_inverter/LacI_pLacI/BBa_R0040">
          <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/tetr_inverter/LacI_pLacI/BBa_R0040"/>
          <sbol:displayId>BBa_R0040</sbol:displayId>
          <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000598"/>
          <sbol:participant rdf:resource="http://sbolstandard.org/example/tetr_inverter/promoter"/>
        </sbol:Participation>
      </sbol:participation>
    </sbol:Interaction>
  </sbol:interaction>
</sbol:ModuleDefinition>
<sbol:Model rdf:about="http://sbolstandard.org/example/toogleswitch">
  <sbol:persistentIdentity rdf:resource="http://sbolstandard.org/example/toogleswitch"/>
  <sbol:displayId>toogleswitch</sbol:displayId>
  <sbol:source rdf:resource="http://virtualparts.org/part/pIKE_Toggle_1"/>
  <sbol:language rdf:resource="http://identifiers.org/edam/format_2585"/>
  <sbol:framework rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000062"/>
</sbol:Model>
<sbol:ComponentDefinition rdf:about="http://www.virtualparts.org/part/pIKE_Toggle_1">
  <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1"/>
  <sbol:displayId>pIKE_Toggle_1</sbol:displayId>
  <dcterms:title>LacI/TetR Toggle Switch</dcterms:title>
  <dcterms:description>LacI/TetR Toggle Switch</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000280"/>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKE_Toggle_1/pIKERightCassette_1">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1/pIKERightCassette_1"/>
      <sbol:displayId>pIKERightCassette_1</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKE_Toggle_1/pIKELeftCassette_1">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1/pIKELeftCassette_1"/>
      <sbol:displayId>pIKELeftCassette_1</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1"/>
    </sbol:Component>
  </sbol:component>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKE_Toggle_1/anno2">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1/anno2"/>
      <sbol:displayId>anno2</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKE_Toggle_1/anno2/location2">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1/anno2/location2"/>
          <sbol:displayId>location2</sbol:displayId>
          <sbol:start>1286</sbol:start>
          <sbol:end>2834</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1/pIKERightCassette_1"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
```


doi:10.2390/biecoll-jib-2015-272

```
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKE_Toggle_1/anno1">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1/anno1"/>
      <sbol:displayId>anno1</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKE_Toggle_1/anno1/location1">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1/anno1/location1"/>
          <sbol:displayId>location1</sbol:displayId>
          <sbol:start>1</sbol:start>
          <sbol:end>1285</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKE_Toggle_1/pIKELeftCassette_1"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/BBa_J61130">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/BBa_J61130"/>
  <sbol:displayId>BBa_J61130</sbol:displayId>
  <dcterms:title>BBa_J61101 RBS</dcterms:title>
  <dcterms:description>RBS2</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000139"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/BBa_J61130"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/BBa_C0012">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/BBa_C0012"/>
  <sbol:displayId>BBa_C0012</sbol:displayId>
  <dcterms:title>lacI</dcterms:title>
  <dcterms:description>lacI coding sequence</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000316"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/BBa_C0012"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/ECK120033736">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/ECK120033736"/>
  <sbol:displayId>ECK120033736</sbol:displayId>
  <dcterms:title>ECK120033736</dcterms:title>
  <dcterms:description>Terminator2</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000141"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/ECK120033736"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/BBa_R0040">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/BBa_R0040"/>
  <sbol:displayId>BBa_R0040</sbol:displayId>
  <dcterms:title>pTetR</dcterms:title>
  <dcterms:description>pTet promoter</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/BBa_R0040"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://identifiers.org/uniprot/Q6QR72">
  <sbol:persistentIdentity rdf:resource="http://identifiers.org/uniprot/Q6QR72"/>
  <sbol:displayId>Q6QR72</sbol:displayId>
  <dcterms:title>TetR</dcterms:title>
  <dcterms:description>TetR protein</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#Protein"/>
