Mapping Genotype to Phenotype using Attribute Grammar

Laura Adam

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Genetics, Bioinformatics and Computational Biology

Jean Peccoud, Chair
David R. Bevan
Harold R. Garner
François Képès
Narendran Ramakrishnan
John J. Tyson

July 25, 2013
Blacksburg, Virginia

Keywords: Synthetic Biology, Genotype, Phenotype, Formal Language, Attribute Grammar, Compilation, Compiler Generation, Prolog, SBML, GenoCAD

Mapping Genotype to Phenotype using Attribute Grammar

Laura Adam

ABSTRACT

Over the past 10 years, several synthetic biology research groups have proposed tools and domain-specific languages to help with the design of artificial DNA molecules. Community standards for exchanging data between these tools, such as the Synthetic Biology Open Language (SBOL), have been developed. It is increasingly important to be able to perform *in silico* simulation before the time and cost consuming wet lab realization of the constructs, which, as technology advances, also become in themselves more complex. By extending the concept of describing genetic expression as a language, we propose to model relations between genotype and phenotype using formal language theory.

We use attribute grammars (AGs) to extract context-dependent information from genetic constructs and compile them into mathematical models, possibly giving clues about their phenotypes. They may be used as a backbone for biological Domain-Specific Languages (DSLs) and we developed a methodology to design these AG based DSLs. We gave examples of languages in the field of synthetic biology to model genetic regulatory networks with Ordinary Differential Equations (ODEs) based on various rate laws or with discrete boolean network models.

We implemented a demonstration of these concepts in GenoCAD, a Computer Assisted Design (CAD) software for synthetic biology. GenoCAD guides users from design to simulation. Users can either design constructs with the attribute grammars provided or define their own project-specific languages. Outputting the mathematical model of a genetic construct is performed by DNA compilation based on the attribute grammar specified; the design of new languages by users necessitated the generation on-the-fly of such attribute grammar based DNA compilers.

We also considered the impact of our research and its potential dual-use issues. Indeed, after the design exploration is performed *in silico*, the next logical step is to synthesize the designed construct's DNA molecule to build the construct *in vivo*. We implemented an algorithm to identify sequences of concern of any length that are specific to Select Agents and Toxins, helping to ensure safer use of our methods.

This work was supported by National Science Foundation Award no. 1060776 to Virginia Tech and a graduate fellowship from the Science Applications International Corporation (SAIC).

Dedication

This incredible scientific and life journey in Blacksburg, Virginia, would not have been possible without:

- My advisor, Jean Peccoud, who, I would like to thank for offering me the opportunity to pursue a PhD in his research group, and especially, for being extremely very supportive and understanding over the past four years.
- My family, who has always been present, supporting and encouraging, for creating a worry-free environment so I have always been able to pursue my dreams.
- My fabulous friends -old and new-, for making life so exciting wherever I am.

Acknowledgments

First of all, I would like to acknowledge my thesis supervisor, Dr. Jean Peccoud; his guidance and support during these past years have been amazing. I am grateful for his patience, understanding, advice and support. He certainly provided me with amazing opportunities to find my professional path.
This thesis has been possible thanks to my colleagues in the synthetic biology group. In particular, implementations in GenoCAD have been made by Mandy Wilson and Russell Hertzberg. Mandy is maintaining GenoCAD and adding any new features. I would like to acknowledge Matt Lux, my fellow PhD student and friend, for all the fruitful discussions we had about research projects and academic life.
Applications of my research have been inspired by Dr. John Tyson's class and his group, in particular, Dr. Tyson himself, Katherine Chen and Neil Adames (from the synthetic biology group) for the Cell cycle Grammar; Iman Tavassoly, for introducing me to the Wilson-Cowan modeling; and Tian Hong, who helped finding parameters for the library of parts that is used on the Wilson-Cowan grammar. Additionally, my stays at the Thematic Research School modeling complex Biological Systems in the Context of Genomics, thanks to François Képès, have been the source of inspiration for the discrete modeling approach presented.
I would like to acknowledge my thesis committee, who raised points to be addressed in my work. Their advice and critiques were the most helpful to guide me along the graduation path and to the completion of this thesis.
Finally, let me mentioned that Dennie Munson has been a great support throughout these years, helping me deal with academic matters. She is also doing a fantastic job for the Genetic, Bioinformatics and Computational Biology program –a program in which I had the freedom to gain knowledge in areas as various as biology, mathematics, and computer science, share my research in seminars, have the opportunity to focus 100% on my research matters thanks to GRA assistantships.

Attribution

Several colleagues aided in the writing and research for some chapters as described here.

Chapter 2: Synthetic Biology Open Language (SBOL) Version 1.1.0

This document is the result of discussions between participants in the Synthetic Biology Open Language Workshops held in Blacksburg, Virginia on January 7-10, 2011, San Diego, California on June 8, 2011, and Seattle, Washington on January 5-6, 2012:
Eduardo Abeliuk (Teselagen), Laura Adam (Virginia Bioinformatics Institute), Aaron Adler (BBN Technologies), J. Christopher Anderson (University of California, Berkeley), David A. Ball (Virginia Bioinformatics Institute), Jacob Beal (BBN Technologies), Swapnil Bhatia (Boston University), Michael Bissell (Amyris, Inc.), Matthieu Bultelle (Imperial College London), Yizhi Cai (Johns Hopkins University), Deepak Chandran (University of Washington), Joanna Chen (Lawrence Berkeley National Lab), Kevin Clancy (Life Technologies), Kendall G. Clark (Clark & Parsia, LLC.), Daniel Cook (University of Washington), Wilbert Copeland (University of Washington), Douglas Densmore (Boston University), Omri A. Drory (Genome Compiler, corp.), Drew Endy (BIOFAB and Stanford University), Michal Galdzicki (University of Washington), John H Gennari (University of Washington), Raik Gruenberg (IRIC, University of Montreal), Jennifer Hallinan (Newcastle University), Timothy Ham (Joint BioEnergy Institute), Nathan J. Hillson (Lawrence Berkeley National Lab), Cassie Huang (Boston University), Jeffrey D. Johnson (University of Washington), Marc Juul Christoffersen (BIOFAB), Kyung H. Kim (University of Washington), Richard Kitney (Imperial College London), Allan Kuchinsky (Agilent Technologies), Sung Won Lim (Genspace), Matthew W. Lux (Virginia Bioinformatics Institute), Curtis Madsen (University of Utah), Akshay Maheshwari (BIOFAB), Goksel Misirli (Newcastle University), Barry Moore (University of Utah), Chris J. Myers (University of Utah), Josh Natarajan (Autodesk Research), Ernst Oberortner (Boston University), Carlos Olguin (Autodesk Research), Jean Peccoud (Virginia Bioinformatics Institute), Josh Perfetto (Cofactor Bio, LLC.), Hector Plahar (Joint BioEnergy Institute), Darren Platt (Amyris, Inc.), Matthew Pocock (Newcastle University), Jackie Quinn (Harvard University), Sridhar Ranganathan (Life Technologies), Cesar A. Rodriguez (Genome Compiler, corp.), Nicholas Roehner (University of Utah), Vincent Rouilly (University of Basel), Herbert M. Sauro (University of Washington), Evren Sirin

(Clark & Parsia, LLC.), Trevor F. Smith (Agilent Technologies), Lucian P. Smith (University of Washington), Guy-Bart Stan (Imperial College London), Jason Stevens (University of Utah), Vinod Tek (Imperial College London), Alan Villalobos (DNA 2.0, Inc.), Mandy Wilson (Virginia Bioinformatics Institute), Chris Winstead (Utah State University), Anil Wipat (Newcastle University), and Fusun Yaman Sirin (BBN Technologies).
The ongoing work was finalized through email exchanges on the SBOL Developers mailing list through March 2012.

Chapter 3: A Step-by-Step Introduction to Rule-Based Design of Synthetic Genetic Constructs Using GenoCAD

- Mandy L. Wilson (Virginia Bioinformatics Institute, Virginia Tech): Mandy worked on the software and tested it. She wrote the manuscript.
- Russell Hertzberg (Blenheim Technologies, Inc): Russell worked on the implementation.
- Jean Peccoud (Virginia Bioinformatics Institute, Virginia Tech): Jean was the principal investigator. He outlined the direction to take with GenoCAD, oversaw the work and the manuscript.

Chapter 4: Development of a domain-specific genetic language to design *Chlamydomonas reinhardtii* expression vectors

- Mandy L. Wilson (Virginia Bioinformatics Institute, Virginia Tech): Mandy implemented the context-free grammar editor in GenoCAD and wrote significant sections the manuscript and figures.
- Sakiko Okumoto (Department of Plant Pathology, Physiology and Weed Science,Virginia Tech): Sakiko worked on the Chloroplast grammar.
- Jean Peccoud (Virginia Bioinformatics Institute, Virginia Tech): Jean was the principal investigator. He oversaw the work, developed the grammar and wrote sections of the manuscript.

Chapter 5: Modeling Structure-Function Relationships in Synthetic DNA Sequences using Attribute Grammars

- Yizhi Cai (Virginia Bioinformatics Institute, Virginia Tech): Yizhi is the primary author of the work and was responsible for the development of the framework to apply attribute grammars to DNA sequences.
- Matt W. Lux (Virginia Bioinformatics Institute, Virginia Tech): Matt worked on the exploration of the design space to illustrate the use of the framework.
- Jean Peccoud (Virginia Bioinformatics Institute, Virginia Tech): Jean was the principal investigator. He oversaw the work and wrote sections of the manuscript.

Chapter 6: Design of Languages for Systems and Synthetic Biology to Translate Genetic Designs into Mathematical Models

- Mandy L. Wilson (Virginia Bioinformatics Institute, Virginia Tech): Mandy integrated the attribute grammar compiler generation module to GenoCAD and modified the grammar editor to support attribute grammars. She also reviewed the paper.
- Matt W. Lux (Virginia Bioinformatics Institute, Virginia Tech): Matt contributed to the definition and framing of our biological languages.
- Tian Hong (Department of Biological Sciences, Virginia Tech): Tian fitted the WC model for the library of parts proposed in order to design a switch and an oscillator.
- Jean Peccoud (Virginia Bioinformatics Institute, Virginia Tech): Jean was the principal investigator. He oversaw the work and provided guidance on the methods and results.

Chapter 7: Strengths and limitations of the federal guidance on synthetic DNA

- Michael Kozar (Virginia Tech): Michael contributed to the development of the algorithm and the tests. He also participated in the writing.
- Gaelle Letort (ENSIMAG): Gaelle contributed to the development of the algorithm and worked on its implementation.
- Olivier Mirat (ENSIMAG): Olivier contributed to the development of the algorithm and worked on its implementation.
- Arunima Srivastava (Virginia Tech): Arunima contributed to the development of the algorithm and worked on its implementation.
- Tyler Stewart (Virginia Tech): Tyler contributed to the development of the algorithm and the tests. He also participated in the writing.

- Mandy L. Wilson (Virginia Bioinformatics Institute, Virginia Tech): Mandy tested the software.
- Jean Peccoud (Virginia Bioinformatics Institute, Virginia Tech): Jean was the principal investigator. He proposed and oversaw the research. He wrote significant sections of the manuscript.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Need for Predictive Genotype to Phenotype Maps in Synthetic Biology

1.1.1 Computational Challenges in Mapping Genotype to Phenotype for Behavioral Predictions

Linking genotype –the genetic constitution of an organism– and its phenotype –observable characteristics– and then using them to make predictions has been a challenge for biologists. In order to develop a computational system that would allow predictions based on available biological knowledge, it is necessary to first understand the relations and dependencies between the genetic code and the outcome of its expression, measurable in terms of observable characteristics.
A classic approach in understanding Genotype-to-Phenotype (G2P) relationships is based on Mendelian genetics. In the 19th century, Mendel summarized his findings in two inheritance laws: Segregation and Independent Assortment. While DNA was first discovered in 1869 by Miescher, it was only understood to contain genetic information in 1944 with the Avery, MacLeod and McCarty experiment [Avery, MacLeod, and McCarty 1944]. Hence, the First Law can now be stated as one copy of each allele of the diploid parent is randomly passed to its offspring, and the Second Law as each trait is passed independently.

Later on, researchers found that genes may be linked and argued that Mendel's laws may fail to explain complex phenotypes [Mackay 2001] where a group of non-allelic genes together influence a phenotypic trait (polygenic or quantitative inheritance). Those multi-factorial traits may be further explained by taking into account epigenetic factors [Wong, Gottesman, and Petronis 2005] such as environment, developmental period, spatial and temporal expression or gene co-regulation [Lander and Schork 1994] and consequently, linear systems cannot

explain those phenotypes [Peccoud et al. 2004].

The linear model appears insufficient to predict phenotypes; however, it remains difficult to build a nonlinear system based on sparse observations. The task resembles opening a black box for which only input and output can be directly measured; the inside mechanism remains unknown. The past few decades brought a significant increase in the amount of biological data thanks to the availability of high-throughput technologies, such as the next generation of sequencing technologies or genome-wide association studies (GWAS) like the 1,000 Genomes Project [Siva 2008], letting us foresee progress in identifying those complex relationships. Simultaneously, computing and data storage capacity have exponentially progressed leading to the development of bioinformatic tools to handle such a quantity of data. This great source of information is necessary to elucidate G2P mapping using data mining approaches to extrapolate a model from hypothesis but not sufficient to compute a new phenotype from genomics information (environmental factors are assumed to be part of the mathematical model), the formal representation of a map is still to be defined. The form in which those are presented is a rather ad-hoc static map. Indeed, the organism-specific databases [Bruskiewich et al. 2003] or cross-organisms databases [Kahraman et al. 2005] that have been proposed either rely on free-text entries or ontologies [Hong et al. 2008; Mungall 2007], a control vocabulary for gene and protein roles across species to describe their biological processes, molecular function and cellular components [Ashburner, Ball, and Blake 2000]. Likewise, dbGaP –the integrative database from NCBI– contains study documentation, phenotypic, and genetic data and statistical results will help to uncover G2P relationships [Mailman et al. 2007].

A DNA sequence cannot carry phenotypic information without context-dependency. Those relationships, still to be fully uncovered, are nonlinear [Peccoud et al. 2004; Pigliucci 2010]. Databases do not appear to be appropriate to store a model computing nonlinear relationships; still, they can be used for analytics purposes by deduction (data mining). Yet, we need to make predictions to adequately build hypotheses to be tested in vivo in order to discover new knowledge. We need a way to model genetic data to encompass nonlinear relationships between molecules in an attempt to explain a phenotype. Ultimately, formalizing Genotype-to-Phenotype relations will shed light on understanding the genetic basis for health and diseases, such as cancers or diabetes. Therefore, new forms of mapping both levels of information should be explored and easily applied to the promising field of synthetic biology.

1.1.2 Designing in Synthetic Biology: CAD Software from Design to Simulation

1.1.2.1 What is Synthetic Biology?

In short, synthetic biology is a relatively new field in which researchers explore the possibilities of engineering life. Synthetic biology aims to design new biological systems for a given behavior or to re-design existing systems in order to understand their functionalities. Synthetic biology has been possible thanks to the rapid progress of biotechnologies. During the second part of the 20th century, major advances were made to get to the point where the sequencing of various organisms became possible, from Phage $\phi - X174$, the first DNA genome by Fred Sanger in 1977, to Human DNA sequences uncovered in 2001 [Venter et al. 2001]. A few striking advances in synthetic biology were made: in 2002, the first chemical synthesis of a complete genome (a poliovirus) was performed [Cello, Paul, and Wimmer 2002] and in 2008, Craig Venter Institute synthesized a complete bacterial genome [Gibson et al. 2008]. In addition, DNA synthesis appears as a major tool for rapid progress in synthetic biology by making the creation of DNA sequences from scratch, both inexpensively and rapidly.

Hence, with the advent of high-throughput DNA sequencing and synthesis, it became possible to produce a list of molecular parts composing a living organism. Deconstruction helps with the simplification of existing organisms to later introduce new functionalities, like the search for a minimal cell [Glass et al. 2006]. This is called the top-down approach to synthetic biology. Alternatively, the bottom-up approach consists of engineering context-independent biological parts that could be properly interfaced and standardized to generate a targeted behavior into an existing organism. Lately, people have also investigated making new chemical cellular life, using only materials that were never alive for protocell research [Szostak, Bartel, and Luisi 2001].
Finally, the recent breakthrough at Craig Venter's institute where cells were controlled by a synthetic genome from a close species that has a small set of genes, yet is capable of independent growth [Gibson et al. 2010], revealed a promising future for this field. The recent advances are very encouraging and help provide a better understanding of complex biochemical systems.

1.1.2.2 Designing Biology

Synthetic biologists are interested in designing new biological systems. The field is largely based on the idea of combining standardized genetic parts to form genetic circuits with specific functions, an analogy with the field of electronics [Densmore and Anderson 2009]. The most famous synthetic biology designs are the genetic toggle switch [Gardner, Cantor, and Collins 2000] and the repressilator [Elowitz and Leibler 2000], which consist of two and three genes where each gene's expression may be inhibited by the other genes' products.

Design requires mastering molecular biology as well as mathematics and the common process to design a genetic construct relies on cutting and pasting of different DNA sequences from desired parts. The process can be quite laborious and is error prone and obviously does not account for dynamical outcomes a priori. It is also unlikely to scale up for complex systems. To overcome these issues, over the past 10 years, several synthetic biology software applications have been developed to help with the management and assembly of genetic parts into biological designs [Lux et al. 2012a], often referred to as Computer-Assisted Design (CAD) for synthetic biology.
We can observe two methodologies in CAD systems: forward and reverse engineering. While the former is concerned with assembling smaller units to form larger systems, the latter is concerned with a behavioral goal and figures out what ought to be designed to observe this goal.

Platforms to design synthetic biology constructs often offer the possibility to both design and then simulate the constructs, as well as some more or less extended parts and construct management features. Eventually they also offer support such as primer design for the realization of the design in wet lab. There are several applications intended as CAD platforms for synthetic biology that I will review.

Interestingly, software offers various scales of precision to design a DNA molecule. The finest scale is found in APE, A Plasmid Editor [Davis 2012], and Gene Designer by DNA 2.0 [Villalobos et al. 2006]. Indeed, both offer the possibility to work on the DNA sequence itself, select sections, look at restriction sites and make local alignments. It is oriented towards the wet lab realization of the construct in a plasmid by proposing to design the primers. These two applications clearly require the understanding of the realization of the construct using wet lab techniques. Therefore, it is not clearly targeted to synthetic biology –a community rather interdisciplinary where some are modelers only. Additionally, those softwares do not support the concepts of parts and the management of databases of genetic parts, although Gene Designer lets users add parts by hand and then organize them as folders. Still talking about DNA sequence itself, genetic parts can be designed with particular properties, for instance based on a thermodynamics model, like the RBS calculator [Salis, Mirsky, and Voigt 2009].
Then, more traditionally in the idea of assembling genetic parts and including the notions of parts management, Clotho has been developed by Douglas Densmore and his colleagues [Densmore and Anderson 2009]. It provides a canvas with a drag and drop interface to virtually assemble genetic parts represented as icons. The design can be saved and several plugins allow for the generation of a model to be simulated and the use of tools for wet lab preparation. Clotho design is somewhat similar to describe a wet lab construct, so is SynbioSS [Hill et al. 2008] in which users input parts to generate the model. In the line of reverse engineering, in BioNetCAD [Rialle et al. 2010] the design step starts using a GUI interface to wire entities which rely on an abstract language. Once the qualitative behavior is found, BioNetCAD looks for genetic elements to satisfy the constraints defined within the CompuBioTicDB database of molecules and devices. Then, the simulation step offers to

handle noise with Hsim simulation from the CellDesigner model.
The last group of software handles directly logical gates, gene network. Explicitly mapping reactions, BioJADE [Goler 2004] from MIT allows its users to design a construct through a GUI interface by selecting components (logic gates mode of biobricks) and then linked them with wires. Later, through a "compilation process" of logic reduction of gates, they let the users simulate the construct. Users of CellDesigner [Funahashi et al. 2003] design by connecting species placed into components through an interface, and then save the network as SBML to be simulated. There is no compilation but the users define reactions by hands, like a visual interface to build a SBML model.
TinkerCell [Chandran, Bergmann, and Sauro 2009] allows users to drag and drop biological elements such as mRNA, proteins etc., as well as reactions. TinkerCell is mainly an interface that offers people to customize the design software itself with their own plugins. Hence, plugins allow for various forms of mathematical production and simulation.

Finally, purely in the tradition of reverse engineering, applications based on a fitness function can search for parameters and adequate network topology; an example would be Genetdes [Rodrigo and Jaramillo 2007]. Because of the lack of *in vivo* parameters [Lux et al. 2012b], this approach remains quite theoretical for DNA molecule design.

This overview provides a list of the different approaches to design, reverse- or forward- and scales, DNA to networks (and foreseeably genomes [Képès et al. 2012]). Several features have been developed and can be found in several CAD systems: parts management, primer design and other molecular biology tools, and simulation through the generation of a corresponding mathematical model. One will notice that among all of these applications there is little guidance as to how to compose a genetic network within applications and most require some expertise or clear intentions. Given the short existence of this field, it is more common to do forward engineering because knowledge about *in vivo* behavior is partially unknown. Hence, a common ground principle for all the software presented is to assume the users have a design in mind. They do not provide any guidance as to how to create a valid biological design, which is perhaps needed in such a new field.

I joined the synthetic biology research group at the Virginia Bioinformatics Institute with the idea to further develop the Computer-Assisted Design (CAD) software that they were working on. Starting my research in a relatively recent field, I conducted user interviews at the 2009 International Genetically Engineered Machine (iGEM) jamboree in Cambridge, MA in which I asked various synthetic biology researchers about their practices to design their constructs. I noticed that no software was systematically used although users were familiar with pretty much all the software options offered to them but rather, they would use *ad hoc* processes to put together the parts such as excel spreadsheets. In parallel, New City Media a design agency in Blacksburg, Virginia, conducted actual and potential users interviews.
With the materials compiled, reinforced by citation data in Table 1.1, it appears that none of the CAD tools seems to be fully satisfactory from a user standpoint since none have been adopted. Features that would interest the users were linked to data exchange and

collaboration as well as more extensive and higher quality parts repositories that would be easy to search.

Table 1.1: Comparison of citations of synthetic biology CAD software. The table lists papers describing synthetic biology software and their citation numbers. These are put into perspectives with a few selected papers describing system biology software applications (the numbers of citations have been retrieved from PubMed on 6/6/2013).

Type for comparison	Software	Publication	Number of citations
CAD synthetic biology	Gene Designer	[Villalobos et al. 2006]	46
	RBS calculator	[Salis, Mirsky, and Voigt 2009]	79
	Clotho	[Densmore and Anderson 2009]	24 (ACM data)
	SynbioSS	[Weeding, Houle, and Kaznessis 2010]	22
	BioNetCAD	[Rialle et al. 2010]	1
	TinkerCell	[Chandran, Bergmann, and Sauro 2009]	18
	BioJade	[Goler 2004]	2
Platforms system biology	Copasi	[Hoops et al. 2006]	177
	Cytoscape	[Shannon et al. 2003]	1281
	Project Galaxy	[Giardine et al. 2005]	187

These remarks echoed to ongoing work in the synthetic biology community that has come together to develop a standard language to exchange data between research labs and across applications. Initially, Provisional BioBrick Language (PoBoL) was proposed and it only focused on biobricks from the MIT registry. Later, an ongoing project in which our research group has been taking part is The Synthetic Biology Open Language (SBOL) that contains both a visual and a core component [Galdzicki et al. 2012]. Hence, up to this point, the version of SBOL released is only descriptive. SBOL can describe a DNA component but does not include any semantics nor any rules as far as making a design; it is a specific visual and vocabulary toolkit. Such language is a great tool to overcome issues of cross applications given the limited scale of each it is important to be able to switch from one CAD tool to another depending on the task. Still, an interest of this thesis is to understand how to map genotype to phenotype. Hence, looking at the techniques used to generate the mathematical model provides examples to answer this challenge.

CAD softwares providing simulation of designs use Domain-Specific Languages (DSLs) for synthetic biology, that is a programming language specifically tailored for DNA. It means that a genetic design can be compiled into a mathematical model [Clancy and Voigt 2010]. Few of those languages have been described in the literature.
The BioPsi language [Pérès et al. 2010], used in BioNetCAD, takes bioobjects and derives the biological processes as formally described by syntax rules. Another formal language is Genetic Engineering of living Cells (GEC) [Pedersen and Phillips 2009]. The syntax helps define parts and designs as modules encompassing their semantics. The Eugene language [Bilitchenko et al. 2011] uses both top down and bottom up design in which users specify a design made out of bioparts. It conveniently allows for a design exploration, looking at it as a combinatorial space. Then the design can be simulated or made in the lab by robots [Leguia et al. 2011].
While Eugene is focused on a formal description of a design, it uses Proto biocompiler [Beal, Lu, and Weiss 2011] to get the mathematical model. An abstract behavior described in a language close to boolean logic can be compiled into a Gene Network layout, which can then be reduced. Finally, the reduced schema can be implemented with matching a DNA sequence from a list of available parts.
In most DSL for synthetic biology, genetic parts have been defined as modules, encompassing dynamical data for their "own" behavior. A genetic design becomes then the aggregation of each dynamical behavior. The idea for modeling interactions is to explicitly include all of the interaction equations. For example, A is a trans-element interacting with B. B is therefore going to carry the equation of its interaction with A whether or not A is present in the system. Still the interaction equation will play a role in the simulation only if A is not null, that is, only if there is an equation for the production of A.
In reality, assembling genetic parts does not merely result in the concatenation of mathematical models of each unit. This view borrows from electrical engineering but does not account for limitations due to context dependencies [Lux et al. 2012a]. There is no computation aspect in this approach, failing to possibly uncover non hard-coded nonlinear relationships and be the support for G2P maps; still, known small logical gates can be assembled into larger systems and simulated. In addition to helping the users with the design of its constructs, most DSLs for synthetic biology intend to provide a mathematical model of the design.

Domain-Specific Languages (DSLs) are able to generate the mathematical model of constructs, given the kinetic data available, but their application are very narrow and unlikely to model Genotype to Phenotype (G2P) maps. Each DSL is fitting a problem in the field of synthetic biology is still changing rapidly and the applications are of a wide variety. There is no existing universal language for biology, nor universal approach to design. For the CAD software to be used, it must be able to support users personalization (mathematical modeling choices, genetics parts, scales of design, etc.). Those modifications require their users to mastering programming skills in order to write plug-ins in the language the software is using. Additionally, the lack of biological knowledge in mapping genotype and phenotype can only call for great customizability by the users.

1.1.3 Insights: Biology as a Natural Language - A View of G2P Maps from Natural Language Processing

A language is a support for information and often, it has a written form that carries a meaning, which depends on the context of expression. Similarly, genetics can be seen as a language. Inspiration for this work comes from Ji's isomorphism paper [Ji 1997]. First of all, genetic expression as a natural language is a common metaphor inferred from the early terms chosen to described genetic mechanisms (transcription, translation, coding sequence, "book of life", etc.). They have been directly borrowed from linguistics. In the context of this thesis I would like to frame it as follows.
The DNA sequence is made of letters As, Cs, Ts and Gs, which are the support for gene expression; the DNA sequence's meaning is voiced by the processes of transcription and translation. Similarly to a text, there is a support for information (the genotype) and a meaning (the phenotype). Context of the genetic expression is very important and is intrinsically linked to the genome layout [Képès et al. 2012], as well as the host mechanism and other experimental conditions, and we know that context is key in understanding the meaning of a sentence. For example, "he went there" means different things depending on the previous sentences.

Additionally, a few specific points of linguistics relate closely to gene expression in Table (1.2).

Table 1.2: Parallels Between Challenges in Natural Language Processing and Genotype to Phenotype Mapping. The methods used for solving these challenges in natural language could –hopefully– be re-used in biology

Linguistics	Genotype to Phenotype Mapping
Text and its meaning	Genotype and its phenotype
Context	Importance of layout, host organism, etc.
Anaphora	Trans-interaction
Inflectional morphology	Cis-interactions
Linguistics universals	Cassette motifs

Trans-interactions are interactions that happen from a distance. They can be the regulation of a promoter by proteins produced in the cell. Those can be seen as their linguistics counterpart, anaphoras, such as the semantic relation between a pronoun and the noun it refers to.
On the other hand, cis-interactions, which are local interactions (for example, a RNA molecule folding, thus preventing its translation) can be compared to an inflectional morphology mechanism, like the agreement of the verb according to the subject.
The concept of defining formal language relies on the underlying idea that language have common properties. Going further in the studies, there are actually linguistics motifs that can be found across natural languages. Similarly, in biology the central dogma calls for the

transcription of DNA into RNA which then serves as a template for the final protein by the translation process. The dogma requires the presence of a promoter region to initiate the transcription, a ribosome-binding site for the translation initiation of the gene coding sequence and finally a terminator to end the transcription – which are the basic elements to form a gene cassette.

We can re-use the natural language processing tools being developed since the 50s to understand how syntax, that is, the DNA sequence, and its meaning, i.e. the phenotype, can be mapped. Meanwhile, the analogy view provides us with numerous tools to understand a DNA sequences and interpret it to predict its phenotype, and ultimately we could generate DNA sequences that convey a certain meaning. Such encouraging perspectives rely on the common properties found across natural languages and ought to be found in genetics.

However, natural languages cannot be handled by computer as they are. Noam Chomsky is considered as father of formal languages, which is a conceptual framework to define a language and allow for sentence processing. A formal language is the set of all the words that can be composed over an alphabet by applying rules. They have been used for interpretation and as models. Depending on the form of the grammar rules, Chomsky's hierarchy divides languages into the increasingly restrictive syntactic dependencies.

With such correlation between G2P and natural languages being established, building a DNA language that associate a meaning to a genetic sentence will shed light on understanding the genetic basis for health and disease, for instance cancers or diabetes. We will now look at the steps that have been taken to formalize biology as formal languages.

1.2 Computational Linguistics Methods Applied to Biology

1.2.1 Formal Languages Background

Definition

A formal grammar G specifies a formal language and is defined as follow. A tuple $G = \langle N, T, P, S \rangle$, where:

- V is Vocabulary is defined as $N \cup T$
- P is a finite set of production rules (left-hand side $\rightarrow$ right-hand side) where the left-side is in N and the right-side is in $V*$
- N is a finite set of nonterminal symbols
- T is a finite set of terminal symbols

- S is a start symbol and belongs to N

In order to illustrate this definition, we provide a sample grammar for Subject-Verb-Object (SVO) sentences in English (see Figure 1.1). In this case, T is the set of words from the dictionary. The start symbols is a *Sentence* and the rules indicates how to form valid SOV sentences. *Subject, Object, Verb, NounPhrase, Modifier* are nonterminals.

Context-Free Grammar (CFG) Context-Free Grammars (CFG) have rules of the form: $X \rightarrow \alpha$, where X is a single nonterminal and α is a string of nonterminals, terminals, or may be empty. According to Chomsky's hierarchy, CFGs are of type two, between regular language (recognized by a finite state automaton) –by definition a sub set of CFGs– and context-sensitive language, which remains a computational challenge and therefore is out of the interest of this thesis.

Derivation and Trees

A word of formal language is obtained by derivation, which can be represented as a tree. Starting from the start symbol, the root, rules are applied step by step and drawn as branches, indicating the deep structure of the word until all branches lead to a nonterminal, the leaves. A tree is shown for the sample grammar in Figure 1.1. If more than one tree can describe a word, the grammar is said to be ambiguous.

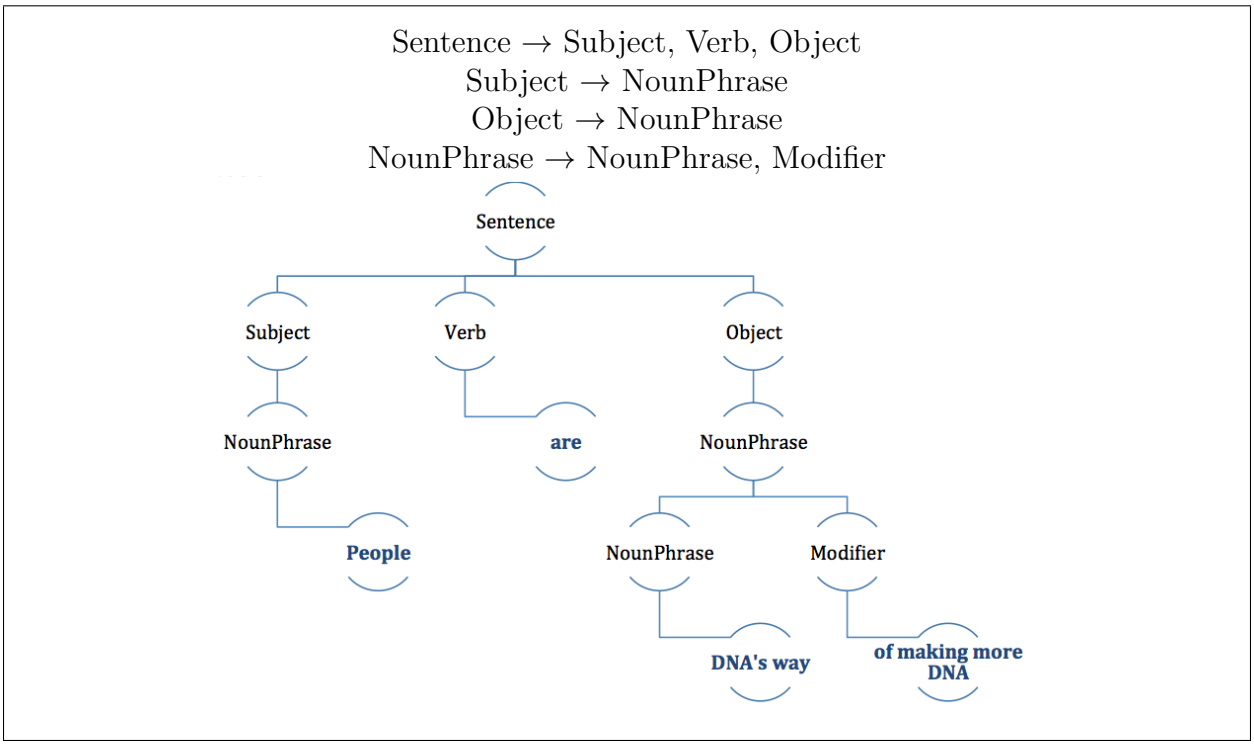


Figure 1.1: Sample English Context-Free Grammar. In this sample example, the context-free rules indicate how to form a valid sentence over the English dictionary. Sentence is the start symbol. From Sentence, rules can be applied to derive a sentence syntactic structure of the language. Terminals, the words of the dictionary, can be chosen to finish the sentence. We used a quote from Edward O. Wilson (1975) as an illustration of the derivation process.

Syntactic Model of DNA Sequences: Context-Free Grammar for Parts-Based DNA Molecules

The use of Context-Free Grammars has been proposed by Collado-Vides in 1991 to organize the functional elements needed for the transcription of genes in prokaryote cells [Collado-Vides 1991]. In 2007, a similar formalism based on Context-Free Grammars has been applied to validate and design genetic constructs made of standard biological parts in the field of synthetic biology [Cai et al. 2007]. Such system insures that all the elements are present in the final designed DNA sequence. It can also help systematically manage part assembly standards for construction [Cai, Wilson, and Peccoud 2010]. It can also indicates what is missing in an existing invalid design.
In this context, the terminals are the genetic parts. The nonterminals are called categories. The grammar rules indicate how categories can be arranged to form a 'valid' design, parts from the library implement it. We provide an example of a language to design poly-cistronic

gene expression cassettes in Figure 1.2.

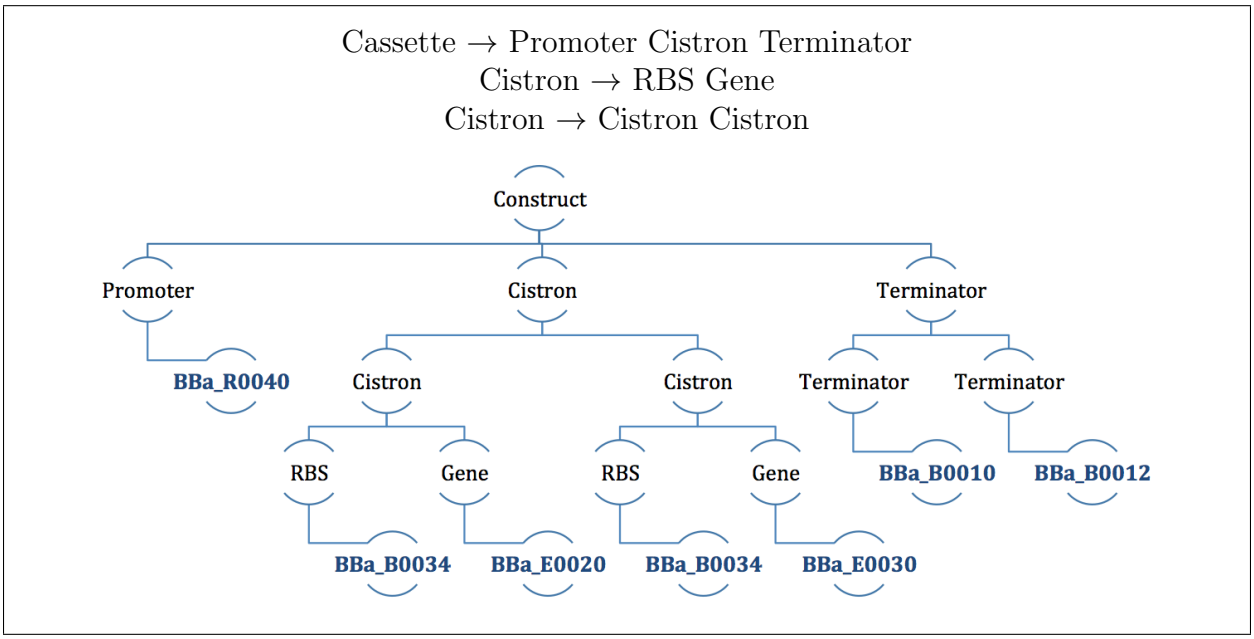


Figure 1.2: Sample Genetic Construct Context-Free Grammar. The rules indicate how to form a valid construct over a library of genetic parts. The rules are applied to define the structure of the construct, ensuring that all of the essential parts for the transcription and translation of the protein are present in the final constructs. Then, parts from each of the categories are chosen to make an actual design. In this sample genetic construct grammar example, we used a the BioBrick repository as our library of parts and re-designed a polycistronic CFP/YFP cassette (BBa_J13004) as an illustration of the derivation process.

1.2.2 Overview of Syntactic Forms of Biological Sequences and Their Significance

1.2.2.1 The Nucleotide Sequence

As traditionally defined, the nucleotides sequence is a chain of four bases: purine bases (adenine -A- and guanine -G-) and pyrimidine bases (thymine -T- and cytosine -C-). The sequence is usually paired with a complementary strand, to form the double helix, as described by Watson and Crick in 1953. The nucleotide sequence serves as a template for the storage of information. When a cell divide, the sequence is copied in order to conserve the genetic information. The DNA sequence contains regions that are transcribed into RNA. It requires the binding of the RNA polymerase to the promoter region. This is the first step in the genetic expression central dogma, which has been a central focus to understand

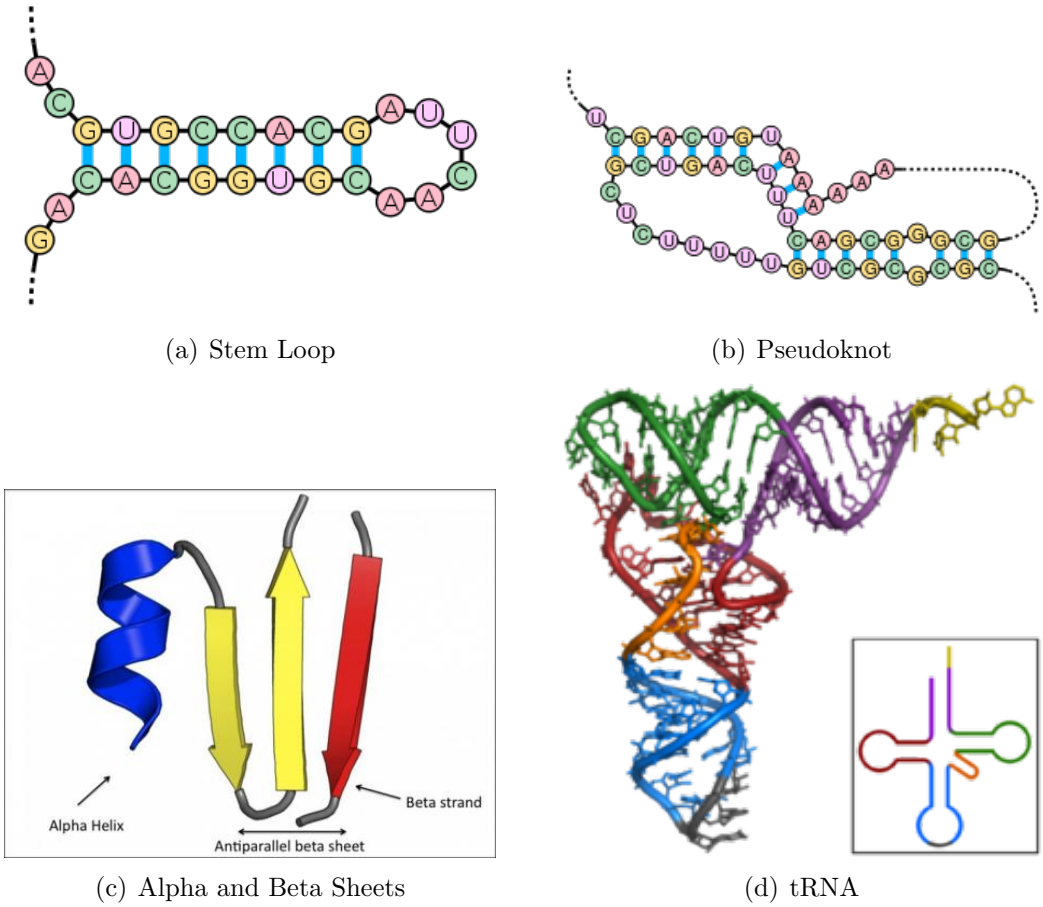
phenotype as the expression of proteins.

An early approach in defining a language to represent DNA molecules used Regular Languages to show the modeling of the translation of a RNA sequence into a protein sequence and, in particular, identify the gene coding sequences by recognizing ribosomal binding sites and start and stop codon locations [Busse and Brendel 1984]. Later work consisted in building a vocabulary based on four tuples that correspond to the pair-wised bases: $T = \{[A/T], [C/G], [G/C], [T/A]\}$. This approach, proposed by Tom Head [Head 1987], offers the possibility to formally describe a double-stranded DNA molecule instead of single stranded ones. But he also gives rules for finding splicing sites extending from regular languages to context free languages.
Indeed, eukaryote genes may contain alternate sequences of exons and introns. After the RNA splicing, the mature RNA molecule contains only exons, which is then translated into a protein. The process of alternate splicing is extremely interesting: depending on how the mRNA is processed the final mRNA is different. Hence, a single molecule of mRNA can be coding for different proteins carrying often different functions. Logically, splicing correspond to transformation rules in formal languages and others have modeled this phenomenon. With further abstraction of the DNA sequence, David Searls [Searls 1988] argues not to make the distinction between As and Gs and Ts and Cs but rather refer to them only by their sugar (ribose or deoxyribose). He then has gene syntax with the cat and 'tata' boxes and then start and stop codons identification. This provides enough details to build a syntactic grammar that describes gene regions, identifies exons and introns and models alternative splicing. While he proposed a simple system, Searls underlines that more sophisticated mechanisms will be needed. Additionally, Kudo et al. [Kudo et al. 1987] used consensus sequences (for 5'-splice site) to search for pattern in natural sequences from GenBank. Not only used for intron/exon, rules may instead represent mutation and with the use of probabilities to weight them, provide tools for global sequence alignment [Searls 1995].

This detailed level of abstraction can conveniently model restriction sites and generate the potential mismatched molecules after ligase and recombination, thanks to the splicing system [Head 1987]. Since DNA is the support for overlapping information (chromatin code or putative loop code), the theory presented when used as translation system obtains the various levels of information, such as considering all the possible frames of translation[Trifonov 1989].

Yet, because genetic regulation would be very difficult to identify at the DNA sequence level in the case of natural sequences (given the natural variations), a higher level of abstraction has been studied. In 1991, Collado-Vides proposed the Unit of Genetic Information (UGI) as the smallest level of description of a DNA sequence. UGIs correspond to DNA sequences with a particular functional role in gene expression such a promoter or an activator region [Collado-Vides 1991]. As shown in his paper, UGIs are not overlapping and therefore could represent regulated gene expression mechanisms, such as operons.

1.2.2.2 RNA, Amino Acids and Proteins



(a) Stem Loop

(b) Pseudoknot

(c) Alpha and Beta Sheets

(d) tRNA

Figure 1.3: Secondary structures of RNA and AA sequences. Those display syntactic patterns that have been modeled with formal languages theories. Images from wikipedia.org and EMBL-EBI 2013

Rewriting rules can make the translation from a DNA sequence to the corresponding amino acid (AA) sequence by enumerating the codon sequences to which the AAs correspond to with transformation rules [Searls 1988]. Then, the RNA secondary structure can be predicted by using grammar rules that will pair matching bases. Using Context Free Grammar, parse trees can have a nested structure, which is very convenient to describe stem-loop and eventually pseudoknots (problem with crossing) as shown in Figure 1.3: it recognizes symmetric sequences that are complementary and can pair to form the secondary structure.
Stochastic Context-Free Grammar (SCFG) for which now rules have a probability, has been used because it is able to find long range interactions and can account for mutation through the probability of an alignment [Knudsen and Hein 1999]. It usually must be trained over a set of valid and non-valid sequences in order to refine the probabilities. Thanks to a parsing

algorithm that maximizes the overall probability, it helps to find the most likely secondary structure of the molecule. Skakibara applied SCFG to transfer-RNA which particular loopy structure in Figure 1.3 make them very suitable and easily discriminated from other RNA molecules [Sakakibara et al. 1994]. Tree-adjoining Grammars have also been proposed. Similar to CFG, they propose to rewrite tree units instead of nonterminals; this allows them to parse pseudoknots [Kato, Seki, and Kasammi 2005; Kobayashi and Yokomori 1994].
More recently, formal description methods used for RNA secondary structure can also be applied to protein secondary structure and rules can identify alpha and beta sheets (see Figure 1.3) [Chiang, Joshi, and Searls 2006], which is based on the formation of hydrogen bounds; for example, work has been done using SCFG for beta sheets with an identification success of 74% [Abe and Mamitsuka 1997].
Interestingly applying rules to the amino acid sequence allows the classification of them hierarchically as groups and obtain information about amino acid biochemical properties (basic, aromatic, hydrophobic etc.). Some of these give clues to predict the secondary structure of the protein; computing the linguistic complexity is then performed to investigate protein structure [Jiménez-Montaño 1984].
Protein domains can be related to functional unit in a sentence and rather than purely syntax to structure mapping, structural semantics is used, closely following the metaphor with linguistics [Gimona 2006]. This work can provide a biochemical description and identify protein domains.

Finally, protein-protein interactions have been the interest of research using Natural Language Processing (information extraction) methods in scientific literature [Miyao et al. 2009; Temkin and Gilder 2003]. This gives an idea of the possible uses of various Natural Language Processing methods.

1.2.2.3 Expressivity of Syntactic Models

As just presented, multiple formalism of language can be used to model various biological molecules. The Figure 1.4 recapitulates these findings. One can understand by looking at the dependencies within a switch topology [Gardner, Cantor, and Collins 2000] that it would be hard to define grammar rules to enforce them.

Regular Languages
Use an automata to recognize a consensus sequence. Rules are of the form: $A \rightarrow a$ or $A \rightarrow A\ b$.

- Rewriting rules for the Central dogma: transcription of DNA sequence into its corresponding RNA molecules, translation of RNA molecule into Amino Acids chain.
- Translational elements of a RNA sequence into a protein sequence: start and stop codon to identify the gene coding sequences [Busse and Brendel 1984].
- Recognize 5' Splicing sites, expanded with weighted model [Kudo et al. 1987].