  <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000020"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://identifiers.org/uniprot/P03023">
  <sbol:persistentIdentity rdf:resource="http://identifiers.org/uniprot/P03023"/>
  <sbol:displayId>P03023</sbol:displayId>
  <dcterms:title>LacI</dcterms:title>
  <dcterms:description>LacI protein</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#Protein"/>
  <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000020"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/BBa_J61120">
```


doi:10.2390/biecoll-jib-2015-272

```
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/BBa_J61120"/>
  <sbol:displayId>BBa_J61120</sbol:displayId>
  <dcterms:title>BBa_J61101 RBS</dcterms:title>
  <dcterms:description>RBS2</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000139"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/BBa_J61120"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/BBa_E0040">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/BBa_E0040"/>
  <sbol:displayId>BBa_E0040</sbol:displayId>
  <dcterms:title>gfp</dcterms:title>
  <dcterms:description>gfp coding sequence</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000316"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/BBa_E0040"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/ECK120029600">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/ECK120029600"/>
  <sbol:displayId>ECK120029600</sbol:displayId>
  <dcterms:title>ECK120029600</dcterms:title>
  <dcterms:description>Terminator1</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000141"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/ECK120029600"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1">
  <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1"/>
  <sbol:displayId>pIKELeftCassette_1</sbol:displayId>
  <dcterms:title>TetR Inverter</dcterms:title>
  <dcterms:description>TetR Inverter</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000280"/>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/ECK120029600">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/ECK120029600"/>
      <sbol:displayId>ECK120029600</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/ECK120029600"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_R0040">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_R0040"/>
      <sbol:displayId>BBa_R0040</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_R0040"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_C0012">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_C0012"/>
      <sbol:displayId>BBa_C0012</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_C0012"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_J61101">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_J61101"/>
      <sbol:displayId>BBa_J61101</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_J61101"/>
    </sbol:Component>
  </sbol:component>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/anno4">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/anno4"/>
      <sbol:displayId>anno4</sbol:displayId>
```


doi:10.2390/biecoll-jib-2015-272

```
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/anno4/location4">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/anno4/location4"/>
          <sbol:displayId>location4</sbol:displayId>
          <sbol:start>1198</sbol:start>
          <sbol:end>1288</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/ECK120029600"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/anno2">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/anno2"/>
      <sbol:displayId>anno2</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/anno2/location2">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/anno2/location2"/>
          <sbol:displayId>location2</sbol:displayId>
          <sbol:start>56</sbol:start>
          <sbol:end>68</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_J61101"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/anno1">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/anno1"/>
      <sbol:displayId>anno1</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/anno1/location1">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/anno1/location1"/>
          <sbol:displayId>location1</sbol:displayId>
          <sbol:start>1</sbol:start>
          <sbol:end>55</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_R0040"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/anno3">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/anno3"/>
      <sbol:displayId>anno3</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKELeftCassette_1/anno3/location3">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/anno3/location3"/>
          <sbol:displayId>location3</sbol:displayId>
          <sbol:start>69</sbol:start>