Context-Free Grammar
Rules are of the form: $A \rightarrow \gamma$

- Splicing sites [Head 1987]
- Using probabilities/stochastic:

  Global sequence alignment [Searls 1995].

  Translation of DNA sequence into AAs using translation rules based on the codon code [Searls 1988]

  RNA stem-loop [Knudsen and Hein 1999]

Context-Sensitive Grammar
Parsing such sentence is unpractical (the decision problem is PSPACE-complete). Rules are of the form: $\alpha A \beta \rightarrow \alpha \gamma \beta$

- Pseudoknot

Dependencies

(a) Regular Languages (b) Context-Free Grammar (c) Context-Sensitive Grammar

Figure 1.4: Syntactic Possibilities in Representing Biological Sequences with Formal Grammars. The work on formal language applied to biology reviewed has been classified according to the Chomsky's hierarchy. They give examples of what formal languages can model as far as complexity of biological sequences and their syntactic dependencies. Searls' paper has also been a source of inspiration for this figure [Searls 1997]

Generation vs. Parsing vs. Translation The formal language theory can be used to generate sentences by a derivation process. Starting from the Start symbol, rules are applied according to the possibilities. Choices are made to compose a new sentence of the language. Alternatively, by parsing a sentence, we can know whether a given sentence is part of the language or not. This requires a lexical analyzer and a parser. The lexer reads the input and return a list of tokens. In our approach, it reads the DNA sequence and return the list of genetic parts. The parser contains the grammar rules. It takes as input the list of tokens and return a Boolean (valid or not). It can also eventually return the parse tree.
If we use attribute grammars, when the parsing step occurs the semantics of the sentence is computed according to the parts attributes and semantic actions. The output, often called the 'intermediary code', can be further optimized. The final target code can then be generated. The compiler program perform all these tasks (lexical analysis, parsing, optimization, code generation).

Limitations Just like natural languages cannot be yet fully handled by computers and the theory of formal language provides a in-between solution, Natural biological sequences cannot be fully represented by formal languages as illustrated in Figure 1.5.

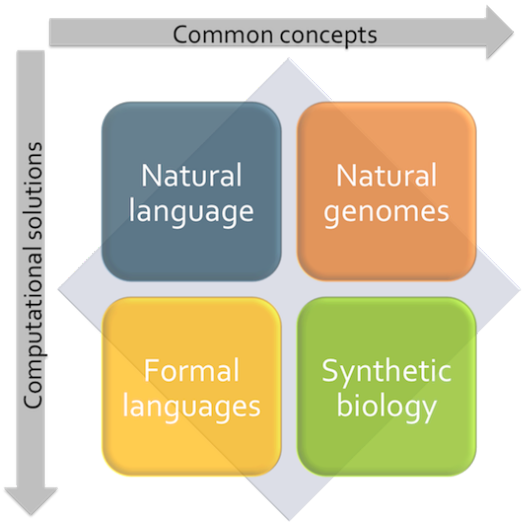


Figure 1.5: Parallels Between Language and Biology and the Need for a Computational Approach. Natural languages are complex to model, formal language theory has been proposed. As complex as natural languages, biological sequences cannot be modeled as is. The field of synthetic biology brings deconstruction of the DNA sequence into functional parts and drafts design principles. They allow to use formal languages to represent most of the biological designs.

Representing natural sequences remains a challenge for computer science because of the complexity of the genetic code (not deterministic, not linear, not context-free). Natural

sequences are ambiguous due to characteristics such as the degeneracy of the codon codes and the alternative splicing mechanism. Using context free grammars, overlaps cannot be represented [Collado-Vides 1991; Searls 1993].

However, there is some promising research that will be discussed in using context free grammar over a vocabulary of genetic parts in the field of synthetic biology. Instead of trying to understand the syntax of natural sequences with formal languages as previously done, we can apply the vision of system biologists and project it to the genome –hence focusing on region and units of interest within an entire genome-. This is promising in tackling natural sequences, because they focus on a smaller set of control elements, in an organized way and do not necessitate the understanding of all the particularities of the complex biological world.

Challenges that are still to be addressed in applying formal languages to represent biological sentence:

- A circular sequence (a plasmid)
- Read a message in both directions (both strands used to encode for proteins)
- One message multiple interpretations (alternative splicing)
- Inexact spelling (mutations)

In the previous section, most works propose either a descriptive model or a way to discriminate between sequences. Most papers use natural sequence datasets to prove that their proposition accurately identify sequences of interest. They propose to recognize the syntax of repeats, loops, etc. [Searls 1993].
Still, some recent work has taken the direction of grammatical inference, although most of it seems to be applied for stochastic and probabilistic grammatical models, rather than finding new syntactic rules or recognizing linguistics patterns in related natural biological sequences. Yet, little of the work has been exploring the generative capacity of formal languages, which appears needed for the field of synthetic biology and in particular, the design aspects.

1.3 Societal Impacts: Designing Safe Mutants

We have seen that CAD systems often integrate steps from the design and collections of genetics parts to the model generation and finally prepare for the wet lab realization. However, no check is being made to make sure the users are not designing potentially dangerous constructs. However, this thesis has considered these security aspects to be part of the design process.

Verifying the safety of a design is similar to checking its DNA sequence as it has been investigated by DNA synthesis companies. There are two approaches for automatically reviewing DNA sequences, either sequence-based prediction, which remains theoretical (and

would require a deeper understanding of G2P relationships), and sequence-based classification, which can hope to be implemented ["National Research Council (US) Committee on Scientific Milestones for the Development of a Gene Sequence-Based Classification System for the Oversight of Select" 2010]. The federal government has been working on regulating DNA synthesis order by outlining a screening protocol [Bucher and Department of Health and Human Services 2009], that we were the first to implement in order to assess its efficacy and work on improving such sequence screening software. It does not run a screen against a curated database (like BlackWatch by Craic computing [*BlackWatch*] that necessitates the setting of an appreciative threshold for a relevant match. Instead, the algorithm breaks down both strands of the DNA sequence and its six-frame translation in fragments that are blasted against the entire NCBI database, and top hits are automatically sorted and classified to find the ones that are uniquely related to Select Agents and Toxins with the help of keywords. We characterized the performance of the best match algorithm describe in the federal guidance on DNA synthesis orders. The resulting software, GenoGUARD, also creates opportunities for sequence screening to help bioinformaticians analyzing genomic data for particular targets. Not only does it encourage the participation of academy and undergraduate students to consider security questions in their science, but it is also a breakthrough in providing an automatic solution to monitor biosafety in gene synthesis industries.

1.4 Organization

This thesis is organized as follows:
The next chapter introduces the work that we have done with the synthetic biology community to develop a Domain Specific Language for synthetic biology, SBOL. The SBOL language allows synthetic biology CAD software users to exchange data. In the third chapter, the synthetic biology CAD software that we are developing at the Virginia Bioinformatics Institute since 2007, GenoCAD, is presented. GenoCAD initially used Context-Free Grammars to guide users through the design of DNA molecules. In order to accommodate the design of DNA molecules for various applications, scales and organisms, the following chapter details how Context-Free Grammars may be defined by GenoCAD users through an intuitive graphical user interface.
Then, we show how attribute grammars may be used to generate a SBML model of a genetic construct. They are a theoretical support for mapping genotype to phenotype, and conveniently explore genetic design spaces based on a target phenotype. This chapter shows a proof of concept.
In order to bring Attribute Grammars for design-to-simulation features to GenoCAD, we worked on two aspects of GenoCAD: the design of synthetic and system biology languages based on attribute grammars to answer project-specific needs, then the generation of compilers from project-specific grammars to automatically derive the mathematical model from the design. Once implemented, simulation of genetic constructs can be performed within GenoCAD, possibly using a newly defined language. Examples given show various languages to

model gene regulatory networks, and how to scale the biological languages to model natural genomes.
In the last chapter, we implemented and characterized a screening algorithm based on the federal guidance to prevent Select Agents and Toxins from being ordered from Gene Synthesis companies.

References

[1] Naoki Abe and H Mamitsuka. "Predicting protein secondary structure using stochastic tree grammars". In: *Machine Learning* 29.2-3 (Jan. 1997), pp. 275–301.

[2] Michael Ashburner, CA Ball, and JA Blake. "Gene Ontology: tool for the unification of biology". In: *Nature ...* 25.1 (2000), pp. 25–29. DOI: 10.1038/75556.Gene.

[3] Oswald T. Avery, Colin M. MacLeod, and Maclyn McCarty. "Studies on the chemical nature of the substance inducing transformation of pneumococcal types". In: *The Journal of Experimental Medicine* 6 (1944).

[4] Jacob Beal, Ting Lu, and Ron Weiss. "Automatic compilation from high-level biologically-oriented programming language to genetic regulatory networks." In: *PloS one* 6.8 (Jan. 2011), e22490. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0022490.

[5] Lesia Bilitchenko et al. "Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems." In: *PloS one* 6.4 (Jan. 2011), e18882. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0018882.

[6] R. M. Bruskiewich et al. "Linking genotype to phenotype: the International Rice Information System (IRIS)". In: *Bioinformatics* 19.Suppl 1 (July 2003), pp. i63–i65. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg1006.

[7] John R Bucher and Department of Health and Human Services. "Screening Framework Guidance for Synthetic Double-Stranded DNA Providers". 2009.

[8] H.G Busse and V Brendel. "Genome structure described by formal languages". In: *Nucleic acids research* 12 (Jan. 1984), pp. D1–4. ISSN: 1362-4962.

[9] Yizhi Cai, Mandy L. Wilson, and Jean Peccoud. "GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs." In: *Nucleic acids research* 38.8 (May 2010), pp. 2637–44. ISSN: 1362-4962. DOI: 10.1093/nar/gkq086.

[10] Yizhi Cai et al. "A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts." In: *Bioinformatics (Oxford, England)* 23.20 (Oct. 2007), pp. 2760–7. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btm446.

[11] Jeronimo Cello, Aniko V Paul, and Eckard Wimmer. "Chemical synthesis of poliovirus cDNA: generation of infectious virus in the absence of natural template." In: *Science (New York, N.Y.)* 297.5583 (Aug. 2002), pp. 1016–8. ISSN: 1095-9203. DOI: 10.1126/science.1072266.

[12] Deepak Chandran, Frank T Bergmann, and Herbert M Sauro. "TinkerCell: modular CAD tool for synthetic biology." In: *Journal of biological engineering* 3 (Jan. 2009), p. 19. ISSN: 1754-1611. DOI: 10.1186/1754-1611-3-19.

[13] David Chiang, AK Aravind K Joshi, and David B DB Searls. "Grammatical representations of macromolecular structure". In: *J Comput Biol* 13.5 (June 2006), pp. 1077–1100. DOI: 10.1089/cmb.2006.13.1077.

[14] Kevin Clancy and Christopher a Voigt. "Programming cells: towards an automated 'Genetic Compiler'." In: *Current opinion in biotechnology* 21.4 (Aug. 2010), pp. 572–81. ISSN: 1879-0429. DOI: 10.1016/j.copbio.2010.07.005.

[15] J. Collado-Vides. "The search for a grammatical theory of gene regulation is formally justified by showing the inadequacy of context-free grammars". In: *Bioinformatics* 7.3 (1991), p. 321.

[16] Wayne Davis. *ApE-A plasmid Editor*. 2012.

[17] Douglas Densmore and J. Christopher Anderson. "Combinational logic design in Synthetic Biology". In: *2009 IEEE International Symposium on Circuits and Systems* (May 2009), pp. 301–304. DOI: 10.1109/ISCAS.2009.5117745.

[18] M B Elowitz and S Leibler. "A synthetic oscillatory network of transcriptional regulators." In: *Nature* 403.6767 (Jan. 2000), pp. 335–8. ISSN: 0028-0836. DOI: 10.1038/35002125.

[19] Akira Funahashi et al. "CellDesigner : a process diagram editor for gene-regulatory and". In: 1.5 (2003), pp. 159–162.

[20] Michal Galdzicki et al. "Synthetic Biology Open Language (SBOL) Version 1.1.0". In: (2012), pp. 1–26.

[21] T S Gardner, C R Cantor, and J J Collins. "Construction of a genetic toggle switch in Escherichia coli." In: *Nature* 403.6767 (Jan. 2000), pp. 339–42. ISSN: 0028-0836. DOI: 10.1038/35002131.

[22] Belinda Giardine et al. "Galaxy: a platform for interactive large-scale genome analysis". In: *Genome research* 15.10 (2005), pp. 1451–1455.

[23] Daniel G Gibson et al. "One-step assembly in yeast of 25 overlapping DNA fragments to form a complete synthetic Mycoplasma genitalium genome". In: (2008).

[24] Daniel G. D.G. Gibson et al. "Creation of a Bacterial Cell Controlled by a Chemically Synthesized Genome". In: *Science* 329.5987 (July 2010), pp. 52–6. ISSN: 1095-9203. DOI: 10.1126/science.1190719.

[25] Mario Gimona. "Protein linguistics - a grammar for modular protein assembly?" In: *Nat Rev Mol Cell Biol* 7.1 (Jan. 2006), pp. 68–73. ISSN: 1471-0072. DOI: 10.1038/nrm1785.

[26] John I Glass et al. “Essential genes of a minimal bacterium.” In: *Proceedings of the National Academy of Sciences of the United States of America* 103.2 (Jan. 2006), pp. 425–30. ISSN: 0027-8424. DOI: 10.1073/pnas.0510013103.

[27] JA Jonathan A Goler. “BioJADE: A Design and Simulation Tool for Synthetic Biological Systems”. In: *Technical report, MIT, Cambridge, MA*. May (2004).

[28] T Head. “Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors”. In: *Bulletin of mathematical biology* 49.6 (Jan. 1987), pp. 737–759.

[29] Anthony D Hill et al. “SynBioSS: the synthetic biology modeling suite.” In: *Bioinformatics (Oxford, England)* 24.21 (2008), pp. 2551–3. ISSN: 1460-2059. DOI: 10.1093/bioinformatics/btn468.

[30] Eurie L Hong et al. “Gene Ontology annotations at SGD: new data sources and annotation methods.” In: *Nucleic acids research* 36.Database issue (Jan. 2008), pp. D577–81. ISSN: 1362-4962. DOI: 10.1093/nar/gkm909.

[31] Stefan Hoops et al. “COPASI—a complex pathway simulator”. In: *Bioinformatics* 22.24 (Dec. 2006), pp. 3067–3074. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btl485.

[32] S Ji. “Isomorphism between cell and human languages: molecular biological, bioinformatic and linguistic implications”. In: *BioSystems* 44.1 (Jan. 1997), pp. 17–39.

[33] MA Jiménez-Montaño. “On the syntactic structure of protein sequences and the concept of grammar complexity”. In: *Bulletin of Mathematical Biology* 46.4 (July 1984), pp. 641–659. ISSN: 0092-8240. DOI: 10.1016/j.bulm.2004.03.003.

[34] Robert Jones and Craic Computing LLC. *BlackWatch*. Tech. rep. Seattle, WA: Craic Computing LLC.

[35] Abdullah Kahraman et al. “PhenomicDB: a multi-species genotype/phenotype database for comparative phenomics.” In: *Bioinformatics (Oxford, England)* 21.3 (Feb. 2005), pp. 418–20. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bti010.

[36] Y Kato, H Seki, and T Kasammi. “On the generative power of grammars for RNA secondary structure”. In: *IEICE Transactions on Information and Systems* E88D.1 (Jan. 2005), pp. 53–64.

[37] François Képès et al. “The layout of a bacterial genome”. In: *FEBS Letters* 586.April (Mar. 2012), pp. 2043–8. ISSN: 00145793. DOI: 10.1016/j.febslet.2012.03.051.

[38] B Knudsen and J Hein. “RNA secondary structure prediction using stochastic context-free grammars and evolutionary history”. In: *Bioinformatics* 15.6 (Jan. 1999), pp. 446–454.

[39] S Kobayashi and T Yokomori. “Modeling RNA secondary structures using tree grammars”. In: *Proceedings of Genome Informatics Workshop V*. Vol. 105. 2. Citeseer, July 1994, pp. 191–206.

[40] Mineichi Kudo et al. "Syntactic pattern analysis of 5'-splice site sequences of mRNA precursors in higher eukaryote genes". In: *Comput Appl Biosci* 3.4 (Nov. 1987), pp. 319–324. ISSN: 1367-4803.

[41] ES Lander and Nj Schork. "Genetic dissection of complex traits". In: *SCIENCE-NEW YORK THEN WASHINGTON-* 265 (1994).

[42] Mariana Leguia et al. *Automated assembly of standard biological parts.* 1st ed. Vol. 498. Elsevier Inc., Jan. 2011, pp. 363–97. ISBN: 9780123851208. DOI: 10.1016/B978-0-12-385120-8.00016-4.

[43] Matthew W Lux et al. "Genetic design automation: engineering fantasy or scientific renewal?" In: *Trends in biotechnology* 30.2 (Feb. 2012), pp. 120–6. ISSN: 1879-3096. DOI: 10.1016/j.tibtech.2011.09.001.

[44] Matthew W Lux et al. "Genetic design automation: engineering fantasy or scientific renewal?" In: *Trends in biotechnology* 30.2 (Feb. 2012), pp. 120–6. ISSN: 1879-3096. DOI: 10.1016/j.tibtech.2011.09.001.

[45] TFC Mackay. "Quantitative trait loci in Drosophila". In: *Nature Reviews Genetics* 2.January (2001).

[46] Matthew D Mailman et al. "The NCBI dbGaP database of genotypes and phenotypes." In: *Nature genetics* 39.10 (Oct. 2007), pp. 1181–6. ISSN: 1546-1718. DOI: 10.1038/ng1007-1181.

[47] Y Miyao et al. "Evaluating contributions of natural language parsers to protein-protein interaction extraction". In: *Bioinformatics* 25.3 (Feb. 2009), pp. 394–400. DOI: 10.1093/bioinformatics/btn631.

[48] Chris Mungall. "Representing phenotypes in OWL". In: *OWL: Experiences ...* (2007).

[49] "National Research Council (US) Committee on Scientific Milestones for the Development of a Gene Sequence-Based Classification System for the Oversight of Select". "A Proposal for Consideration: Sequence-Based Classification of Select Agents". In: *Sequence-Based Classification of Select Agents: A Brighter Line*. Washington (DC): National Academies Press (US), 2010. ISBN: 9780309159043.

[50] Jean Peccoud et al. "The selective values of alleles in a molecular network model are context dependent." In: *Genetics* 166.4 (Apr. 2004), pp. 1715–25. ISSN: 0016-6731.

[51] Michael Pedersen and Andrew Phillips. "Towards programming languages for genetic engineering of living cells." In: *Journal of the Royal Society, Interface / the Royal Society* 6 Suppl 4.April (Aug. 2009), S437–50. ISSN: 1742-5662. DOI: 10.1098/rsif.2008.0516.focus.

[52] Sabine Pérès et al. "Computing biological functions using BioPsi, a formal description of biological processes based on elementary bricks of actions." In: *Bioinformatics (Oxford, England)* 26.12 (June 2010), pp. 1542–7. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btq169.

[53] Massimo Pigliucci. "Genotype-phenotype mapping and the end of the 'genes as blueprint' metaphor." In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 365.1540 (Feb. 2010), pp. 557–66. ISSN: 1471-2970. DOI: 10.1098/rstb.2009.0241.

[54] Stéphanie Rialle et al. "BioNetCAD: design, simulation and experimental validation of synthetic biochemical networks." In: *Bioinformatics (Oxford, England)* 26.18 (Sept. 2010), pp. 2298–304. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btq409.

[55] Guillermo Rodrigo and Alfonso Jaramillo. "Computational design of digital and memory biological devices." In: *Systems and synthetic biology* 1.4 (Dec. 2007), pp. 183–95. ISSN: 1872-5325. DOI: 10.1007/s11693-008-9017-0.

[56] Y Sakakibara et al. "Stochastic context-free grammars for tRNA modeling". In: *Nucleic Acids Res* 22.23 (Nov. 1994), pp. 5112–5120.

[57] Howard M Salis, Ethan a Mirsky, and Christopher a Voigt. "Automated design of synthetic ribosome binding sites to control protein expression." In: *Nature biotechnology* 27.10 (Oct. 2009), pp. 946–50. ISSN: 1546-1696. DOI: 10.1038/nbt.1568.

[58] D Searls. "Representing genetic information with formal grammars". In: *Proceedings of the 7th National Conference on Artificial Intelligence* (1988).

[59] D.B. Searls. "String variable grammar: a logic grammar formalism for the biological language of DNA". In: *The Journal of Logic Programming* 24.1-2 (1995), pp. 73–102.

[60] DB Searls. "The computational linguistics of biological sequences". In: *Artificial Intelligence and Molecular Biology*. 1993, pp. 47–120.

[61] D.B. David B Searls. "Linguistic approaches to biological sequences". In: *Bioinformatics* 13.4 (1997), p. 333. ISSN: 1367-4803.

[62] Paul Shannon et al. "Cytoscape: a software environment for integrated models of biomolecular interaction networks". In: *Genome research* 13.11 (2003), pp. 2498–2504.

[63] Nayanah Siva. "1000 Genomes project." In: *Nature biotechnology* 26.3 (Mar. 2008), p. 256. ISSN: 1546-1696. DOI: 10.1038/nbt0308-256b.

[64] J W Szostak, D P Bartel, and P L Luisi. "Synthesizing life." In: *Nature* 409.6818 (Jan. 2001), pp. 387–90. ISSN: 0028-0836. DOI: 10.1038/35053176.

[65] J M Temkin and M R Gilder. "Extraction of protein interaction information from unstructured text using a context-free grammar". In: *Bioinformatics* 19.16 (Jan. 2003), pp. 2046–2053. DOI: 10.1093/bioinformatics/btg279.

[66] E. N. Trifonov. "The multiple codes of nucleotide sequences". In: *Bulletin of Mathematical Biology* 51.4 (July 1989), pp. 417–432. ISSN: 0092-8240. DOI: 10.1007/BF02460081.

[67] J C Venter et al. "The sequence of the human genome." In: *Science (New York, N.Y.)* 291.5507 (Feb. 2001), pp. 1304–51. ISSN: 0036-8075. DOI: 10.1126/science.1058040.

[68] Alan Villalobos et al. “Gene Designer: a synthetic biology tool for constructing artificial DNA segments.” In: *BMC bioinformatics* 7 (2006), p. 285. ISSN: 1471-2105. DOI: 10.1186/1471-2105-7-285.

[69] Emma Weeding, Jason Houle, and Yiannis N Kaznessis. “SynBioSS designer: a web-based tool for the automated generation of kinetic models for synthetic biological constructs”. In: *Briefings in bioinformatics* 11.4 (2010), pp. 394–402.

[70] Albert H C Wong, Irving I Gottesman, and Arturas Petronis. “Phenotypic differences in genetically identical organisms: the epigenetic perspective.” In: *Human molecular genetics* 14 Spec No.1 (Apr. 2005), R11–8. ISSN: 0964-6906. DOI: 10.1093/hmg/ddi116.

Chapter 2

Synthetic Biology Open Language (SBOL) Version 1.1.0

Published in:

Galdzicki, M., Wilson, M. L., Rodriguez, C. A., Pocock, M. R., Oberortner, E., Adam, L., Adler, A., et al. (2012). Synthetic Biology Open Language (SBOL) Version 1.1.0. BBF RFC 87 2012. p. 1-26. http://dspace.mit.edu/handle/1721.1/73909

2.1 Motivation

Synthetic biology is an engineering discipline where biological components, usually in the form of segments of DNA, are assembled to form devices and systems with more complex functions. A number of software tools have been developed to help synthetic biologists to design, optimize, validate and share these DNA systems, but the lack of a defined information standard for synthetic biology makes it extremely difficult to combine the appropriate applications into a refined systems process. To move the synthetic biology field towards best practices in engineering, synthetic biologists need software that can unambiguously interpret and exchange information about DNA components.

2.1.1 Computer Exchange Format

The lack of a standard exchange format means that synthetic biology information access and transfer is limited to manual efforts such as copy-and-paste and ad hoc scripts. Not only are these prohibitively lengthy approaches to data transfer, they can also be error-prone,

either due to changes in the underlying architectures of the data sources or simple human error. Establishing a standard exchange format would not only save time, but would also help reduce the errors of manual transfer.