          <sbol:end>1197</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKELeftCassette_1/BBa_C0012"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/BBa_J61101">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/BBa_J61101"/>
  <sbol:displayId>BBa_J61101</sbol:displayId>
  <dcterms:title>BBa_J61101 RBS</dcterms:title>
  <dcterms:description>RBS1</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000139"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/BBa_J61101"/>
```


doi:10.2390/biecoll-jib-2015-272

```
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/BBa_R0010">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/BBa_R0010"/>
  <sbol:displayId>BBa_R0010</sbol:displayId>
  <dcterms:title>pLacI</dcterms:title>
  <dcterms:description>pLacI promoter</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000167"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/BBa_R0010"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://identifiers.org/uniprot/P42212">
  <sbol:persistentIdentity rdf:resource="http://identifiers.org/uniprot/P42212"/>
  <sbol:displayId>P42212</sbol:displayId>
  <dcterms:title>GFP</dcterms:title>
  <dcterms:description>GFP protein</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#Protein"/>
  <sbol:role rdf:resource="http://identifiers.org/biomodels.sbo/SBO:0000011"/>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1">
  <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1"/>
  <sbol:displayId>pIKERightCassette_1</sbol:displayId>
  <dcterms:title>LacI Inverter</dcterms:title>
  <dcterms:description>LacI Inverter</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000280"/>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_R0010">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_R0010"/>
      <sbol:displayId>BBa_R0010</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_R0010"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_C0040">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_C0040"/>
      <sbol:displayId>BBa_C0040</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_C0040"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_J61130">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_J61130"/>
      <sbol:displayId>BBa_J61130</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_J61130"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_E0040">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_E0040"/>
      <sbol:displayId>BBa_E0040</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_E0040"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/ECK120033736">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/ECK120033736"/>
      <sbol:displayId>ECK120033736</sbol:displayId>
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/ECK120033736"/>
    </sbol:Component>
  </sbol:component>
  <sbol:component>
    <sbol:Component rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_J61120">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_J61120"/>
      <sbol:displayId>BBa_J61120</sbol:displayId>
```


doi:10.2390/biecoll-jib-2015-272

```
      <sbol:access rdf:resource="http://sbols.org/v2#public"/>
      <sbol:definition rdf:resource="http://www.partsregistry.org/BBa_J61120"/>
    </sbol:Component>
  </sbol:component>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno1">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno1"/>
      <sbol:displayId>anno1</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno1/location1">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno1/location1"/>
          <sbol:displayId>location1</sbol:displayId>
          <sbol:start>1</sbol:start>
          <sbol:end>55</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_R0010"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno5">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno5"/>
      <sbol:displayId>anno5</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno5/location5">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno5/location5"/>
          <sbol:displayId>location5</sbol:displayId>
          <sbol:start>743</sbol:start>
          <sbol:end>1463</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_E0040"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno6">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno6"/>
      <sbol:displayId>anno6</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno6/location6">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno6/location6"/>
          <sbol:displayId>location6</sbol:displayId>
          <sbol:start>1464</sbol:start>
          <sbol:end>1554</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/ECK120033736"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno3">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno3"/>
      <sbol:displayId>anno3</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno3/location3">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno3/location3"/>
          <sbol:displayId>location3</sbol:displayId>
          <sbol:start>69</sbol:start>
          <sbol:end>729</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_C0040"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
```


doi:10.2390/biecoll-jib-2015-272

```
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno4">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno4"/>
      <sbol:displayId>anno4</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno4/location4">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno4/location4"/>
          <sbol:displayId>location4</sbol:displayId>
          <sbol:start>730</sbol:start>
          <sbol:end>742</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_J61130"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
  <sbol:sequenceAnnotation>
    <sbol:SequenceAnnotation rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno2">
      <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno2"/>
      <sbol:displayId>anno2</sbol:displayId>
      <sbol:location>
        <sbol:Range rdf:about="http://www.virtualparts.org/part/pIKERightCassette_1/anno2/location2">
          <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/anno2/location2"/>
          <sbol:displayId>location2</sbol:displayId>
          <sbol:start>56</sbol:start>
          <sbol:end>68</sbol:end>
          <sbol:orientation rdf:resource="http://sbols.org/v2#inline"/>
        </sbol:Range>
      </sbol:location>
      <sbol:component rdf:resource="http://www.virtualparts.org/part/pIKERightCassette_1/BBa_J61120"/>
    </sbol:SequenceAnnotation>
  </sbol:sequenceAnnotation>
</sbol:ComponentDefinition>
<sbol:ComponentDefinition rdf:about="http://www.partsregistry.org/BBa_C0040">
  <sbol:persistentIdentity rdf:resource="http://www.partsregistry.org/BBa_C0040"/>
  <sbol:displayId>BBa_C0040</sbol:displayId>
  <dcterms:title>tetR</dcterms:title>
  <dcterms:description>tetR coding sequence</dcterms:description>
  <sbol:type rdf:resource="http://www.biopax.org/release/biopax-level3.owl#DnaRegion"/>
  <sbol:role rdf:resource="http://identifiers.org/so/SO:0000316"/>
  <sbol:sequence rdf:resource="http://www.virtualparts.org/part/BBa_C0040"/>
</sbol:ComponentDefinition>
<sbol:Sequence rdf:about="http://www.virtualparts.org/part/BBa_J61101">
  <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/BBa_J61101"/>
  <sbol:displayId>BBa_J61101</sbol:displayId>
  <sbol:elements>aaagacaggacc</sbol:elements>
  <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
</sbol:Sequence>
<sbol:Sequence rdf:about="http://www.virtualparts.org/part/BBa_J61120">
  <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/BBa_J61120"/>
  <sbol:displayId>BBa_J61120</sbol:displayId>
  <sbol:elements>aaagacaggacc</sbol:elements>
  <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
</sbol:Sequence>
<sbol:Sequence rdf:about="http://www.virtualparts.org/part/BBa_E0040">
  <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/BBa_E0040"/>
  <sbol:displayId>BBa_E0040</sbol:displayId>
  <sbol:elements>atgcgtaaaggagaagaacttttcactggagttgtcccaattcttgttgaattagatggtgatgttaatgggcac
  aaattttctgtcagtggagagggtgaaggtgatgcaacatacggaaaacttacccttaaatttatttgcactactggaaaactacctgtt
  ccatggccaacacttgtcactactttcggttatggtgttcaatgctttgcgagatacccagatcatatgaaacagcatgactttttcaag
  agtgccatgcccgaaggttatgtacaggaaagaactatatttttcaaagatgacgggaactacaagacacgtgctgaagtcaagtttgaa
  ggtgatacccttgttaatagaatcgagttaaaaggtattgattttaaagaagatggaaacattcttggacacaaattggaatacaactat
  aactcacacaatgtatacatcatggcagacaaacaaaagaatggaatcaaagttaacttcaaaattagacacaacattgaagatggaagc
  gttcaactagcagaccattatcaacaaaatactccaattggcgatggccctgtccttttaccagacaaccattacctgtccacacaatct
  gccctttcgaaagatcccaacgaaaagagagaccacatggtccttcttgagtttgtaacagctgctgggattacacatggcatggatgaa
  ctatacaaataataa</sbol:elements>
  <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
</sbol:Sequence>
<sbol:Sequence rdf:about="http://www.virtualparts.org/part/ECK120033736">
  <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/ECK120033736"/>
```


doi:10.2390/biecoll-jib-2015-272

```
    <sbol:displayId>ECK120033736</sbol:displayId>
    <sbol:elements>ttcagccaaaaaacttaagaccgccggtcttgtccactaccttgcagtaatgcggtggacaggatcggcggtttt
    cttttctcttctcaa</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.virtualparts.org/part/BBa_R0010">
    <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/BBa_R0010"/>
    <sbol:displayId>BBa_R0010</sbol:displayId>
    <sbol:elements>tccctatcagtgatagagattgacatccctatcagtgatagagatactgagcac</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.virtualparts.org/part/BBa_R0040">
    <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/BBa_R0040"/>
    <sbol:displayId>BBa_R0040</sbol:displayId>