A standard exchange format would also provide a greater range of tools available to synthetic biologists. Although there is a wide variety of software tools out there, the difficulty inherent to designing interfaces between them sometimes makes it more effective for software developers to write their own applications rather than take advantage of something already out there; a standard exchange format would alleviate their need to develop interfaces or duplicate software, which in turn would free them up to develop new tools. Furthermore, if a standard format enabled programmatic access to public information resources [Galdzicki et al. 2011], such as the Registry of Standard Biological Parts (`http://partsregistry.org`) and the BIOFAB Electronic Datasheets (`http://biofab.org/data`), software developers would be able to take advantage of these repositories directly within their applications. For example, CAD and modeling tools, such as TinkerCell (`http://tinkercell.com/`) and iBioSim (`http://www.async.ece.utah.edu/iBioSim/`), would be able to retrieve components for new designs. A gene network design created by tools such as the Proto Biocompiler (Beal 2011) could be further refined by Eugene into collections of physical implementations [Bilitchenko et al. 2011].

In addition to improving the ability to share data across applications, a standard format would make it easier for synthetic biologists to exchange data with their collaborators at other sites. For example, synthetic biologist researchers could use software such as GD-ICE (`http://code.google.com/p/gd-ice`) and Clotho (http://clothocad.org) to integrate their own data from local laboratory repositories with their collaborators' designs and publicly available data. A synthetic biologist who designed new DNA constructs with a software tool such as GenoCAD (`http://www.genocad.org`) could send them to a collaborator who would then review and edit them using Gene Designer (`https://www.dna20.com/tools/genedesigner.php`).

The definition of a standard for electronic information exchange would also help the refinement of standards for the DNA components themselves, through an iterative process of feedback to synthetic biology research groups concerned with standardization.

In summary, a standard exchange format would encourage reuse of existing DNA components; it would reduce error caused by manual or ad hoc data exchange; it would aid in collaboration between researchers; and it would save time which could then be used for advancing research and the development of new software tools.

Electronic exchange of synthetic biology information in a common format and using a common vocabulary will encourage the creation of interoperable software to support the information needs of synthetic biologists. Software developers will be able to write fewer data converters and offer access to a larger number of data sources. Finally, compatibility with Semantic Web information technology will serve as leverage for the software developed by this broad community, as it will facilitate reuse of previously generated knowledge across independent research efforts.

To establish such an information standard, we have proposed the Synthetic Biology Open

Language (SBOL) as a launching point for a community development effort. As software tools are adapted to progress in the synthetic biology field, we expect SBOL to evolve to meet the needs of synthetic biology researchers and engineers.

2.2 Introduction to SBOL

The first version of the Synthetic Biology Open Language (SBOL) is defined to meet the information exchange needs of synthetic biologists. SBOL is composed of:

- The **core data model**, to structure the information used to describe DNA designs,
- and the **vocabulary**, which defines the terminology used in the data model.

To encourage expansion of this standard's capabilities, the guiding principle is openness. Therefore SBOL allows and expects extensions to version 1.1.0, proposed by the community. This collaboration in defining the common information exchange framework is driven by the community of researchers participating in the Synthetic Biology Data Exchange Group (`http://sbolstandard.org/`). The goal of this specification is to define the terminology and relationships used to describe DNA designs. In order to provide a shared understanding between engineers seeking to exchange DNA designs, SBOL provides a common definition of the concepts needed. As much as possible, we attempt to make explicit the meaning of all terminology and data structures.

2.2.1 Scope

Version 1.1.0 of the SBOL core data model is limited to the description of discrete segments of DNA, called DNA components. To remove ambiguity when specifying the design of synthetic DNA, the information about DNA components is structured. DNA components described using the SBOL core data model may have an associated DNA sequence, or may be left as abstract descriptions. This flexibility allows for the description of DNA component designs which have not yet been realized, as well as those which are specific implementations of that design.
This version does not, however, provide a mechanism to represent the biological complexity of complete cellular systems beyond the DNA level. Additional biological knowledge needed to engineer aspects of complete biological systems will be modeled by future SBOL extensions. Furthermore, extensions of SBOL may add the ability to describe DNA components before and after a process, such as assembly, evolution, or implementation of a design *in silico*. Existing tools, such as GenoCAD, Eugene, and TASBE already offer solutions for bridging the "before and after", so they can provide a basis for future specifications.

2.2.2 Abstraction Level

Within SBOL, we consider DNA regions as elements of design for DNA circuits [Savageau 2001], analogous to electrical circuits [Kaern, Blake, and Collins 2003]. This conceptualization of DNA segments as an element of design is a level of abstraction used to form the basis of engineering synthetic biological systems [Endy 2005a]. This level of abstraction (Figure 2.1) has been shown to be useful in the practice of forward engineering of biological systems [Nandagopal and Elowitz 2011]. Therefore, we define these elements as DNA Components (Figure 2.1) in the SBOL vocabulary, and represent them as computational objects in the SBOL core data model.
DNA Components form the basic objects used in design, assembly, testing, and analysis. For example, DNA components can be hypothesized to have a biological function, be deemed necessary for DNA assembly processes, or have served as landmarks in analysis.

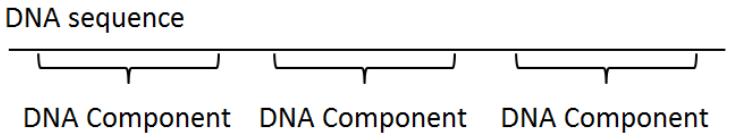


Figure 2.1: The basic abstraction level of identified DNA sequence segments as DNA Components.

2.2.3 Examples of Simple DNA Components

An example of the design of an expression cassette is shown in Figure 2.2; it illustrates DNA components along a DNA sequence. The symbols used represent their role in gene expression.

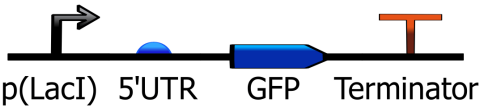


Figure 2.2: Example visualization of a series of DNA components. It includes a promoter, a 5'UTR, a CDS, and a Terminator. Together, these DNA components constitute the design of the expression cassette which expresses GFP. The iconography used represents the types of these DNA components (see SBOL:Visual for more on this extension `http://www.sbolstandard.org/initiatives/sbol-visual`). Individual icons are labeled with an informal name for the specified DNA component.

An example DNA construct which fulfills the design specified in Figure 2 is the BioBrick™

Part BBa_J04430 (`http://partsregistry.org`) (Figure 2.3). This example illustrates the representation of a DNA construct as a DNA component with annotations.

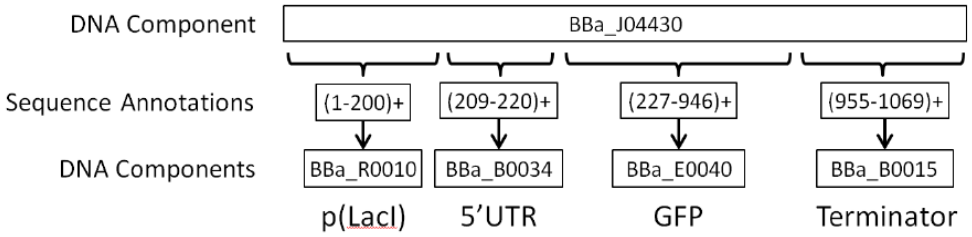
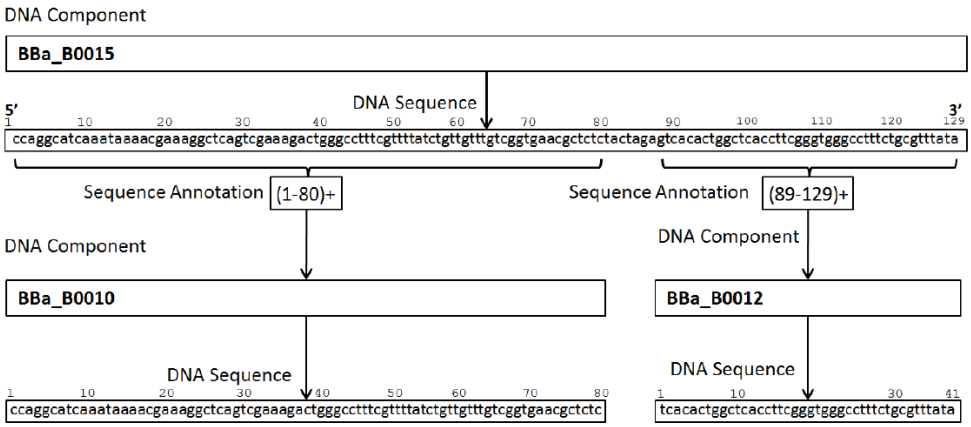


Figure 2.3: A diagram of BioBrick^{TM} BBa_J04430 represented by SBOL objects. Sequence annotations of BBa_J04430 are used to describe the location of DNA components that are found within its sequence. These annotations are DNA components which correspond to the design specified in Figure 2.2.

Each DNA component can be further described with additional information (2.4).



(a)

DNA Component

display ID:	BBa_B0015
name:	B0015
description:	double terminator (B0010-B0012)
type:	http://identifiers.org/obo.so/SO:0000167
uri:	http://partsregistry.org/Part:BBa_B0015

(b)

Figure 2.4: SBBa_B0015, a sub-component of BBa_J04430. (a) With DNA sequence and sequence annotations. (b) To describe BBa_B0015 in more detail a set of fields for a human readable ID, name, and text description is defined. Structured information fields will enable basic retrieval capabilities ie. type using the Sequence Ontology [Eilbeck et al. 2005; Mungall, Batchelor, and Eilbeck 2011]; and uri as a unique identifier.

2.3 Description of SBOL

SBOL version 1.1.0, focuses on the representation of DNA components as the SBOL:Core:model. SBOL:Vocabulary defines the key terms used in the core model.

2.3.1 Overview of SBOL

2.3.1.1 SBOL Vocabulary

In version 1.1.0, SBOL:Vocabulary defines the core concepts used by SBOL in the structured description of synthetic DNA designs. SBOL:Core terms are *DnaComponent*, *DnaSequence*, and *SequenceAnnotation*. To provide user-defined groupings of *DnaComponents*, SBOL:Core also defines a *Collection*. Additional terms, such as those used as used to classify DNA Components by type (see Section 2.3.5.1) are defined by the Sequence Ontology [Eilbeck et al. 2005; Mungall, Batchelor, and Eilbeck 2011]. New terminology should be added in collaboration with the Sequence Ontology project.

2.3.1.2 SBOL Core Data Model

To enable electronic exchange of information, SBOL:Core:model defines the data model describing the DNA sequence and groupings of components used for biological engineering. The model specifies the object and data properties associated with instances of the concepts defined by SBOL:Core terms. SBOL:Core:model represents a consensus of the minimal information needed to describe DNA sequences used in synthetic biological designs.

2.3.2 Conventions

The project to define SBOL is comprised of constitutive parts. The convention for naming them takes the form SBOL:Specification Part Name[:sub specification]; for example, SBOL:Core:model is used to denote the core data model part of the SBOL as it uses the portion of SBOL:Vocabulary called SBOL:Core and specifies the data model to structure it. SBOL:Vocabulary terms are italicized in the remainder of this document to distinguish them. SBOL:Core:model Class names are written in upper *CamelCase*, starting with an upper case letter. For example, *DnaComponent* is the class or type of object that represents a 'DNA Component'. Field or property names defined within classes follow lower *camelCase*. For example, bioStart is a field that specifies the position of the first base pair of a *SequenceAnnotation*.

Instances of SBOL:Core:model classes are written as abbreviations.

Abbreviation key:

- $DC_{\varnothing}$ – a DnaComponent w/o type, w/o sequence
- DC_t – a DnaComponent w/ type
- DC_s – a DnaComponent w/ sequence

- DC_{st} – a DnaComponent w/ sequence and type
- SA_{posN} – a SequenceAnnotation w/ position coordinates [N-ordinal notation only]
- SA_{rpN} – a SequenceAnnotation w/ relative position (precedes)
- $SA_{\varnothing}$ – a SequenceAnnotation w/ position unspecified
- Col – a Collection

The terms "MUST", "REQUIRED", "SHALL", "MUST NOT", "SHALL NOT", "SHOULD", "RECOMMENDED", "SHOULD NOT", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BBF RFC 0]. The following URI namespaces are defined as prefix: "namespace": sbol: "`http://sbols.org/v1#`" xsd: "`http://www.w3.org/2001/XMLSchema#`"

2.3.2.1 SBOL versions and releases

SBOL follows a version strategy inspired by Semantic Versioning (`http://semver.org`) and simplified for SBOL as a specification of a standard in contrast to a public API. SBOL version numbers MUST take the form X.Y.Z where X, Y, and Z are integers. X is the major version, Y is the minor version, and Z is a patch version. Each element MUST increase numerically. For instance: $1.9.0 < 1.10.0 < 1.11.0$. Prior to version 1.0.0 a less formal version system was in place. Going forward, the following strategy MUST be followed.

- Major versions (X) correspond to releases of SBOL, the submission of the specification document to the BioBrick Foundation as a Request For Comment (BBF RFC) is REQUIRED.
- Minor versions (Y) correspond to revisions to the specification as approved by the SBOL Forum, as defined by the SBOL governance document (`http://www.sbolstandard.org/sbol-governance`), and confirmed by the voting process. The submission of the specification document to the BioBrick Foundation as a Request For Comment (BBF RFC) is OPTIONAL.
- Patch versions (Z) correspond to draft revisions made by the SBOL Editors during the specification process. The submission of the specification document to the BioBrick Foundation as a Request For Comment (BBF RFC) is NOT RECOMMENDED.

The SBOL definitions that follow this section conform to these conventions.

2.3.3 SBOL vocabulary

The SBOL:Vocabulary defines terms used in SBOL. Below we define terms for the Core. Non-SBOL term definitions such as those needed to classify DnaComponents by type can be obtained from the Sequence Ontology [Eilbeck et al. 2005; Mungall, Batchelor, and Eilbeck 2011]. For example, a promoter region, coding sequence, and transcriptional terminator are all defined by the Sequence Ontology (see the Appendix A for a list of examples). Terminology outside of the scope of the Sequence Ontology should be submitted as new term requests to its curators (`http://www.sequenceontology.org/`).

2.3.3.1 Core

The SBOL:Core terms are defined to be used as concepts common to descriptions of DNA sequence designs in synthetic biology.

- **DNA Component**

  A DNA component represents a segment of DNA that serves to abstract the DNA sequence as an individual object, which can then be manipulated, combined, and reused in engineering new biological systems.

  SBOL name: *DnaComponent*
  SBOL URI: `http://sbols.org/v1\#DnaComponent`
  Nominal definition: [Standard] Biological Part [Endy 2005b; Shetty et al. 2008], BioBrick™ Part, DNA Part

- **DNA Sequence**

  The DNA sequence is a contiguous sequence of nucleotides. The sequence is a fundamental information object for synthetic biology and is needed to reuse components, replicate synthetic biology work, and to assemble new synthetic biological systems. Therefore, both experimental work and theoretical sequence composition research depend heavily on the exact base pair sequence specification associated with DnaComponents.

  SBOL name: *DnaSequence*
  SBOL URI: `http://sbols.org/v1\#DnaSequence`
  Nominal definition: The deoxyribonucleic acid sequence, primary structure of DNA

- **Sequence Annotation**

  The sequence annotation is the position and strand orientation of a notable subsequence found within the DnaComponent being described. Annotations provide the link which describes the DNA sequence of a component in terms of other components,

subComponents. When a DNA component is abstract, SequenceAnnotations specify relative positions between subComponents.

SBOL name: *Collection*
SBOL URI: `http://sbols.org/v1\#Collection`

Nominal definition: position, location, order [of subComponents]

- **Collection**

  A collection is an organizational container, a group of DnaComponents. For example, a set of restriction enzyme recognition sites, such as the components commonly used for BBF RFC 10 BioBricks^{TM} could be grouped. A Collection might contain DNA components used in a specific project, lab, or custom grouping specified by the user.

  SBOL name: *SequenceAnnotation*
  SBOL URI: `http://sbols.org/v1\#SequenceAnnotation`
  Nominal definition: Set, Collection, Bag of Parts, Library

2.3.4 Definition of the SBOL Core Data Model

This section defines the structure of SBOL:Core:model. In Figure 2.5, the UML class diagram notation is used to describe the SBOL:Core:model classes, their properties, and the main associations between classes. 2.4 provides complete examples encoded in SBOL. There are four classes in the data model, DnaComponent, DnaSequence, SequenceAnnotation, and Collection, which correspond to the four concepts needed to unambiguously describe the DNA design of synthetic biological systems. Each instance of a DnaComponent class refers to an actual or planned DNA component. When using SBOL to describe information about DNA components, an instance of the DnaComponent class MUST be created. The DnaComponent instance MAY specify an associated DnaSequence instance that it pertains to, and MAY be described using SequenceAnnotation instances to specify the position of subcomponents (DnaComponent). Collection instances MAY have associated DnaComponent instances. These concepts are illustrated in Figure 2.5.

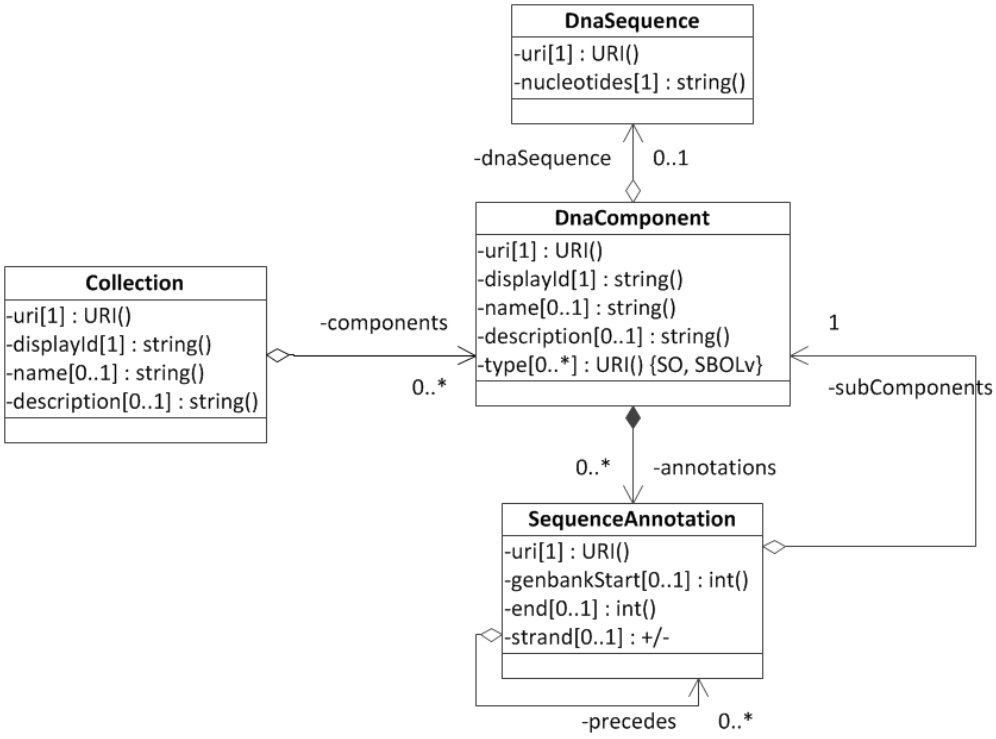


Figure 2.5: The SBOL core data model. The SBOL core data model is specified using a UML 2.0 diagram (`http://www.omg.org/spec/UML/2.0/`). Classes (rectangles) are named at the top and connected by associations (arrows). Each association is labeled with its role name, and has a range type and a plurality, such as "exactly zero or one dnaSequence" $[0 \cdots 1]$ or "one and only one subComponent" [1]. These can be interpreted as Sets of objects which are instances of the Class specified. An arrowhead indicates that the association can be traversed in that direction. Diamonds classify the association; open-faced diamonds are shared aggregation, meaning the object at the end of the arrow can exist independently of the source object, and filled diamonds indicate composite aggregation, or a part-whole relationship, which means that a part instance must be included in at most one whole and cannot exist independently. Data properties for objects of each class are listed in a separate compartment below, with the cardinality and corresponding data types specified.

2.3.5 SBOL Core Model Classes

We define each class individually and specify requirements for their intended use in the SBOL:Core data model.

2.3.5.1 DnaComponent

Objects representing a DNA component MUST be instances of the class sbol:DnaComponent, defined as:

- **Class:** DnaComponent
- **Context:** Instances of the DnaComponent class represent segments of DNA as defined by sbol:DnaComponent. The DnaComponent's DNA sequence CAN be annotated using SequenceAnnotation instances, positionally defined descriptors of the sequence which specify additional DnaComponent instances as subComponents. A DnaComponent MAY specify one DnaSequence instance it abstracts. DnaComponent instances MAY be grouped into Collections.

A DnaComponent instance MUST have the following REQUIRED data properties.

- **Data properties:**

  uri: (required) One and only one field of type URI (IETF RFC 2396). This property uniquely identifies the instance, and is intended to be used whenever a reference to the instance is needed, such as when referring to a DnaComponent stored on a server from another location. This URI MAY be a fully qualified URI (e.g. `http://sbols.org/data#P0123`), which then MUST remain constant forever, or this URI MAY consist of a relative identifier only (e.g. #P0123), which then MUST be unique within the enclosing context (e.g. Collection, an XML document). This form MAY be used for the informal exchange of free-floating temporary SBOL instances. For any other purpose it is RECOMMENDED that the relative identifier can be resolved against the enclosing context (e.g. the uri of a Collection or the fixed address of an XML document) into a fully qualified URI which then, again, MUST remain constant forever.

  displayId: (required) One and only one field of type xsd:string starting with a letter or underscore followed by only alphanumeric and underscore characters. The displayId is a human readable identifier for display to users. For example, users could use this identifier in combination with the namespace of the source as an unambiguous reference to the DNA construct.

OPTIONALLY a DnaComponent instance MAY have the following RECOMMENDED data and object properties.

- **Object properties:**

dnaSequence: (optional) Zero or one value of type DnaSequence. This property specifies the DNA sequence which this DnaComponent object represents. See also: DnaSequence.

annotations: (optional) Zero or more values of type SequenceAnnotation. This property links to SequenceAnnotation instances, each of which specifies the position and strand orientation of a DnaComponent that describes a subComponent of this DNA component.

- **Data properties:**

  name: (optional) Zero or one value of type xsd:string. The name of the DNA component is a human- readable string providing the most recognizable identifier used to refer to this DnaComponent. It often confers meaning of what the component is in biological contexts to a human user. A name may be ambiguous, in that multiple, distinct DnaComponents may share the same name. For example, acronyms are sometimes used (eg. pLac-O1) which may have more than one instantiation in terms of exact DNA sequence composition. As these names are intended for human consumption, they SHOULD be kept short and meaningful, as may be done by using an acronym, or re-using names that have commonly been used in literature.

  description: (optional) Zero or one value of type xsd:string. The description is a free-text field that contains text such as a title or longer free-text-based description for users. This text is used to clarify what the DnaComponent is to potential users (eg. engineered Lac promoter, repressible by LacI). The description could be lengthy, so it is the responsibility of the user application to format and allow for arbitrary length.

  type: (optional) Zero or more values of type URI (IETF RFC 2396) referencing the Sequence Ontology (see Appendix A for commonly used terms). These provide a defined terminology of types of DnaComponents. This vocabulary may be extended, (use "Request a Term" `http://sequenceontology.org`).

2.3.5.2 DnaSequence:

Objects representing a DNA Sequence MUST be instances of the class sbol:DnaSequence, defined as:

- **Class:** *DnaSequence*
- **Context:** Instances of the DnaSequence class contain the actual DNA sequence string. This specifies the sequence of nucleotides that comprise the DnaComponent being described. For SBOL DnaSequence, the base pairs MUST be represented by a sequence of lowercase characters corresponding to the 5' to 3' order of nucleotides in the DNA segment described, eg. "actg". The string value MUST conform to the restrictions listed below:

a. The DNA sequence MUST use the Nomenclature for incompletely specified bases in nucleic acid sequences [Cornish-Bowden 1985]. Rules adopted by IUPAC.

Symbol	Meaning
a	a; adenine
c	c; cytosine
g	g; guanine
t	t; thymine
m	a or c
r	a or g
w	a or t
s	c or g
y	c or t
k	g or t
v	a or c or g; not t
h	a or c or t; not g
d	a or g or t; not c
b	c or g or t; not a
n	a or c or g or t

b. Blank lines, spaces, or other symbols must not be included in the sequence text.

c. The sequence text must be in ASCII or UTF-8 encoding. For the alphabets used, the two are identical.

A DnaSequence instance MUST have the following REQUIRED data properties.

- **Data properties:**

  uri: (required) One and only one field of type URI (IETF RFC 2396). This property uniquely identifies the instance, and is intended to be used whenever a reference to the instance is needed. See also DnaComponent.uri

  nucleotides: (required) One and only one value of type xsd:string. See requirements for value of string in the class definition.

2.3.5.3 SequenceAnnotation:

Objects representing a Sequence Annotation MUST be instances of the class sbol:SequenceAnnotation, defined as:

- **Class:** SequenceAnnotation

- **Context:** Individual instances of the SequenceAnnotation class specify a relationship of the subComponent (DnaComponent) to the DnaComponent being annotated, its 'parent' DnaComponent. This relationship is the position and strand orientation of the subComponent relative to the parent DnaComponent orientation. The SequenceAnnotation location CAN be specified by the bioStart and bioEnd positions of the subComponent, along with the DNA sequence. Alternatively, the partial order of SequenceAnnotations along a DnaComponent can be specified by indicating the precedes relationship to other SequenceAnnotations. As a convention, numerical coordinates in this class use position 1 (not 0) to indicate the initial base pair of a DNA sequence. This convention is followed by the broader Molecular Biology community, especially in the relevant literature. The strand orientation, or direction, of the subComponent's sequence relative to the parent DnaComponent is specified by the strand [+/-]. For strand: '+' the sequence of the subComponent is the exact sub-sequence, and for '-' it is the reverse-complement of the parent DnaComponent's sequence in that region. A SequenceAnnotation instance MUST specify a subComponent of type DnaComponent. SequenceAnnotations MUST belong to exactly 1 DnaComponent.

A SequenceAnnotation instance MUST have the following REQUIRED object properties.

- **Object properties:**

  subComponent: (required) One and only one value of type DnaComponent. This property specifies the DnaComponent which is being annotated on the parent DnaComponent's sequence. Analogous to 'a feature' in other systems, the DnaComponent value serves to indicate information about the sequence at the position specified by the SequenceAnnotation's location data properties or the relative position object property. The sequence of the DnaComponent specified by the subComponent property MUST be logically consistent with the strand value.

- **Data properties:**

  uri: (required) One and only one field of type URI (IETF RFC 2396). This property uniquely identifies the instance, and is intended to be used whenever a reference to the instance is needed. see also DnaComponent.uri

A SequenceAnnotation instance MAY have one of the following Location Data or Relative Position Object property groups.

Location Data Group

- **Data properties:**

  bioStart: (optional) Zero or one value of type xsd:positiveInteger. Positive integer coordinate of the position of the first base of the subComponent on the DnaComponent. bioStart coordinate is relative to the parent sequence.

bioEnd: (optional) Zero or one value of type xsd:positiveInteger. Positive integer coordinate of the position of the last base of the subComponent on the DnaComponent. bioEnd coordinate is relative to the parent sequence.

strand: (optional) Zero or one value of type xsd:string. Strand orientation '+' or '-' is a 'subComponent relative to the parent DnaComponent orientation' flag. For a full explanation see Logical consistency of the subComponent's DnaSequence value.

Relative Position Object Group

- **Object property:**

  precdes: (optional) Zero or more values of type SequenceAnnotation. The precedes relation specifies the relative order of SequenceAnnotations for a given DnaComponent. It is a constraint on the order of subComponents when there is not enough information to specify exact positions. Precedes indicates the intended location by specifying that a SequenceAnnotation is to come before another when DnaSequence information becomes available. For example, you may want to say the promoter SequenceAnnotation precedes the CDS SequenceAnnotation, which precedes the terminator SequenceAnnotation. This ordering gives us the position, relative to other SequenceAnnotations (which can have a location or a relative position (using precedes)). During a validation process, the set of precedes relations on the SequenceAnnotation are required to be linearized to a sequence.

Well-formed constraint: Location Data

The Location Data fields of SequenceAnnotation are bioStart, bioEnd. They must be either, both present or absent. It is a well-formedness violation for one to be present while the other is absent.

Well-formed constraint: Relative Position

The Relative Position field of SequenceAnnotation is precedes. A relative position is valid if one of the following is true: precedes with a value of type SequenceAnnotation; or precedes is un-specified, for example when it is the last SequenceAnnotation of a linear sequence.

Logical consistency of Location Data

Given sa1:SequenceAnnotation, sa1.bioStart and sa1.bioEnd are logically consistent if: sa1.bioStart is greater or equal to 1 (positive integer); and sa1.bioEnd is greater or equal to sa1.bioStart; and the DNA sequence length of the DnaComponent specified by sa1.subcomponent is equal to $sa1.bioEnd - sa1.bioStart + 1$.

Logical consistency of Location Data and Relative Position

Given sa1:SequenceAnnotation and sa2:SequenceAnnotation, where sa1 precedes sa2, they are logically consistent if: absent data: sa1.bioEnd or sa2.bioStart are not provided; or

consistent data: both sa1.bioEnd and sa2.bioStart are provided and sa1.bioEnd is strictly less than sa2.bioStart

Logical consistency of the subComponent's DnaSequence value

When present, the subComponent's DnaSequence MUST relate to the exact sequence found in the interval specified by the Location Data of the SequenceAnnotation. The subComponent's sequence is logically consistent if the value of strand is: '+' and the subComponent's sequence specifies the sequence of the parent DnaComponent in that region. '-' and the subComponent's sequence specifies the reverse-complement of the parent DnaComponent's sequence in that region, see Section 2.4.4 Figure 2.10 for example. If the strand value is un-specified and the DnaSequence of the subComponent is present, the subComponent's sequence specifies the exact sub-sequence of the parent DnaComponent.

2.3.5.4 Collection:

Instances representing a Collection MUST be instances of the class sbol:Collection, defined as:

- **Class:** *Collection*
- **Context:** Individual instances of the Collection class represent an organizational container which helps users and developers conceptualize a set of DnaComponents as a group. Any combination of these instances CAN be added to a Collection instance, annotated with a displayID, name, and description and be published or exchanged directly. For example, a set of restriction enzyme recognition sites, such as the components commonly used for BBF RFC 10 BioBricks^{TM} could be placed into a single Collection. A Collection might contain DNA components used in a specific project, lab, or custom grouping specified by the user. Arbitrary groupings and new Collection instances SHOULD NOT be created and named when the groupings are not defined, but also Collections SHOULD NOT be created whenever an arbitrary set is possible. Instances should only be used to represent a grouping that is useful to a user.

A Collection instance MUST have the following REQUIRED data properties.

- **Data Properties:**

  uri: (required) One and only one field of type URI (IETF RFC 2396). This property uniquely identifies the instance, and is intended to be used whenever a reference to the instance is needed. see also DnaComponent.uri

  displayId: (required) One and only one value of type xsd:string starting with a letter or underscore followed by only alphanumeric and underscore characters. The

displayID is a human-readable identifier for display to users. For example, users could use this identifier in combination with the namespace of the source as an unambiguous reference to the DNA construct.

A Collection instance CAN have the following OPTIONAL object and data properties.

- **Object properties:**

  components: (optional) Zero or more instances of type DnaComponent which are members of this Collection and represent DNA segments for engineering biological systems. For example, standard biological parts, BioBricks, pBAD, B0015, BioBrick Scar, Insertion Element, or any other DNA segment of interest as a building block of biological systems.

- **Data Properties:**

  name: (optional) Zero or one value of type xsd:string. The common name of the Collection is the most recognizable identifier used to refer to this Collection. It SHOULD confer what is contained in the Collection. It is often ambiguous (eg. Mike's Arabidopsis Project A; Parts from Sleight, et al. (2010) J.Bioeng; BBF RFC 10 DNA Components; My Bookmarked Parts).

  description: (optional) Zero or one value of type xsd:string. Descriptions are a free-text field, therefore they SHOULD contain human-readable text describing the Collection for users to interpret what this Collection 'is'. This text SHOULD focus on an informative statement about the reason for grouping the Collection members. The result should allow users to interpret the reason for inclusion of members in this Collection (eg "Collecting parts which could be used to build honey production directly into mouse-ear cress"; "T9002 and I7101 variants from Sleight 2010, designs aim to improve stability over evolutionary time"; "Components useful when working with BBF RFC 10").

2.4 Examples

Sharing information about a variety of DnaComponents using the SBOL allows unambiguous specification of their DNA sequence-based descriptions. This section presents examples illustrative of different use cases as defined by SBOL:Core:model.

2.4.1 Annotated Composite *DnaComponent*

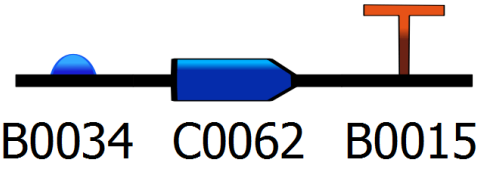
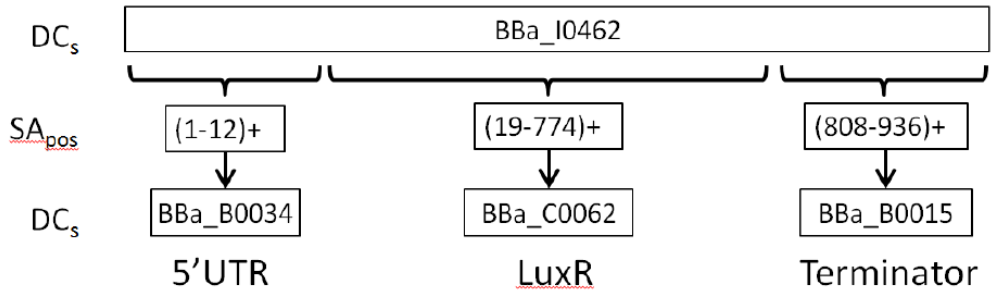


Figure 2.6: An example of a simple DNA design. BBa_I0462 (`http://partsregistry.org/Part:BBa_I0462`) drawn using SBOL:Visual symbols in TinkerCell (`http://tinkercell.com`) and composed of BBa_B0034, BBa_C0062, BBa_B0015 DnaComponents. The icons are labelled with a shorthand notation of the displayId from `http://partsregistry.org`.

The first example is the SBOL:Core:model for the BioBrick™ BBa_I0462. The BBa_I0462 DnaComponent codes for the LuxR protein when inserted downstream of a promoter (Figure 2.6). Information comes from the Registry of Standard Biological Parts (`http://partsregistry.org`) to describe this canonical composite BioBrick™ part.

In Figure 2.7-a the BioBrick™ part BBa_I0462, a DnaComponent, is depicted with annotations of three DnaComponents: a ribosome binding site (BBa_B0034), the coding sequence for LuxR (BBa_C0062), and a double terminator BBa_B0015. In Figure 2.7-b, the same DnaComponent is described using pseudocode as an example of SBOL:Core:model as text.




```
DnaComponent [
uri: http://partsregistry.org/Part:BBa\_I0462
displayId: BBa\_I0462 name: I0462
description: LuxR protein generator
annotations:[
SequenceAnnotation [
uri: http://sbols.org/anot\#1234567
bioStart: 1
bioEnd: 12
strand: +
subComponent:[
DnaComponent [
uri: http://partsregistry.org/Part:BBa\_B0034
displayId: BBa_B0034 name: B0034
type: ribosome\_entry\_site
]
]
]
SequenceAnnotation [
uri: http://sbols.org/anot\#2345678
bioStart: 19
bioEnd: 774
strand: +
subComponent:[
DnaComponent [
uri: http://partsregistry.org/Part:BBa\_C0062
displayId: BBa\_C0062
name: luxR
type: CDS
]
]
]
SequenceAnnotation [
uri: http://sbols.org/data\#3456789
bioStart: 808
bioEnd: 936
strand: +
subComponent:[
DnaComponent [
uri: http://partsregistry.org/Part:BBa\_B0015
displayId: BBa\_B0015
name: B0015
type: terminator
]
]
]
]
DnaSequence [
uri:
http://sbols.org/seq\#d23749adb3a7e0e2f09168
cb7267a6113b238973
nucleotides:
aaagaggagaaatactagatgaaaaacataaatgccgacg....
]
]
```

Figure 2.7: Annotated Composite DnaComponent. A diagram of the SBOL instances used to describe BBa_I0462. The gaps shown between the sequence annotations are unannotated segments of DNA. Pseudocode is used to demonstrate the use of core data model structure and data fields in a complete example of a DnaComponent.

2.4.2 Multi-Tiered Annotated *DnaComponent*

The next example depicts the composition of BBa_I0462 in terms of each of its subComponents (Figure 2.8).

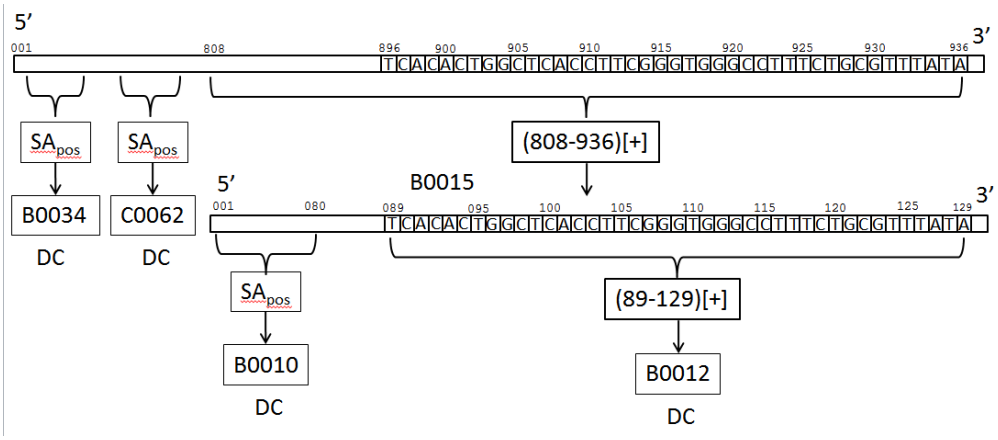


Figure 2.8: An expanded instance of BBa_I0462. It demonstrates key features of SBOL:Core:model. In this instance, the BBa_B0015 component of BBa_I0462 from the examples above is composed of two elements itself, BBa_B0010 and BBa_B0012. The letters of the DNA sequence in the top DnaComponents is omitted, so only the sequence corresponding to BBa_B0012 is shown.

2.4.3 Partially Realized Design Template

This example illustrates the partial specification of designs in terms of DnaComponent layout constraints. Figure 2.9 demonstrates the use of SequenceAnnotations with a Relative Position to specify the order of DnaComponents within a planned DnaComponent.

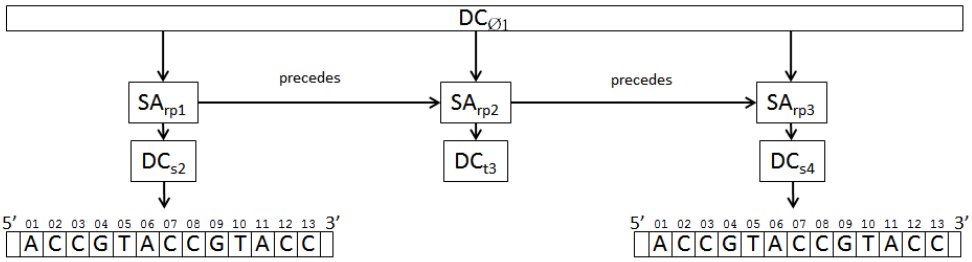


Figure 2.9: The design template for DnaComponent $DC_{\varnothing 1}$. It specifies that at least three DnaComponents must be present in this design. Their ordering is constrained, DC_{s2} precedes DC_{t3} and DC_{t3} precedes DC_{s4}. In this template the DC_{s2} and DC_{s4} already have a DnaSequence specified, however DC_{t3} does not, instead it specifies a type which it must me constrained to. Therefore, the DC_{t3} component can be filled in to match the type constraint later.

2.4.4 DnaSequence of subComponent on the minus strand

This example (Figure 2.10) demonstrates the reverse-complement relationship between the DnaSequence of the parent DnaComponent and its subcomponent annotated on the minus strand.

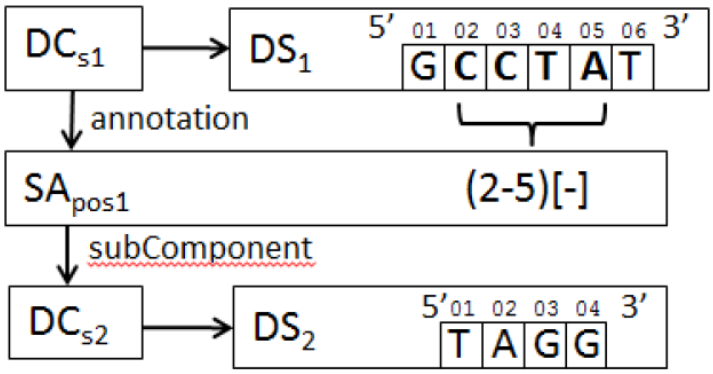


Figure 2.10: Reverse-complement relationship on the minus strand. The SequenceAnnotation's (SA_{pos1}) strand is specified as '-', the subComponent's (DC_{s2}) DnaSequence (DS_1) is the reverse-complement of the parent DnaComponent's (DC_{s1}) sequence (DS_2) in the annotated region.

To provide an organizational container for multiple DnaComponent instances, we provide the Collection class. The example in Figure 2.11 shows a Collection with multiple DnaComponents grouped together and ready to be shared between software applications.

2.4.5 Collection

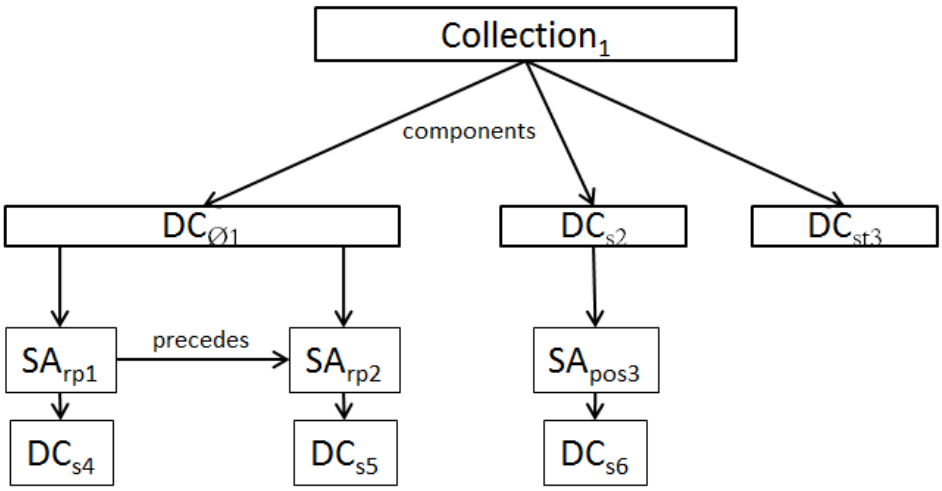


Figure 2.11: Collection. The Collection is a convenience object to group DnaComponents $DC_{\varnothing 1}$, DC_{s2}, and DC_{st3}. Described collections are a natural conceptualization of a group of objects to be shared at one time or that serve a specific purpose.

2.5 Serialization

Examples of serialization are maintained on the web, and will be updated as the libSBOL reference implementation (`http://github.com/synbiodex`) is finalized. (`http://www.sbolstandard.org/initiatives/serialization`)

2.6 Best Practices

For SBOL version 1.1.0, best practices are being maintained in a dynamic document on the web. (`http://www.sbolstandard.org/initiatives/best-practices`) In future versions Best Practices will be included in the specification.

Acknowledgments

This document is the result of discussions between participants in the Synthetic Biology Open Language Workshops held in Blacksburg, Virginia on January 7-10, 2011, San Diego,

California on June 8, 2011, and Seattle, Washington January 5-6, 2012. The ongoing work was finalized through email exchanges on the SBOL Developers mailing list through March 2012. We are indebted to the following individuals for their contribution in these discussions: Eduardo Abeliuk (Teselagen), Laura Adam (Virginia Bioinformatics Institute), Aaron Adler (BBN Technologies), J. Christopher Anderson (University of California, Berkeley), David A. Ball (Virginia Bioinformatics Institute), Jacob Beal (BBN Technologies), Swapnil Bhatia (Boston University), Michael Bissell (Amyris, Inc.), Matthieu Bultelle (Imperial College London), Yizhi Cai (Johns Hopkins University), Deepak Chandran (University of Washington), Joanna Chen (Lawrence Berkeley National Lab), Kevin Clancy (Life Technologies), Kendall G. Clark (Clark & Parsia, LLC.), Daniel Cook (University of Washington), Wilbert Copeland (University of Washington), Douglas Densmore (Boston University), Omri A. Drory (Genome Compiler, corp.), Drew Endy (BIOFAB and Stanford University), Michal Galdzicki (University of Washington), John H Gennari (University of Washington), Raik Gruenberg (IRIC, University of Montreal), Jennifer Hallinan (Newcastle University), Timothy Ham (Joint BioEnergy Institute), Nathan J. Hillson (Lawrence Berkeley National Lab), Cassie Huang (Boston University), Jeffrey D. Johnson (University of Washington), Marc Juul Christoffersen (BIOFAB), Kyung H. Kim (University of Washington), Richard Kitney (Imperial College London), Allan Kuchinsky (Agilent Technologies), Sung Won Lim (Genspace), Matthew W. Lux (Virginia Bioinformatics Institute), Curtis Madsen (University of Utah), Akshay Maheshwari (BIOFAB), Goksel Misirli (Newcastle University), Barry Moore (University of Utah), Chris J. Myers (University of Utah), Josh Natarajan (Autodesk Research), Ernst Oberortner (Boston University), Carlos Olguin (Autodesk Research), Jean Peccoud (Virginia Bioinformatics Institute), Josh Perfetto (Cofactor Bio, LLC.), Hector Plahar (Joint BioEnergy Institute), Darren Platt (Amyris, Inc.), Matthew Pocock (Newcastle University), Jackie Quinn (Harvard University), Sridhar Ranganathan (Life Technologies), Cesar A. Rodriguez (Genome Compiler, corp.), Nicholas Roehner (University of Utah), Vincent Rouilly (University of Basel), Herbert M. Sauro (University of Washington), Evren Sirin (Clark & Parsia, LLC.), Trevor F. Smith (Agilent Technologies), Lucian P. Smith (University of Washington), Guy-Bart Stan (Imperial College London), Jason Stevens (University of Utah), Vinod Tek (Imperial College London), Alan Villalobos (DNA 2.0, Inc.), Mandy Wilson (Virginia Bioinformatics Institute), Chris Winstead (Utah State University), Anil Wipat (Newcastle University), and Fusun Yaman Sirin (BBN Technologies).

References

[1] Lesia Bilitchenko et al. "Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems." In: *PloS one* 6.4 (Jan. 2011), e18882. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0018882.

[2] A Cornish-Bowden. "Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984." In: *Nucleic acids research* (1985).

[3] Karen Eilbeck et al. “The Sequence Ontology: a tool for the unification of genome annotations”. In: *Genome Biol* 6.5 (2005), R44.

[4] Drew Endy. “Foundations for engineering biology”. In: *Nature* 438.7067 (2005), pp. 449–53. ISSN: 1476-4687. DOI: 10.1038/nature04342.

[5] Drew Endy. “Foundations for engineering biology”. In: *Nature* 438.7067 (2005), pp. 449–453. ISSN: 1476-4687. DOI: 10.1038/nature04342.

[6] Michal Galdzicki et al. “Standard biological parts knowledgebase.” In: *PloS one* 6.2 (Jan. 2011), e17005. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0017005.

[7] Mads Kaern, William J Blake, and J J Collins. “The engineering of gene regulatory networks.” In: *Annual review of biomedical engineering* 5 (Jan. 2003), pp. 179–206. ISSN: 1523-9829. DOI: 10.1146/annurev.bioeng.5.040202.121553.