    <sbol:elements>tccctatcagtgatagagattgacatccctatcagtgatagagatactgagcac</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.virtualparts.org/part/BBa_J61130">
    <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/BBa_J61130"/>
    <sbol:displayId>BBa_J61130</sbol:displayId>
    <sbol:elements>aaagaaacgaca</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.virtualparts.org/part/BBa_C0040">
    <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/BBa_C0040"/>
    <sbol:displayId>BBa_C0040</sbol:displayId>
    <sbol:elements>atgtccagattagataaaagtaaagtgattaacagcgcattagagctgcttaatgaggtcggaatcgaaggttta
    acaacccgtaaactcgcccagaagctaggtgtagagcagcctacattgtattggcatgtaaaaaataagcgggctttgctcgacgcctta
    gccattgagatgttagataggcaccatactcacttttgccctttagaaggggaaagctggcaagatttttacgtaataacgctaaaagt
    tttagatgtgctttactaagtcatcgcgatggagcaaaagtacatttaggtacacggcctacagaaaaacagtatgaaactctcgaaaat
    caattagccttttttatgccaacaaggtttttcactagagaatgcattatatgcactcagcgctgtggggcattttacttttaggttgcgta
    ttggaagatcaagagcatcaagtcgctaaagaagaaaggggaaacacctactactgatagtatgccgccattattacgacaagctatcgaa
    ttatttgatcaccaaggtgcagagccagcctttcttattcggccttgaattgatcatatgcggattagaaaaacaacttaaatgtgaaagt
    gggtccgctgcaaacgacgaaaactacgctttagtagcttaataa</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.virtualparts.org/part/BBa_C0012">
    <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/BBa_C0012"/>
    <sbol:displayId>BBa_C0012</sbol:displayId>
    <sbol:elements>atggtgaatgtgaaaccagtaacgttatacgatgtcgcagagtatgccggtgtctcttatcagaccgtttcccgc
    gtggtgaaccaggccagccacgtttctgcgaaaacgcgggaaaaagtggaagcggcgatggcggagctgaattacattcccaaccgcgtg
    gcacaacaactggcgggcaaacagtcgttgctgattggcgttgccacctccagtctggccctgcacgcgccgtcgcaaattgtcgcggcg
    attaaatctcgcgccgatcaactgggtgccagcgtggtggtgtcgatggtagaacgaagcggcgtcgaagcctgtaaagcggcggtgcac
    aatcttctcgcgcaacgcgtcagtgggctgatcattaactatccgctggatgaccaggatgccattgctgtggaagctgcctgcactaat
    gttccggcgttatttcttgatgtctctgaccagacacccatcaacagtattatttttctcccatgaagacggtacgcgactgggcgtggag
    catctggtcgcattgggtcaccagcaaatcgcgctgttagcgggcccattaagttctgtctcggcgcgtctgcgtctggctggctggcat
    aaatatctcactcgcaatcaaattcagccgatagcggaacgggaagggcgactggagtgccatgtccggttttcaacaaaccatgcaaatg
    ctgaatgagggcatcgttcccactgcgatgctggttgccaacgatcagatggcgctgggcgcaatgcgcgccattaccgagtccgggctg
    cgcgttggtgcggatatctcggtagtgggatacgacgataccgaagacagctcatgttatatcccgccgttaacccaccatcaaacaggat
    tttcgcctgctggggcaaaccagcgtggaccgcttgctgcaactctctcagggccaggcggtgaagggcaatcagctgttgcccgtctca
    ctggtgaaaagaaaaaccaccctggcgcccaatacgcaaaccgcctctccccgcgcgttggccgattcattaatgcagctggcacgacag
    gtttcccgactggaaagcgggcaggctgcaaacgacgaaaactacgctttagtagcttaataa</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
  <sbol:Sequence rdf:about="http://www.virtualparts.org/part/ECK120029600">
    <sbol:persistentIdentity rdf:resource="http://www.virtualparts.org/part/ECK120029600"/>
    <sbol:displayId>ECK120029600</sbol:displayId>
    <sbol:elements>ttcagccaaaaaacttaagaccgccggtcttgtccactaccttgcagtaatgcggtggacaggatcggcggtttt
    cttttctcttctcaa</sbol:elements>
    <sbol:encoding rdf:resource="http://www.chem.qmul.ac.uk/iubmb/misc/naseq.html"/>
  </sbol:Sequence>
</rdf:RDF>
```


doi:10.2390/biecoll-jib-2015-272

Appendix B: Publications

Bartley BA, Choi K, Sauro HM. Engineering Living Systems from Biophysical Principles, Biophysical Journal, Volume 112, Issue 6, p1050–1058, 28 March 2017

Der BS, Glassey E, **Bartley BA**, Enghuus C, Goodman DB, Gordon DB, Voigt CA, Gorochowski TE. DNAplotlib: programmable visualization of genetic designs and associated data. ACS synthetic biology. 2016 Oct 17.

Roehner N, Beal J, Clancy K, **Bartley B**, Misirli G, Grunberg R, Oberortner E, Pocock M, Bissell M, Madsen C, Nguyen T, Zhang M, Zhang Z, Zundel Z, Densmore D, Gennari J, Wipat A, Sauro H, Myers C. Sharing structure and function in biological design with SBOL 2.0, ACS Synthetic Biology, vol. 5, no. 6, pp. 498-506, Apr. 2016.

Bartley B, Beal J, Clancy K, Hillson N, Misirli G, Roehner N, Sauro H, Oberortner E, Madsen C, Pocock M, Wipat A, Nguyen T, Zhang Z, Myers C, Gennari J, Bissell M. Synthetic Biology Open Language (SBOL) Version 2.0.0. Journal of Integrative Bioinformatics, vol. 12, no. 2, pp. 272, Sep. 2015.

Kyung HK, Choi K, **Bartley B**, Sauro, HM. Controlling E. coli Gene Expression Noise. TBioCAS. 2015 Sep 9(4):495-504.

Copeland WB, **Bartley BA**, Chandran D, Galdzicki M, Sleight SC, Kim KH, Sauro HM. Computational Tools for Metabolic Engineering. Metabolic Engineering special issue on synthetic biology. Accepted: 2012 Mar 2.

Sleight SC, **Bartley BA**, Lieviant JA, Sauro HM. Designing and engineering evolutionary robust genetic circuits. J Biol Eng. 2010 Nov 1;4:12.

Sleight SC, **Bartley BA**, Lieviant JA, Sauro HM. In-Fusion BioBrick assembly and re-engineering. Nucleic Acids Res. 2010 May;38(8):2624-36.