[8] Christopher J Mungall, Colin Batchelor, and Karen Eilbeck. “Evolution of the Sequence Ontology terms and relationships.” In: *Journal of biomedical informatics* 44.1 (Feb. 2011), pp. 87–93. ISSN: 1532-0480. DOI: 10.1016/j.jbi.2010.03.002.

[9] Nagarajan Nandagopal and Michael B Elowitz. “Synthetic biology: integrated gene circuits.” In: *Science (New York, N.Y.)* 333.6047 (Sept. 2011), pp. 1244–8. ISSN: 1095-9203. DOI: 10.1126/science.1207084.

[10] Michael a. Savageau. “Design principles for elementary gene circuits: Elements, methods, and examples.” In: *Chaos (Woodbury, N.Y.)* 11.1 (Mar. 2001), pp. 142–159. ISSN: 1089-7682. DOI: 10.1063/1.1349892.

[11] Shetty et al. “Engineering BioBrick vectors from BioBrick parts”. In: *Journal of biological engineering* 2 (2008), p. 5. DOI: 10.1186/1754-1611-2-5.

Chapter 3

A Step-by-Step Introduction to Rule-Based Design of Synthetic Genetic Constructs Using GenoCAD

Published in:

Wilson, M. L., Hertzberg, R., Adam, L., & Peccoud, J. (2011). A step-by-step introduction to rule-based design of synthetic genetic constructs using GenoCAD. Methods in enzymology, 498, 173–88. doi:10.1016/B978-0-12-385120-8.00008-5

Abstract

GenoCAD is an open source web-based system that provides a streamlined, rule-driven process for designing genetic sequences. GenoCAD provides a graphical interface that allows users to design sequences consistent with formalized design strategies specific to a domain, organization, or project. Design strategies include limited sets of user-defined parts and rules indicating how these parts are to be combined in genetic constructs. In addition to reducing design time to minutes, GenoCAD improves the quality and reliability of the finished sequence by ensuring that the designs follow established rules of sequence construction. GenoCAD.org is a publicly available instance of GenoCAD that can be found at www.genocad.org. The source code and latest build are available from SourceForge to allow advanced users to install and customize GenoCAD for their unique needs.

This chapter focuses primarily on how the GenoCAD tools can be used to organize genetic parts into customized personal libraries, then how these libraries can be used to design sequences. In addition, GenoCAD's parts management system and search capabilities are

described in detail. Instructions are provided for installing a local instance of GenoCAD on a server. Some of the future enhancements of this rapidly evolving suite of applications are briefly described.

3.1 Introduction

The vision of rationally designing synthetic biological systems has proved more elusive than anticipated [Kwok 2010]. The complexity of artificial gene networks has not made significant progress since 2006 [Purnick and Weiss 2009], which may indicate that the ad hoc processes used to develop proof-of-concept systems do not scale up well. The field still lacks a suitable framework to design more complex systems. Several authors have proposed to approach DNA sequences as a language to program biological systems [Clancy and Voigt 2010; Goler, Bramlett, and Peccoud 2008]. This idea may provide the foundation upon which it will be possible to develop computer-assisted design software applications for synthetic biology. A fast growing ecology of software tools to assist synthetic biologists in the development of new biological systems has been reviewed recently [Marchisio and Stelling 2009]. Gene Designer [Villalobos et al. 2006] is a stand-alone application with smooth graphical editor allowing users to drag and drop genetic parts into a larger DNA sequence. TinkerCell is another desktop application allowing users to design genetic constructs from standard parts and simulate the dynamics of the gene network they encode [Chandran, Bergmann, and Sauro 2009]. SynBIOSS is a web-based alternative to TinkerCell [Hill et al. 2008; Weeding, Houle, and Kaznessis 2010]. GEC [Pedersen and Phillips 2009] and Clotho (www.clothocad.org) are programming environments specifically designed for synthetic biology.
Like TinkerCell or Gene Designer, GenoCAD has a graphical user interface accessible to users without any programming experience. Instead of being a stand-alone application, GenoCAD is a database-driven web-based application [Czar, Cai, and Peccoud 2009]. Like Clotho and GEC, GenoCAD relies on a solid foundation derived from the theory of computer languages [Cai et al. 2007; Cai et al. 2009]. GenoCAD is an open source application distributed under the Apache software license. An instance of GenoCAD is available at www.genocad.org and is referred to as GenoCAD.org in this chapter to differentiate it from the GenoCAD software itself.

3.2 Overview of GenoCAD

Before building sequences in GenoCAD, it is helpful to understand the overall structure of the application and how the various pieces fit together to provide the user with a safe and streamlined design experience.
DNA sequences are made up of smaller standardized genetic DNA segments such as promoters, transcription terminators, genes, protein domains, and others. Within GenoCAD, these

segments are referred to as "parts." GenoCAD.org has a library with thousands of distinct basic parts [Cai, Wilson, and Peccoud 2010; Peccoud et al. 2008]. Users are not limited to the parts included in the global GenoCAD database. They can add new sequences in their personal workspace without having to make them available to other GenoCAD users. Design strategies composed of rules describing how parts can be combined are called grammars in GenoCAD.

The concept of a Design Strategy within GenoCAD is similar to the role a grammar plays within language. A writer may use a series of words that include a subject, a predicate, indirect objects, and prepositions, but if they fail to use the prescribed grammar for the language in question, the words may not come together to form a meaningful sentence. Design strategies in GenoCAD work much the same way. A design strategy uses rules to define which classes of parts, called categories, can be used to design a DNA sequence, and in what order they may appear. For example, the design of an E. coli gene expression cassette requires –at minimum– a Promoter, a Ribosome Binding Site (RBS), a Gene, and a Terminator. In GenoCAD, Promoters, RBS, Genes, and Terminators are categories, and the rules that ensure parts from each of the categories above are included define the E. coli design strategy. The design strategy ensures that categories may only be used in the appropriate order, so, in the example above, the RBS and Gene can only be inserted between a Promoter and Terminator. The rules of the design strategy also prevents parts from categories external to the E. coli design strategy from being included in the design.

Design strategies are currently coded within the GenoCAD database. That's where personal libraries come in. Libraries are named lists of parts that include only the parts the user wants to have available when creating DNA sequences for a specific project. Libraries are always design strategy-specific (e.g., an E. coli library may not contain Yeast parts), and work in conjunction with their design strategies to prevent user error during the design of a sequence. Personal libraries in GenoCAD can contain a combination of user-defined and global parts. GenoCAD Designs are DNA sequences that have been constructed using design strategies, libraries, and parts.

3.3 Requesting an Account on GenoCAD.org

When accessing GenoCAD.org for the first time, the first page presented is the Parts tab. Although most of the available features of GenoCAD.org may be viewed without logging in, many of them are disabled or have limited functionality for the unauthenticated user. To take full advantage of the features GenoCAD.org has to offer, a user account is required.

The link to apply for an account is located on the Log in tab. After loading the Log In page, the applicant would then click on the link, "Don't have an account? Request one." At this point, the Application for Account page is loaded in the browser (Figure 3.1).

The use of GenoCAD.org is free. However, in order to minimize the risk that this resource may be used to develop biological weapons, applications for account are reviewed to verify that the applicants can be identified and have legitimate needs to use the GenoCAD.org Web

site. More information on the guidelines GenoCAD.org uses for this validation is available from the GenoCAD Privacy Policy referenced at the top of the Application for Account page. Accordingly, the more information the applicant provides on the application, the more quickly his account can be validated; if the request can be verified from the information submitted in the initial application, the turn-around time for approval is less than 48 hours. Once the account is approved, the applicant receives an e-mail that includes a link to log into the system. Users can change their personal information or password by clicking on the My Profile link available in the submenu at the top of most of the pages in the system.
This user registration process is specific to GenoCAD.org because the resource is publicly available. Organizations installing GenoCAD on their own servers could most probably link GenoCAD to an existing user directory (LDAP, Active Directory).

Figure 3.1: Application for Account. In order to take full advantage of the site features, users of GenoCAD.org must request an account. The identity of the applicant is verified before the account is approved. Users are encouraged to fill out the application as completely as possible to expedite the review process.

3.4 Browsing the Parts Catalog

When logging into GenoCAD, the Parts tab, or parts listing, is the default page (Figure 3.2). The tabs along the top of the page guide the user to different features of the GenoCAD application, while the navigational menu on the left side of the screen contains functionality pertaining only to the Parts tab. The default navigational tab selected is Public Libraries. GenoCAD.org has thousands of public, or global, parts spread across four design strategies and a number of public libraries that users may choose from in developing their own personal libraries. GenoCAD offers two different options to aid users in finding parts.

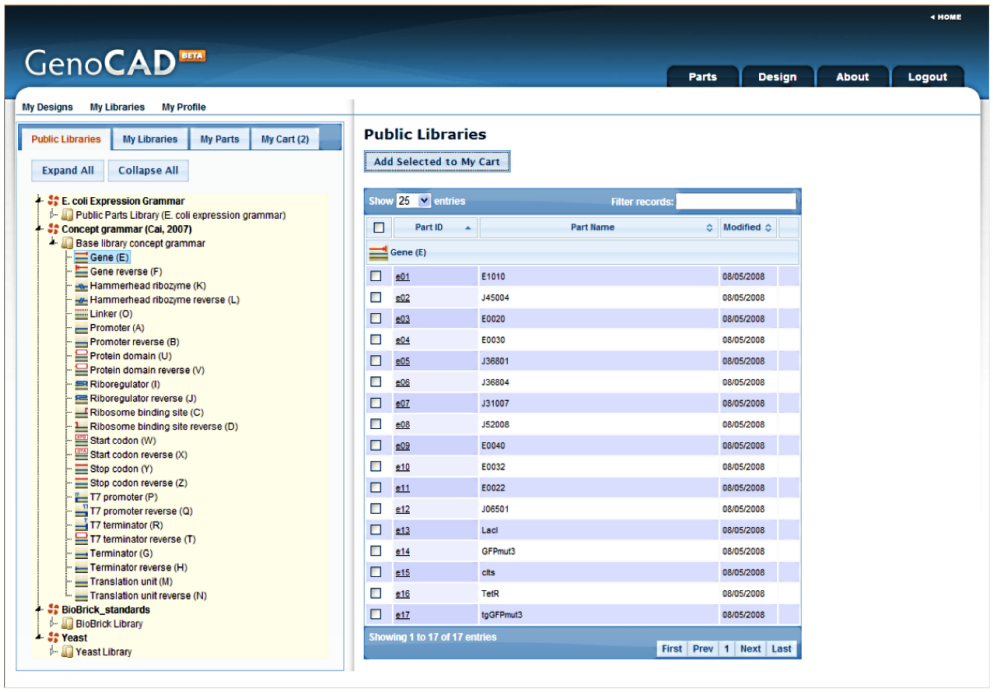


Figure 3.2: GenoCAD Parts Catalog. The Parts Catalog page is the first page users see upon logging into GenoCAD. In addition to providing a list of the parts available within the system, the Parts Catalog allows users to add their own parts to the system and to create libraries of parts that can be used to constrain the design process for specific projects.

Below the Public Libraries tab, there is a collapsible hierarchical menu that is made up of Design Strategies, Libraries, and Categories, although initially only design strategies (often called grammars) and libraries are displayed. In order to see which categories are represented beneath a specific library, the user can use the mouse to click on the small arrow next to the library in question to expand the list of categories below the selected library. Alternately, the Expand All button expands the menu to show the categories beneath all of the libraries, and the Collapse All button collapses the tree structure back to the design strategy/library levels.

To view the parts available under a given library or category, the user can click on the library or category name in the hierarchical menu on the left. The parts that fall under the selected library or category are displayed on the right side of the screen, sorted by category.The default number of parts shown is 25 parts per page; to see more of the parts for the selected library or category, the user can page through the parts list using the First, Prev, Next, Last, or page number links at the bottom of the page. The number of parts displayed per page can be changed by changing the value in the Show Entries drop down at the top of the page, where the options are 25, 50, 100, and All parts per page.
Once the parts are loaded on the right side of the page, they can be sorted by clicking on the column labels at the top of the screen. Additional information on some of the parts may be retrieved by hovering the mouse over the tool tip where applicable. Part details for a particular part may be viewed by clicking on the desired part's Part ID; the View Part screen displays a description of the part, what categories it falls under, what libraries it belongs to, and its DNA sequence. The Filter text box at the top of the listing allows the user to search for a text string contained within the Part ID, Name, or Description fields.

3.5 Searching for Parts

To look for parts by attribute rather than by browsing within an existing library, the user may do a site search for a part. The text box above the menu bars can be used for a quick text search; for example, if "Promoter" is entered in the text box, the search returns all of the global promoters, along with any promoters the requestor has entered into the system and owns as a user. Any text attribute of a part can be searched using the quick search, including portions of the part DNA segment. "Quick search" results are sorted by design strategy, library, and category, just as the Public Libraries parts are sorted.
For an even more complex text search, GenoCAD provides an Advanced Search option, available by clicking on a link at the top of the page. On the Advanced Search page, the user is able to build a query by selecting from a variety of attributes to limit the search results to more likely candidates. For example, to search for parts that contain the sequence "AGGA" and that are also Promoters, the drop down lists and text boxes may be used to create a query for parts where the sequence CONTAINS AGGA AND category IS EQUAL to Promoters. As in the Quick Search, results are sorted by grammar, library, and category, as they would by the Quick Search.
The Advanced Search also allows the user to do Basic Local Alignment Search Tool (BLAST) [Altschul 1990] sequence homology searches against the parts in the catalog to identify sequences that have regions of local similarity to the search sequence, including those that share functional or evolutionary relationships or are members of the same gene family. In this case, GenoCAD allows users to do a BLAST search that only includes results from the GenoCAD parts catalog. The Advanced Search also provides support for doing combined searches that include doing a text search upon the results of a BLAST search, although to the user, it appears as if the search was done in a single step; the searching algorithm

recognizes a combined search and handles the BLAST processing first, then applies the text filter.

3.6 Using My Cart to Create Libraries

As users find parts they are interested in including in their libraries, they may add them to their Cart (Figure 3.3). The paradigm here is similar to that of an online shopping cart –whereas on Amazon customers add books to a shopping cart and then order them all at the same time, the GenoCAD Cart serves as a temporary repository where parts of interest may be saved temporarily; when users are done looking for parts, the Cart to may be used to create or append to personal libraries.
To add a part from the parts listing or search results to the Cart, the mouse is used to select the checkbox beside the part or parts to include; to check all the parts on the page, the check box at the top of the list may be checked. When all the parts of interest on a particular page are checked, the Add Selected Parts toMy Cart button is used to add the parts to the user's cart.

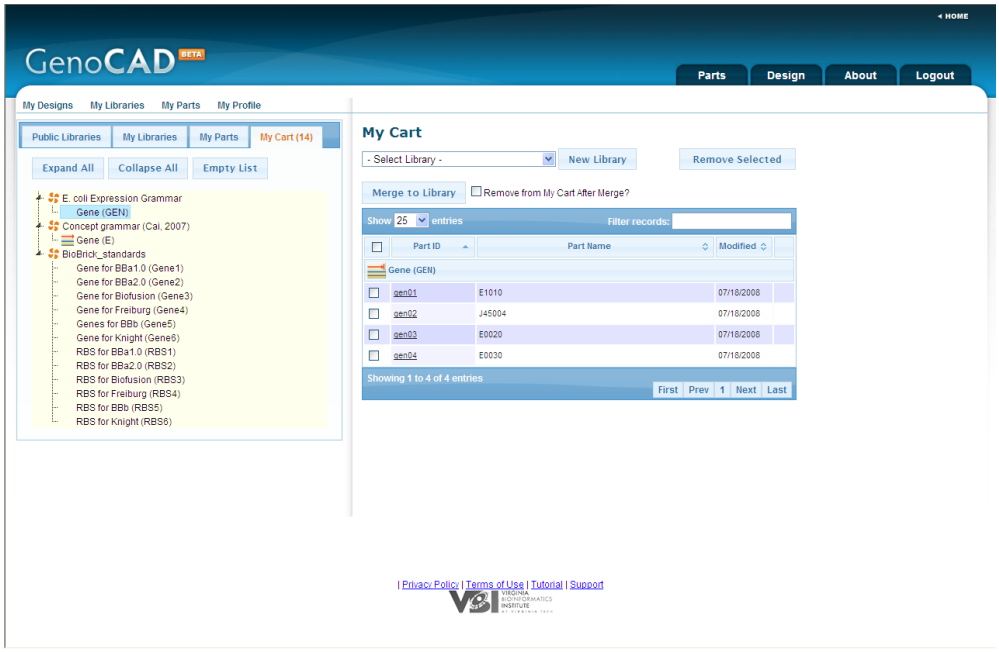


Figure 3.3: My Cart. The user can use the cart as a temporary listing of parts of interest that they may wish to use in designs. When the user is done selecting parts, they can merge parts in My Cart directly into personal libraries.

When the user is ready to create a library, he clicks on the My Cart tab from the left navigation menu. The menu hierarchy shown below the My Cart tab is divided into design

strategies and categories, but not libraries; parts are design strategy-specific, which means that parts from one design strategy cannot be used for a library from another design strategy. Before parts can be added to a library, the library must already exist. The New Library button pops up a window to allow the specifics of the new library to be defined –the design strategy this library adheres to, the name of the library, and the description. Once saved, the new library is added to the drop down list at the top of the screen as long as there are parts in My Cart that belong to the selected design strategy.
To merge parts from My Cart into the new library, the user starts out by selecting the design strategy or category from the hierarchical menu on the left that contains the parts to assign first. When the parts have loaded, as before, the user checks the specific parts to include from the right side of the screen; clicking the checkbox on the top of the page selects all the parts on that page. When finished selecting parts from this page, the target library is selected from the drop down on the top of the page and the Merge to Library button adds the parts to the selected library. If the Remove from My Cart? checkbox is checked, the selected parts are removed from My Cart as they are merged to the target library; if this checkbox is not checked, then the parts remain available for assignment to other libraries.
Users can remove unwanted parts from My Cart in a couple of ways. The Empty Cart button removes all of the parts from My Cart, and the Remove Selected button removes only the checked parts.

3.7 My Libraries

The My Libraries tab in the left navigation bar allows users to view their own personal libraries, either for editing library information, removing parts, or adding existing parts from their personal libraries to their Cart for use in a different library (Figure 3.4). The My Libraries view is very similar to the Public Libraries view, except that the libraries displayed there are the logged-in user's personal libraries, and My Libraries has additional functionality to allow users to manage their libraries and parts.

- New Library: As on My Cart, the user may add a new (empty) library from the left navigation bar under My Libraries.
- Management Console: Over the parts listing on the right side of the screen, there is a box that lists the name of the selected library and a Management Console link. Clicking this link exposes several additional options that can be used for managing libraries:
  - Add Selected to My Cart: This is identical to the corresponding button on the Public Libraries tab; it takes the selected parts and adds them to My Cart for assignment into other libraries.

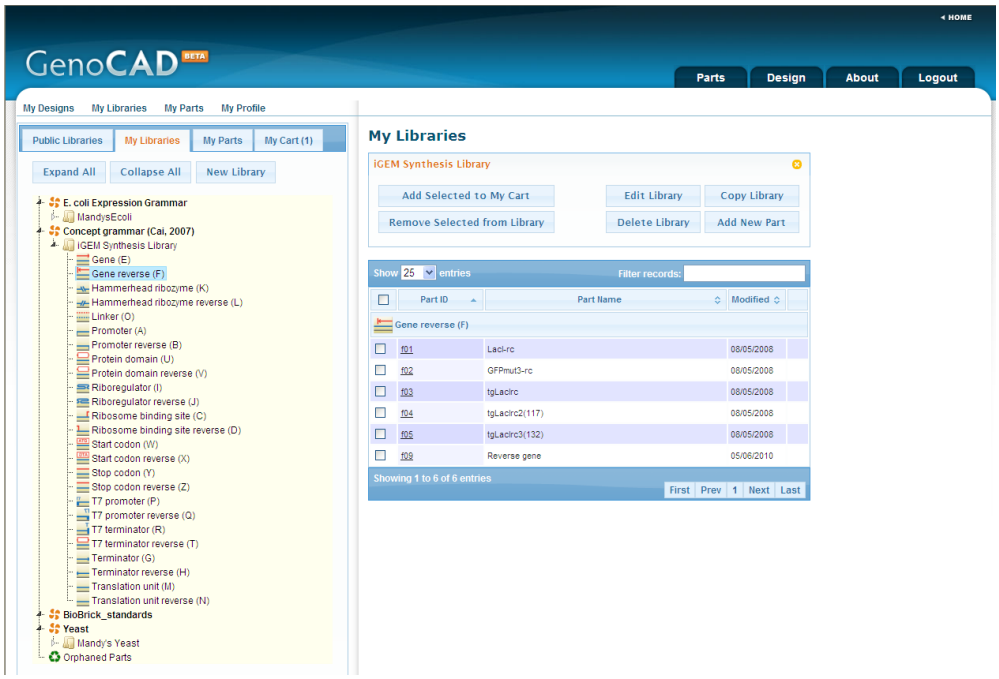


Figure 3.4: My Libraries View. This view allows the user to modify their libraries. From here they can manage their libraries, add new parts, and reassign orphaned parts.

- Remove Selected from Library: This removes the selected parts from the currently selected library.
- Edit Library: This allows the name or description of the library to be changed. Although the design strategy drop down is displayed on the Edit Library screen, it cannot be changed at this point. This is because parts, like libraries, are design strategy-specific, so if a library were able to change design strategies after it was established, there could be a large number of parts in that library which would not be available for designs.
- Delete Library: The Delete Library button can be used to remove libraries that were created by mistake or that are no longer needed. The Delete Library button assumes that the currently selected library is the one to delete. After prompting the user to verify the deletion, the library is deleted. The parts under the library, however, are not deleted, and are still available for assignment to other libraries. This is discussed in more detail under Orphaned Parts.
- Add New Part: If the necessary part cannot be found within the GenoCAD parts repository, the user can add his or her own part. The Add New Part button allows the user to enter a part name, a DNA sequence, and a description (Figure 3.5). After a design strategy/grammar is selected, the libraries available are limited to those that use that design strategy; new parts must be assigned to at least

one library. The design strategy selection also limits which categories may be selected for the new part; at least one category must be selected, but all categories that apply may be selected. Something to consider is that GenoCAD does not allow duplicate DNA segments (parts) per library, so if one part (sequence) can be used under multiple categories, then it should be mapped to all appropriate categories rather than loading multiple parts, one for each category. It should also be noted that users may only view global parts and the parts they have added, so a particular user's search results can include his/her own parts that meet the criteria, but cannot include parts added by other users.

- Orphaned Parts: At the bottom of the left navigation bar is an entry called "Orphaned Parts" that are indicated with a Recycling icon. Orphaned parts are always personal parts, and parts become orphaned when they are removed from all of their parent libraries. To remove a part from the Orphaned Parts library, it needs to be assigned to another library. Parts that remain orphaned for a configurable period of time, or a default of 2 months, are deleted by a cleanup process.

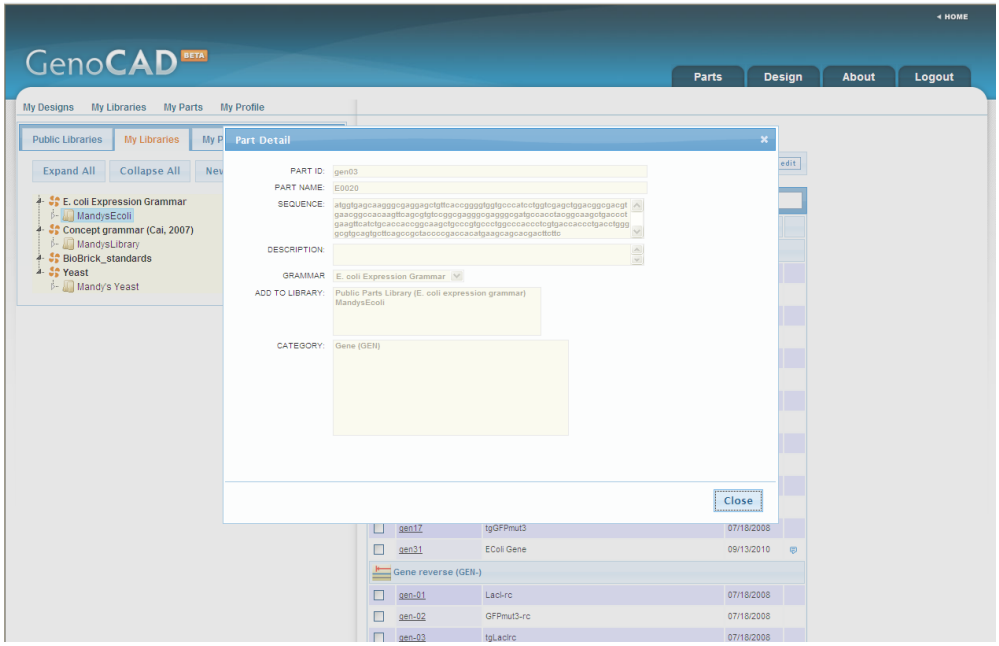


Figure 3.5: View Part. The user may view a part from any of the Parts Registry views. Users may also edit their own parts from either the My Parts or My Libraries views.

3.8 My Parts

The My Parts tab allows users to view the parts they have added to the system. The navigation on this tab is only by design strategy and category, and library is not included in the hierarchy; this is because a part can belong to multiple libraries, but the libraries are all referencing the same part. For example, if a shared part is edited, the changes appear in all the libraries that use that part.
The following options are available from the My Parts tab:

- Edit Existing Parts: In GenoCAD, users may edit their own parts, but not global parts added to their private libraries. The user edits one of his own parts by clicking on the Part ID of the part he wishes to change. This pops up the Edit Part screen where the part's name, sequence, description, libraries, and categories may be changed. As with libraries, parts are design strategy/grammar-specific and cannot be moved to other design strategies/grammars because it could affect existing libraries or designs.
- New Part: The New Part button is on the left navigation bar, and it works the same way as the Add New Part button on the My Libraries tab.
- Add Selected to My Cart: This button adds the selected parts to My Cart in the same way that parts can be added from Public Libraries or Global Libraries.

3.9 Designing Sequences

When the users are finished assembling their personal libraries, then they are ready to create design sequences using a process described elsewhere [Cai, Wilson, and Peccoud 2010; Cai et al. 2007; Czar, Cai, and Peccoud 2009]. Briefly, to begin creating sequences, the user clicks on the Design tab. When the Design page initially loads, the first step is to select a design strategy/grammar and a library; the defaults are "E. coli Expression Grammar" and "Public Parts Library (E. coli Expression Grammar)." If a different design strategy/grammar is selected, the library drop down automatically updates to include only libraries that match the selected design strategy.
Once the design strategy and library have been selected, the user may begin building his design. Starting from the start symbol (usually S), the user iteratively selects a number of rewriting rules of the selected grammar, transforming each category into subcategories and then into parts (Figure 3.6).
When a part from the library has been assigned to each category that compose the design, the sequence is complete, and an alert appears on the upper-right-hand side of the screen ("Your sequence is ready!"). The Download button displays the designed sequence that can be saved to a file. Since this sequence has been built using standard design strategies, and is made up of standardized genetic segments that are easier to check, there should be fewer

errors in this sequence than would be encountered if it had been edited at the sequence level using traditional sequence editing software.
At any point during the design process, the Save Design link at the upper-right hand corner of the Design page may be selected to save the design for future viewing and editing. When saving a design, the user is prompted to enter the design name and a description of that design. If a design is saved before it is finished, when the design is reviewed it displays its state at the Step where the user left off, but it cannot be downloaded as a finished sequence. As with libraries and parts, it is possible to edit saved designs, or even make copies of designs so the copy can be modified without losing the original design. To view saved designs, the user clicks on My Designs from the submenu at the top of the screen.

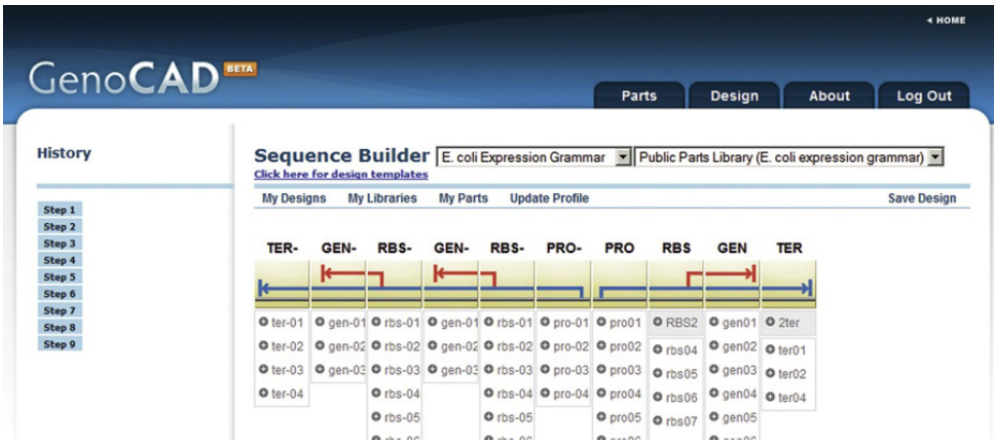


Figure 3.6: Design sequence. Using GenoCAD's design strategies (also called grammars) and personalized libraries, the user can develop quality sequences in a very short period of time using a point-and-click user interface.

3.10 Installing GenoCAD

After experimenting with GenoCAD on GenoCAD.org, advanced users will want to install GenoCAD on their own servers. This solution allows organizations to protect their intellectual property by leaving sensitive information behind their firewall. It also makes it possible to customize the GenoCAD database content to the specific needs of an organization instead of relying on generic grammars and parts libraries.
GenoCAD is developed using the PHP Zend framework. This section describes how GenoCAD 1.5.4 may be installed on a local server. If installing a future version of GenoCAD, it is recommended that the instructions in the INSTALL.txt file be followed. To run GenoCAD, the latest stable version of PHP (at least 5.0 or greater) and the latest stable release of MySQL are recommended.

- Download the source code. To begin with, the source code should be downloaded from SourceForge. Although the source code is version-controlled using SourceForge's instance of subversion, it is possible to download the latest stable release as a package by clicking on the Download Now! button provided by SourceForge. If the latest stable release is downloaded as a package, it needs to be unzipped before proceeding to the next step.

- Copy the genocad directory to the server's web root directory. To simplify the coordination of the Zend parts of the application, it is strongly recommended that the GenoCAD instance have its own domain URL, rather than installing it as a subdirectory under another root, for example, having the index.php file in `https://mygenocadinstance.myinstitution.edu/` is better than having index.php in `https://www.myinstitution.edu/mygenocadinstance`.

- Edit the php.ini file to set short_open_tag to "On."

- Create a Virtual Host for Zend. Zend works best if it behaves as if it were in the root directory of the domain, but because there is some non-Zend-compliant legacy code interfacing with parts of the application written using Zend, a Virtual Host must be created for the genocad/ zend directory. The exact technique may vary depending on the web server being used, but there is a good description of what needs to be done in this Zend article: `http://framework.zend.com/manual/en/learning.quickstart.create-project.html`; the primary focus of this article is how to create a Zend project, but it includes a section on creating a virtual host. As an example on how to create a virtual host on an Apache server, the relevant portions of the public website's httpd.conf are displayed below:

```
<VirtualHost *:80>
<Clipped out sections of this tag that didn't need to change>
<Directory ''/srv/www/vhosts/www.genocad.org/htdocs/no-ssl/zend/public/''>
DirectoryIndex index.php Allow Override All Order allow,deny Allow from all
</Directory>
</VirtualHost>
Alias /zend/ ''/srv/www/vhosts/www.genocad.org/htdocs/no-ssl/zend/public/''
```

- Create a mysql database instance for the local GenoCAD database. Import genocad.sql to create a seed database. As an example, the command for importing this database would probably be something like this:

```
mysql -u <username> -p <name of genocad repository created in previous step>
< genocad.sql
```

- Modify the following files to set database connection settings to point to the database just created:

  Common.php (set variables "Database", "Host", "Port", "User", and "Password" under the $CCConnectionSettings array.)

  common_genocad.php (set variables $server, $user, $pwd, and $db)

  `zend\application\configs\application.ini` (set variables resources.db.params.host, resources.db.params.username,
  resources.db.params.password, and resources.db.params.dbname).

- Edit the `genocad\includes\top_nav.inc` file to change all references to `www.genocad.org` to the local GenoCAD instance's URL. If running genocad as a subdirectory off the web root URL (instead of as a URL that routes directly to the genocad directory), the "" suffix needs to be included on the URL references (i.e., `http://localhost/genocad`).

- Restart the web server.

3.11 Anticipated Evolutions

Since GenoCAD is an active research project, it is already possible to give an insight into some of the upcoming enhancements.
In the current version of GenoCAD, users may view their own designs, parts, and libraries, but may not share them with other users. This level of granularity of the security model is adequate if the user is working alone, but it is limiting in situations where different users need to collaborate on a project. The collaboration features will allow users to grant read or read/write access to their libraries, parts, and designs. The permissions will be granular, so users could continue to keep some of their work private, while collaborating on other projects with others. Managers will also be able to create teams, so different people in an organization could share responsibility over different projects.
Users can create their own parts and libraries, but there is no interface to allow them to create and modify their own design strategies. It is possible to customize existing grammars or create new grammars directly in the back-end database. It is desirable to progressively give users more control over the design strategies they use in their project. It is, however, fairly challenging to formalize the grammar development process and develop user interfaces that can successfully guide users having no previous experience with formal grammars through that process [Oliveira et al. 2009].
GenoCAD currently saves designs as the series of rewriting rules that produces a DNA sequence. This creates a number of potential problems. For instance, the sequence of a part used in a design may be edited in the database. In that case, the change of the part sequence would propagate to the design sequence unbeknownst to the design owner. It is anticipated

that future designs will be saved as DNA sequences. The consistency of the design sequence with the latest version of the design strategy and parts library will be periodically checked using previously described parsing algorithms [Cai et al. 2007].
Another upcoming feature is the possibility to translate DNA sequences into dynamic models of the molecular interactions encoded in the sequence. It was recently proposed to translate synthetic DNA sequences into SBML files using attribute grammars, an extension of the grammars currently used in GenoCAD. GenoCAD's attribute grammars are grammars augmented with a semantic model describing the biological function of parts in the context in which they are used. Attribute grammars include parts attributes and semantic actions associated to rules. Together they make it possible to translate the DNA sequence of construct into equations describing the construct dynamics [Cai et al. 2009]. Implementing such a translator in GenoCAD will make it possible to embed an existing SBML simulator in GenoCAD [Bergmann and Sauro 2008]. Once this feature has been implemented, it will be possible to automate the exploration of the design space generated by a grammar with the goal of finding optimal solutions to a design problem [Ball, Moody, and Peccoud 2010; Cai et al. 2009].

Acknowledgments

The development of GenoCAD is supported by NSF Award EF-0850100. Laura Adam is supported by a fellowship from SAIC.

References

[1] S Altschul. "Basic Local Alignment Search Tool". In: *Journal of Molecular Biology* 215.3 (Oct. 1990), pp. 403–410. ISSN: 00222836. DOI: 10.1006/jmbi.1990.9999.

[2] David a. Ball, Stephen E. Moody, and Jean Peccoud. "CytoIQ: an adaptive cytometer for extracting the noisy dynamics of molecular interactions in live cells". In: *Image Processing* (2010), pp. 75681D–75681D–11. DOI: 10.1117/12.842342.

[3] Frank T Bergmann and Herbert M Sauro. "Comparing simulation results of SBML capable simulators". In: *Bioinformatics* 24.17 (2008), pp. 1963–1965.

[4] Yizhi Cai, Mandy L. Wilson, and Jean Peccoud. "GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs." In: *Nucleic acids research* 38.8 (May 2010), pp. 2637–44. ISSN: 1362-4962. DOI: 10.1093/nar/gkq086.

[5] Yizhi Cai et al. "A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts." In: *Bioinformatics (Oxford, England)* 23.20 (Oct. 2007), pp. 2760–7. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btm446.

[6] Yizhi Cai et al. “Modeling structure-function relationships in synthetic DNA sequences using attribute grammars.” In: *PLoS computational biology* 5.10 (Oct. 2009), e1000529. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1000529.

[7] Deepak Chandran, Frank T Bergmann, and Herbert M Sauro. “TinkerCell: modular CAD tool for synthetic biology.” In: *Journal of biological engineering* 3 (Jan. 2009), p. 19. ISSN: 1754-1611. DOI: 10.1186/1754-1611-3-19.

[8] Kevin Clancy and Christopher a Voigt. “Programming cells: towards an automated ’Genetic Compiler’.” In: *Current opinion in biotechnology* 21.4 (Aug. 2010), pp. 572–81. ISSN: 1879-0429. DOI: 10.1016/j.copbio.2010.07.005.

[9] Michael J Czar, Yizhi Cai, and Jean Peccoud. “Writing DNA with GenoCAD.” In: *Nucleic acids research* 37.Web Server issue (July 2009), W40–7. ISSN: 1362-4962. DOI: 10.1093/nar/gkp361.

[10] JA Jonathan A Goler, Brian W BW Bramlett, and Jean Peccoud. “Genetic design: rising above the sequence”. In: *Trends in biotechnology* 26.10 (Oct. 2008), pp. 538–544. ISSN: 0167-7799. DOI: 10.1016/j.tibtech.2008.06.003.

[11] Anthony D Hill et al. “SynBioSS: the synthetic biology modeling suite.” In: *Bioinformatics (Oxford, England)* 24.21 (2008), pp. 2551–3. ISSN: 1460-2059. DOI: 10.1093/bioinformatics/btn468.

[12] Roberta Kwok. “Five hard truths for synthetic biology.” In: *Nature* 463.7279 (Jan. 2010), pp. 288–90. ISSN: 1476-4687. DOI: 10.1038/463288a.

[13] Mario a M.A. Marchisio and Jörg Stelling. “Computational design tools for synthetic biology.” In: *Current opinion in biotechnology* 20.4 (2009), pp. 479–85. ISSN: 1879-0429. DOI: 10.1016/j.copbio.2009.08.007.

[14] Nuno Oliveira et al. “VisualLISA: Visual programming environment for attribute grammars specification”. In: *2009 International Multiconference on Computer Science and Information Technology* (Oct. 2009), pp. 691–698. DOI: 10.1109/IMCSIT.2009.5352765.

[15] Jean Peccoud et al. “Targeted development of registries of biological parts”. In: *PLoS One* 3.7 (Jan. 2008), e2671. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0002671.

[16] Michael Pedersen and Andrew Phillips. “Towards programming languages for genetic engineering of living cells.” In: *Journal of the Royal Society, Interface / the Royal Society* 6 Suppl 4.April (Aug. 2009), S437–50. ISSN: 1742-5662. DOI: 10.1098/rsif.2008.0516.focus.

[17] P.E.M. Priscilla E M Purnick and Ron Weiss. “The second wave of synthetic biology: from modules to systems”. In: *Nature Reviews Molecular Cell Biology* 10.6 (June 2009), pp. 410–422. ISSN: 1471-0080. DOI: 10.1038/nrm2698.

[18] Alan Villalobos et al. “Gene Designer: a synthetic biology tool for constructing artificial DNA segments.” In: *BMC bioinformatics* 7 (2006), p. 285. ISSN: 1471-2105. DOI: 10.1186/1471-2105-7-285.

[19] Emma Weeding, Jason Houle, and Yiannis N Kaznessis. "SynBioSS designer: a web-based tool for the automated generation of kinetic models for synthetic biological constructs". In: *Briefings in bioinformatics* 11.4 (2010), pp. 394–402.

Chapter 4

Development of a domain-specific genetic language to design *Chlamydomonas reinhardtii* expression vectors

Abstract

Motivation: Expression vectors used in different biotechnology applications are designed with domain-specific rules. For instance, promoters, origins of replication, or homologous recombination sites are host-specific. Similarly, chromosomal integration or viral delivery of an expression cassette impose specific structural constraints. As de novo gene synthesis and synthetic biology methods permeate many biotechnology specialties, the design of application-specific expression vectors becomes the new norm. In this context, it is desirable to formalize vector design strategies applicable in different domains.

Results: Using the design of constructs to express genes in the chloroplast of *Chlamydomonas reinhardtii* as an example, we show that a vector design strategy can be formalized as a domain-specific language. We have developed a graphical editor of context-free grammars usable by biologists without prior exposure to language theory. This environment makes it possible for biologists to iteratively improve their design strategies throughout the course of a project. It is also possible to monitor that vectors designed with early iterations of the language are consistent with the latest iteration of the language.

Availability: The context-free grammar editor is part of the GenoCAD application. A public instance of GenoCAD is available at `http://www.genocad.org`. GenoCAD source

code is available from SourceForge and licensed under the Apache v2.0 open source license.

Supplementary Information: Several versions of the Chloroplast grammar are included in the online supplement.

4.1 Introduction

Most bioinformatics software packages include sequence editors that facilitate the design and assembly of new DNA sequences. Automatic recognition of sequence features, identification of restriction sites, and tools to add sequence annotations help biologists visualize the different elements of the DNA sequences they manipulate. Software let users switch between graphical representations of DNA sequences that provide a macroscopic view and textual representations more suitable to examine sequences with a base-level resolution. Irrespective of the software environment used, the cut-and-paste approach to sequence editing increase the chance of introducing errors such as leaving or deleting a DNA segment, accidently inserting it twice, or truncating a functional element. For a large, multi-gene construct (e.g. an expression cassette encoding all components of a biochemical pathway), such risk could become unacceptably high. In many cases errors are uncovered after several months of unsuccessful attempts to express a gene. Many of these errors can be avoided by developing a library of genetic parts prior to designing DNA sequences. The Registry of Standard Biological Parts [Peccoud et al. 2008] was the first database of genetic parts. The notion of genetic parts supports a different approach to sequence design. Complex genetic constructs can be designed using drag-and-drop user interfaces that rely on icons to represent different categories of genetic parts [Villalobos et al. 2006]. This added level of abstraction makes it easier to understand the structure of a new sequence. It also avoids sequence manipulation errors. However, it still makes it possible to design sequences lacking components required for proper gene expression.

We demonstrated that the structure of many gene expression vectors can be modeled as context-free grammars [Cai et al. 2007]. GenoCAD, a web-based application to design synthetic DNA sequences, relies on the notion of *grammars* to organize large collections of genetic parts [Cai, Wilson, and Peccoud 2010]. It also includes a wizard-like sequence editor that guides users through a series of design decisions corresponding to the rewriting rules of a grammar select-ed by the user [Czar, Cai, and Peccoud 2009].

Initially, users could only choose from a set of public grammars when designing sequences. These public grammars were developed by manually adding records in the GenoCAD backend database. The process was tedious and only people familiar with the GenoCAD data model could develop new grammars. These early grammars were interesting as proof of concepts but they did not necessarily reflect the design rules that users wished to use for specific research projects.

We have now formalized the grammar development process and developed a graphical user

interface enabling life-scientists to develop context-free grammars. The GenoCAD Grammar Editor allows users to revise existing grammars, or even to develop brand-new grammars. These grammars can be fairly generic to generate a broad range of expression vectors for a new host, for example. Alternatively, they can be made very specific to capture project-specific design constraints such as the ones resulting from intellectual property licensing agreements. These languages describing families of synthetic DNA molecules are comparable to Domain Specific Languages (DSL) used in computer programming. Specialized DSL facilitate communication between programmers and domain experts by directly expressing concepts specific of the domain. HTML and Cascading Style Sheets are examples of DSL that allow graphic designers who do not see themselves as computer programmers to program the rendering of webpages in web browsers.
The design of expression vectors to express genes in the chloroplast of *Chlamydomonas reinhardtii* is an example of a domain that calls for the development of a DSL. Site-specific gene insertion into Chlamydomonas chloroplast can be performed through homologous recombinations, making it an attractive system to study processes such as photosynthesis. In addition, Chlamydomonas chloroplast is capable of carrying out anaerobic reactions such as hydrogen production [Esquível et al. 2011; Hemschemeier, Melis, and Happe 2009], making it an attractive venue to introduce anaerobic pathways such as nitrogen fixation. Chalmydomonas chloroplast genome has a prokaryotic arrangement, encoding both mono- and polycistronic genes, offering potential for introduction of polycistronic synthetic expression cassettes like in bacteria [Temme, Zhao, and Voigt 2012]. Yet, the DSLs that are available for bacterial synthetic constructs would not be sufficient, because in many cases the exact locations of regulatory components are not known. For example, ribosomal binding sequences (RBSs) are not characterized for most of the genes, and it is predicted that the translational initiation is more complex than the ancestry bacterial system [Manuell, Quispe, and Mayfield 2007]. This necessitates more flexibility than a simple promoter-RBS-CDS arrangement found in synthetic constructs for bacteria; in some cases it is not practical to separate the promoter from 5'-UTR sequence that potentially includes RBS and other regulatory elements. Almost undoubtedly, the expression of multiple genes from a complex, polycistronic unit would require empirical optimization of genetic parts through biological assays, hence a rapid method to assemble multiple parts without a fear of introducing sequence mistakes is highly beneficial.

4.2 System and Methods

4.2.1 Accessing the grammar editor

The grammar editor can be accessed from GenoCAD's Parts module by clicking on the Parts header, then the Grammars tab from the left-hand navigation bar. This displays a list of the grammars available in the system, sorted under Public Grammars (the ones anyone can use and view) and User Grammars (those that belong to the logged-in user.) From this

list, users can select a grammar from the list on the left, and see a summary of the selected gram-mar on the right, including a description of the grammar, the number of categories and rules that define the grammar, and the number of libraries and parts associated with that grammar.

4.2.2 Category definition

The grammar development process starts by declaring categories of genetic parts, such as promoters, transcription terminators, or coding sequences. Each category needs an alphanumeric code. By convention, three or four capital letters such as PRO for promoter or TER for terminator work well. In addition, the category needs to be defined by providing a short text description of the type of genetic element corresponding to the category. An optional field is available to associate the category with a specific GenBank qualifier. This is used to ensure that parts in that category will be properly annotated when exporting expression vectors as GenBank files. Finally, the software allows users to select an icon that will be used to represent parts of this category in the GenoCAD design tool.
All GenoCAD grammars come with a set of predefined reserved categories. The first is the Start Category (S) which is the default root to the hierarchical rules tree; it is required in the grammar definition that one category be designated as the root, but it is possible to designate a separate category to fulfill this role. The other reserved categories are used as sequence delimiters. Brackets are used to indicate the orientation of a DNA sequences, parentheses are used to delimit plasmids in constructs that involve more than one plasmid, and braces are used to delimit chromosomes in genome design projects.

4.2.3 Rule Declaration

As soon as a few categories have been declared, it is possible to start declaring rewriting rules. The rules can be added by means of a drag-and-drop interface that allows the user to pull categories from a list on the left and organize them on the right. Rules are identified using a short alphanumeric code. Lower case codes are used to differentiate them from category codes.
In order to facilitate the development of sets of transformation rules, it is convenient to identify three groups of categories. *Rewritable* categories are categories that are composed of one or more different categories. For example rule sgen: $CDS \rightarrow ATGGENSTP$ states that the category CDS is composed of a start codon (ATG), an open reading frame (GEN), and a stop codon (STP). *Terminal* categories are those that may not be transformed via a rule to another category or set of categories; usually DNA segments (parts) will be associated with terminal categories. ATG and STP are examples of terminal categories. *Orphaned* categories are those not used in any production rule.
Initially, all categories are in the orphaned group. As new rules are added to the grammar,

the software automatically reassigns categories to one of the three groups according to their use. A properly developed grammar should not have any orphaned category. Predefined categories that are not applicable to a particular grammar should be deleted. For instance, the chromosome delimiters are rarely used.

4.2.4 Grammar management

The development of a new grammar proceeds through multiple iterations that include both category and rule definitions. The development of new rules can necessitate the declaration of new categories. A test tool makes it possible to test the new grammar by experimenting with the existing rules. These simulations often uncover limitations of the existing rule sets. Users may only make changes to their own grammars. Users can also make a copy of an existing public grammar and use this as a template for the development of a new grammar. In addition, grammar files may be imported into the user's workspace; such files are available from a number of sources. We have published several GenoCAD grammars on `Figshare.com` or as exported by a colleague.
When users copy or export a grammar, they have the option of also including any parts libraries, and, by extension, designs and parts, with the grammar; this is a quick way to export a complete data set and share it with a colleague who can load it into their own GenoCAD workspace for review or further modification.

4.2.5 Parser generation and plasmid verification

When a plasmid is created using GenoCAD, the design tool guides the user through grammar- and library-appropriate choices, thereby preventing the user from developing an invalid design. However, subsequent changes to the design's underlying grammars, libraries, or parts may alter the design's validation status. GenoCAD has implemented a three-step validation feature to allow for revalidation of developed designs (Figure 4.1).

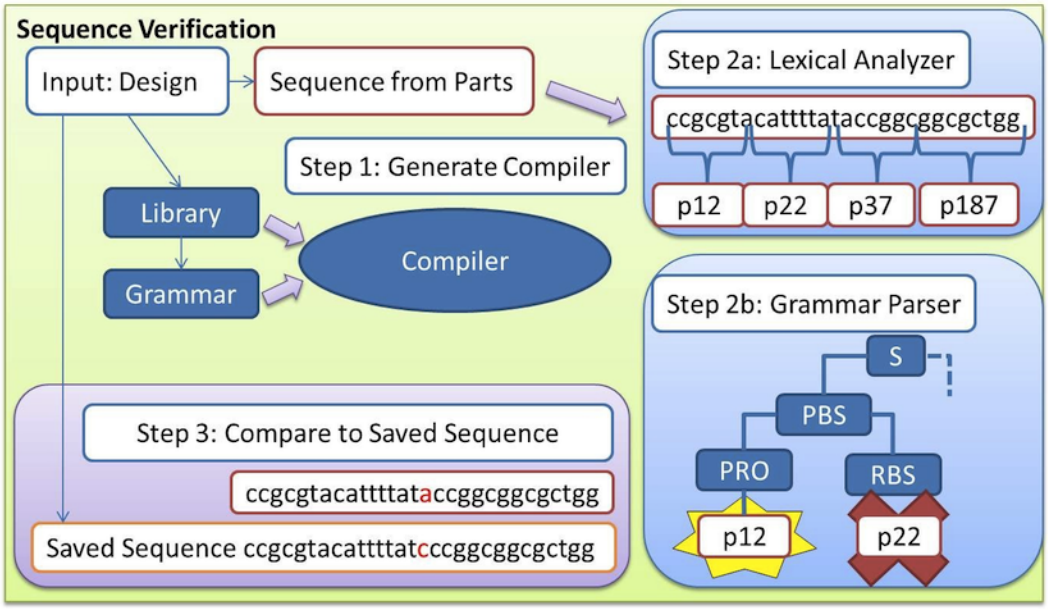


Figure 4.1: Sequence Validation. When a design is marked for revalidation, the following data elements are identified: the sequence, as built from the underlying parts of the design; the library of parts associated with the design; the grammar associated with the library; and the sequence as it was when the design was last saved. In the first step, the library of parts and the grammar are used to write out a Prolog compiler. In Step 2, the sequence is submitted to the compiler for LR lexical analysis to try to identify the parts that make up this design; if this succeeds, then the parts list are run through a top-down grammatical parser to ensure that the parts are ordered in a way that is supported by the grammar. If Step 2 indicates that the design is valid, in the last step the sequence that was just compiled is compared to the sequence from when the sequence was last saved. This helps identify changes to underlying parts in a design; while not, strictly speaking, a question of validation at this point, it at least flags the user that the design does not resolve to the same sequence that it did before.

The first step is to generate a compiler specific to the grammar and library that the design is associated with (Figure 4.2). Written in Prolog, this compiler consists of some hard-coded functions to handle processing of the design, but also contains the grammar's rule specifications, a list of the parts supported by the library, and the relation-ship between the categories and the parts. The compiler's algorithm incorporates strategies from Moore's algorithm [Moore 2000] to remove immediate left recursion from context-free grammars, because Prolog does not natively support it; this feature makes it possible for users to define rules like "$CAS \rightarrow CAS, CAS$", where the first category on the right side of the rule is the same as the category on the left side. It should also be noted that this compiler needs to be regenerated every time the design is validated to ensure that all modifications to the

grammar, library, or parts have been captured.

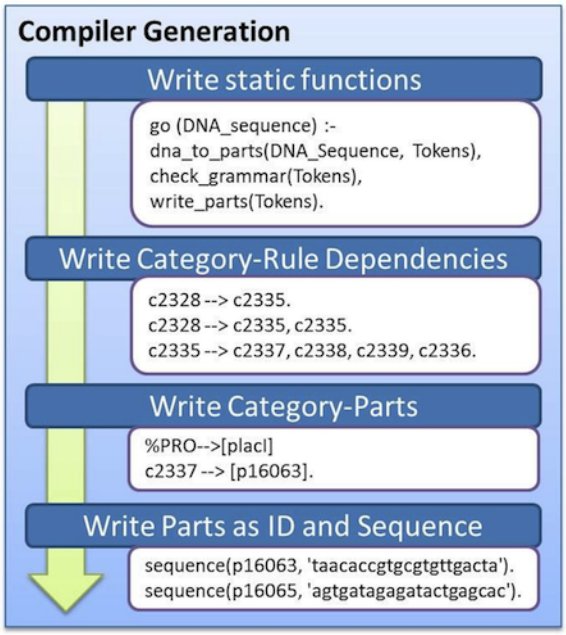


Figure 4.2: Compiler Generation Workflow. To validate designs, GenoCAD generates an on-demand compiler tied to the specific library and grammar associated with the design. This compiler has to be rebuilt every time a design is validated because changes to underlying grammars, libraries, or even parts can affect the validation status of that design. When writing the compiler, the first step is to write out the static functions that are used by any library and grammar – the dna_to_parts function, for example, or check_grammar. The second step is to write the rules based on category identifiers using Definite Clause Syntax (DCS) format; that is, if I have a grammar rule that says that category RBGN can be resolved to categories RBS and Gene, then the rule could be written as "$RBGN \rightarrow RBS, GEN$"; within the code, though, we use unique primary keys (ie, $c < category\_id >$) from the database to avoid replication that would cause issues with Prolog. In the third step, we map the categories from the rules to their associated parts, again using their unique identifiers. Finally, in the last step, we map the unique part identifiers to their associated DNA sequences.

In the second step, the sequence of the design is generated from its associated parts, and this sequence is passed to the compiler for a two-phased validation strategy. In the first phase, a lexical analyzer uses a LR parsing algorithm to divide the sequence into parts; this may seem redundant, since the sequence was just assembled from the design's underlying parts, but this verifies that the design consists only of parts still present in its parts library. If the first phase returns a list of parts instead of an invalid status, the second phase of the compilation process parses the grammar hierarchy, following a top-down parsing strategy from the start category down to the parts level to see if this series of parts can be reconciled

to a grammatically valid construct. This step confirms that the design still conforms to the grammar.
If the sequence is deemed valid during the second step, the third step is to compare the sequence generated from the parts of the design to the sequence of the design from the last time it was saved; this is a simple string comparison and does not involve the compiler. We perform this check because a design may still be consistent with its parts library and grammar, but changes to the sequences of underlying parts could result in a different design sequence than the one that was originally developed. In this case, the design is marked with a caution sign, which warns that while the design is still valid based on its parts and grammars, the underlying sequence has changed. The validation statuses are described in Figure 4.3.

Status	Definition
	Out-of-date: One of the sequences of one of the underlying parts has changed, so while the design is still valid according to the library and grammar, it doesn't resolve to the same sequence as before.
	Unfinished: This design was saved in an unfinished state (ie, parts are not selected for all of the categories.) An unfinished design cannot be validated using the compiler.
	Invalid: Either one of the parts used in this design is no longer in the design's part library, one of the parts has changed functional categories (ie, was a promoter, now is a terminator), or the underlying grammar rules have changed.
	Needs Validation: One of the underlying parts has changed, and it may or may not have invalidated the design's status. Revalidation can be forced by clicking on the status icon.
	Valid: Design is consistent with its library and grammar and with the previously saved version.

Figure 4.3: Validation Statuses. This table illustrates what the various validation statuses mean and why they occur.

4.3 Results

We have applied the workflow described in the previous section to develop a grammar to design vectors expressing one or multiple genes from a DNA sequence inserted in the chloroplast chromosome.

4.3.1 Category definition

The list of categories includes those found in many grammars, like open reading frames, tagging sequences, or transcription terminators. Here, we have decided to declare start and stop codons as a category. This common convention facilitates the design of fusion proteins by concatenation of multiple protein coding sequences lacking stop codons.
The grammar also includes categories that are very specific to this application. For instance, the translation initiation sites are not as well-characterized in *C.reinhardtii* as in *E.coli*. The promoter category is defined as sequences that include both transcription and translation initiation sites. The *C.reinhardtii chloroplastic* chromosome has several operons in which adjacent coding sequences are separated by Short Interval Sequences. We decided to define these sequences as a part category that will be used to make polycistronic cassettes.
In order to target a specific region of the chloroplast chromosome by homologous recombination, the constructs need to include flanking sequences in 5' and 3' position of the expression cassette.
The grammar also includes a number of rewritable categories corresponding to large structural blocks. Adhering to a convention in language theory, the S category is used as starting point of a design process. Another category labeled "targeted expression cassette" corresponds to expression cassettes flanked by two genomic sequences for homologous recombination.
Upon completion of the category definition process (see Figure 4.4), we imported a few representative sequences into each category.

Category ID	Icon	Type	Description
S (Start)		Rewritable	Start category; the default "root" category of the grammar.
CAS		Rewritable	Expression cassette delimited by a promoter in 5' and a transcription terminator in 3'.
CDS		Rewritable	Open reading frame composed of several protein domains. Does not include start and stop codons.
GEN		Rewritable	Gene or protein domain. By convention does not include start and stop codons.
TER		Rewritable	Terminator; can be used either singly or in pairs.
TCS		Rewritable	Targeted expression cassette. Expression cassette flanked with two adjacent genomic sequences for homologous recombination.
[,]		Terminal	Negative orientation delimiters
(,)		Terminal	Plasmid delimiters
PBS		Terminal	Sequence associated with the initiation of transcription and translation.
ATG		Terminal	Start codon
VEC		Terminal	Vector
SIS		Terminal	Short Interval Sequences used to make polycistronic cassettes
STP		Terminal	Stop codon
5FLR, 3FLR		Terminal	5' / 3' Flanking region for homologous recombination

Figure 4.4: Categories of genetic parts used in the *C. Reinhardtii* chloroplast grammar.

4.3.2 Rules declaration

The best way to review the rule set is to start from S. S can be rewritten into a single targeted expression cassette (rule 1tcs), a complete plasmid that includes both a cassette and a vector backbone (rule 1plas), or a construct that includes two plasmids (rule 2plas) which could be convenient in cases where the size of the insert exceeds the cloning capacity of the vector backbone.
Three rules are then used to rewrite the CAS category. Rule 2cas is used to introduce an additional cassette in the design. Rule rcas is used to change the orientation of the cassette. Finally rule prct is used to break the cassette down into promoter, open reading frame, and transcription terminator.
The rewriting of CDS is the focus on the next two rules. Rule 2cds is used to create polycistronic constructs. Rule sgen reveals the start and stop codons that delimit the open reading frame (GEN).
Finally, the grammar has three rules to rewrite the GEN category. Rule 2gen creates the possibility to make fusion proteins by combining two open reading frames. Rules tgen and gent correspond to the addition of an epitope tag on the N or C terminus of the open reading frame. The rules are listed in Figure 4.5.

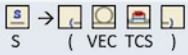
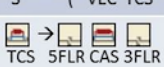
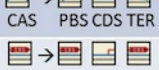
Rule Code	Rule	Description
1tcs		This rule is used to design only one expression cassette
1plas		This rule is used to specify the expression cassette along with the vector where it is inserted. The output is the entire plasmid sequence.
2plas		This rule is for designs that involve two plasmids.
tgs		Specifies the flanking regions for homologous recombination.
2cas		This rule makes it possible to have more than one expression cassette on a construct.
rcas		This rule is used to specify that the cassette is coded on the negative strand.
prct		A gene expression cassette is composed of a promoter, open reading frame, and a transcription terminator.
2cds		This rule makes it possible to design polycistronic constructs.
sgen		The open reading frame is composed of a single gene flanked by a start and stop codon.
2gen		This rule can be used to fuse two coding sequences that are not tags.
tgen		This rule is used to add a tag before the gene. It can be used iteratively to add more than one tag.
gent		This rule is used to add a tag after the gene. It can be used iteratively to add more than one tag.
2ter		This rule makes it possible to include a double terminator.

Figure 4.5: The rules for the *C. Reinhardtii* chloroplast grammar.

4.4 Discussion

4.4.1 Grammar development workflow

The Chloroplast Grammar was quite a bit more complex than the example grammar outlined before. The first step involved white-boarding – identifying what categories were needed to adequately describe the Chloroplast designs we were shooting for, in what order they would appear, and which categories were optional and/or could repeat. The grammar was exported and emailed to our collaborator, a plant physiologist, who suggested revisions and added parts libraries for creating and testing designs. In this way, exchanging questions and comments over email along with grammar export files, the grammar underwent several iterations of revision; however, aside from the first meeting with our collaborator, there were no other face-to-face meetings during the development of the Chloroplast grammar.

4.4.2 Grammar development guidelines

A grammar is a model of a design strategy applicable to a certain application domain. There are often multiple ways to express a set of design principles. Some grammars are better than others.
Rewriting rules that resolve to a long list of categories are usually an indication of bad grammar design. It is often possible to break down this transformation into a set of simpler rewriting rules by introducing intermediate categories corresponding to shorter aggregations of categories. Defining these new categories is often an opportunity to develop an abstraction hierarchy.
When defining categories and rules, it is important to identify re-usable patterns. This limits the number of categories and rewriting rules.
A good grammar should be expressive enough to allow the generation of a broad range of constructs but it should be constrained enough to limit the possibilities of designing faulty constructs. How constraining should the grammar be is something that should be discussed with the domain expert. For example, the grammar presented here allows users to design constructs distributed over two different plasmids. When two plasmids are used together, it is important to ensure that they rely on different selection markers. It was decided that it was not necessary to express this constraint in rule 2plas, but the grammar could easily be modified should it become necessary to enforce this constraint.

4.4.3 Customization versus Standardization

The need to provide biologists with tools to express custom design strategies is somewhat antithetic of the need for standardization. For instance, we could not identify any GenBank qualifier corresponding to the domain specific categories defined in the grammar such as SIS, start codon (ATG), stop codon (STP), or VEC. This limitation would lead to an incomplete rendering of designs built with this grammar if they are exported in GenBank format for import into another application.
Similarly, it proved challenging to use SBOLv, the set of standard icons developed by the Synthetic Biology Open Language project [Galdzicki et al. 2012] (see Chapter 2). This difficulty forced us to develop a custom set of icons.

4.4.4 Limitation of existing parsers

Ideally, it would be possible to evaluate if legacy vector sequences are consistent with a domain-specific language. This comparison could be useful to validate a grammar during the development phase by ensuring that it properly generates vectors known to work well. Vectors that cannot be parsed with the DSL would provide opportunities to augment the grammar by adding more design rules that may have been omitted in the first iterations of

the grammar development process.
Unfortunately most legacy sequences cannot be analyzed using the parsers derived from the DSL. One issue is that most expression vectors are circular and parsers require linear input sequences. The parsing output depends on the origin used to linearize the plasmid sequence. The analysis of legacy sequence would require parsers capable of processing circular sequences.
Legacy vectors include a few islands of functional elements separated by seas of DNA sequences that reflect the cloning process used to derive these vectors from existing plasmids. Analyzing these legacy vectors would require island parsers that could recognize the island of functional blocks and ignore the sequences that separate them.

Authors

Mandy L Wilson, Sakiko Okumoto, Laura Adam and Jean Peccoud

Affiliations:

MLW, LA, JP: Virginia Bioinformatics Institute, Virginia Tech, Blacksburg VA 24061, USA
SO: Department of Plant Pathology, Physiology, and Weed Science, Virginia Tech, Blacksburg VA 24061, USA
JP: ICTAS Center for Systems Biology of Engineered Tissues, MC 0193 Virginia Tech, Blacksburg, VA 24061, USA

Funding:

This work was supported by the National Science Foundation [Grant EF-0850100 to JP].

References

[1] Yizhi Cai, Mandy L. Wilson, and Jean Peccoud. "GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs." In: *Nucleic acids research* 38.8 (May 2010), pp. 2637–44. ISSN: 1362-4962. DOI: 10.1093/nar/gkq086.

[2] Yizhi Cai et al. "A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts." In: *Bioinformatics (Oxford, England)* 23.20 (Oct. 2007), pp. 2760–7. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btm446.

[3] Michael J Czar, Yizhi Cai, and Jean Peccoud. "Writing DNA with GenoCAD." In: *Nucleic acids research* 37.Web Server issue (July 2009), W40–7. ISSN: 1362-4962. DOI: 10.1093/nar/gkp361.

[4] Maria G Esquível et al. "Efficient H2 production via Chlamydomonas reinhardtii." In: *Trends in biotechnology* 29.12 (Dec. 2011), pp. 595–600. ISSN: 1879-3096. DOI: 10.1016/j.tibtech.2011.06.008.

[5] Michal Galdzicki et al. "Synthetic Biology Open Language (SBOL) Version 1.1.0". In: (2012), pp. 1–26.

[6] Anja Hemschemeier, Anastasios Melis, and Thomas Happe. "Analytical approaches to photobiological hydrogen production in unicellular green algae." In: *Photosynthesis research* 102.2-3 (2009), pp. 523–40. ISSN: 1573-5079. DOI: 10.1007/s11120-009-9415-5.

[7] Andrea L Manuell, Joel Quispe, and Stephen P Mayfield. "Structure of the chloroplast ribosome: novel domains for translation regulation." In: *PLoS biology* 5.8 (Aug. 2007), e209. ISSN: 1545-7885. DOI: 10.1371/journal.pbio.0050209.

[8] RC Moore. "Removing left recursion from context-free grammars". In: *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. 2000.

[9] Jean Peccoud et al. "Targeted development of registries of biological parts". In: *PLoS One* 3.7 (Jan. 2008), e2671. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0002671.

[10] Karsten Temme, Dehua Zhao, and Christopher a Voigt. "Refactoring the nitrogen fixation gene cluster from Klebsiella oxytoca." In: *Proceedings of the National Academy of Sciences of the United States of America* 109.18 (May 2012), pp. 7085–90. ISSN: 1091-6490. DOI: 10.1073/pnas.1120788109.

[11] Alan Villalobos et al. "Gene Designer: a synthetic biology tool for constructing artificial DNA segments." In: *BMC bioinformatics* 7 (2006), p. 285. ISSN: 1471-2105. DOI: 10.1186/1471-2105-7-285.

Chapter 5

Modeling Structure-Function Relationships in Synthetic DNA Sequences using Attribute Grammars

Published in:

Cai, Y., Lux, M. W., Adam, L., & Peccoud, J. (2009). Modeling structure-function relationships in synthetic DNA sequences using attribute grammars. PLoS computational biology, 5(10), e1000529. doi:10.1371/journal.pcbi.1000529

Abstract

Recognizing that certain biological functions can be associated with specific DNA sequences has led various fields of biology to adopt the notion of the genetic part. This concept provides a finer level of granularity than the traditional notion of the gene. However, a method of formally relating how a set of parts relates to a function has not yet emerged. Synthetic biology both demands such a formalism and provides an ideal setting for testing hypotheses about relationships between DNA sequences and phenotypes beyond the gene-centric methods used in genetics. Attribute grammars are used in computer science to translate the text of a program source code into the computational operations it represents. By associating attributes with parts, modifying the value of these attributes using rules that describe the structure of DNA sequences, and using a multi-pass compilation process, it is possible to translate DNA sequences into molecular interaction network models. These capabilities are illustrated by simple example grammars expressing how gene expression rates are dependent upon single or multiple parts. The translation process is validated by systematically generating, translating, and simulating the phenotype of all the sequences in

the design space generated by a small library of genetic parts. Attribute grammars represent a flexible framework connecting parts with models of biological function. They will be instrumental for building mathematical models of libraries of genetic constructs synthesized to characterize the function of genetic parts. This formalism is also expected to provide a solid foundation for the development of computer assisted design applications for synthetic biology.

Author Summary

Deciphering the genetic code has been one of the major milestones in our understanding of how genetic information is stored in DNA sequences. However, only part of the genetic information is captured by the simple rules describing the correspondence between gene and proteins. The molecular mechanisms of gene expression are now understood well enough to recognize that DNA sequences are rich in functional blocks that do not code for proteins. It has proved difficult to express the function of these genetic parts in a computer readable format that could be used to predict the emerging behavior of DNA sequences combining multiple interacting parts. We are showing that methods used by computer scientists to develop programming languages can be applied to DNA sequences. They provide a framework to: 1) express the biological functions of genetic parts, 2) how these functions depend on the context in which the parts are placed, and 3) translate DNA sequences composed of multiple parts into a model predicting how the DNA sequence will behave in vivo. Our approach provides a formal representation of how the biological function of genetic parts can be used to assist in the engineering of synthetic DNA sequences by automatically generating models of the design for analysis.

5.1 Introduction

"How much can a bear bear?" This riddle uses two homonyms of the word "bear". The first instance of the word is a noun referring to an animal, and the second is a verb meaning "endure". Although the word "bear" has over 50 different meanings in English, its meaning in any given sentence is rarely ambiguous. In a simple case like this riddle, the meaning of each word can be deciphered by looking at other words in the same sentence. In other cases, it is necessary to take into account a broader context to properly interpret the word. For instance, it may be necessary to read several sentences to decide if "bear claw" refers to a body part or a pastry. A reader will progressively derive the meaning of a text by recognizing structures consistent with the language grammar. It is often difficult to understand the meaning of a text by relying exclusively on a dictionary.

It is interesting to compare this bottom-up emergence of meaning with the top-down approach that made genetics so successful. The discipline was built upon a quest to define

hereditary units that could be associated with observable traits well before the physical support of heredity was discovered [Keller 2001; Sturtevant 2001]. The one-to-one relationship between genes and traits was later refined by Beadle and Tatum's hypothesis that the gene action was mediated by enzymes [Singer and Berg 2004; Tatum 1959]. Cracking the genetic code has been one of the major milestones in understanding the information content of nucleic acids sequences. By demonstrating the colinearity of DNA, RNA, and protein sequences, the genetic code was instrumental in the identification of specific DNA sequences as genes. The influence of this legacy on contemporary biology cannot be underestimated. Models used in quantitative genetics predict phenotypes from unstructured lists of alleles at different loci [Falconer, Mackay, and Frankham 1996; Lynch and Walsh 1998]. Similarly, genome annotations remain very gene-centric. Most bioinformatics databases have been designed to collect information relative to coding regions or candidate genes. Few, if any, annotations of non-coding regions or higher order structures are being systematically recorded even for model organisms like yeast [Güldener et al. 2005; Weng et al. 2003].

Yet, despite its success, the notion of gene appears insufficient to express the complexity of the relation between an organism genome and its phenotype [Keller 2001; Keller and Harel 2007] The elucidation of the molecular mechanisms controlling gene expression has revealed a web of molecular interactions that have been modeled mathematically to show that important phenotypic traits are the emerging properties of a complex system [Chen et al. 2004; Ramsey et al. 2006; Stricker et al. 2008; Tigges et al. 2009; Von Dassow et al. 2000; Wang, Zhang, and Ouyang 2006]. The development of this more integrated understanding of the cell physiology leads to a progressive adoption of the more neutral notion of genetic part as a replacement for the notion of genes associated with specific traits. Making sense of the list of parts generated in genomics, proteomics, and metabolomics has been a major challenge for the systems biology community [Bains 2001; Brasch, Hartley, and Vidal 2004; Eggert, Mitchison, and Field 2006; Fitzkee et al. 2005; Mueller, Martens, and Apweiler 2007; Stewart Jr 2005].

It is becoming apparent that the genetic code captures only a small fraction of the information content of DNA molecules [Rabani, Kertesz, and Segal 2008; Segal et al. 2006]. Yet, if there is a general agreement that the cell dynamics is somehow coded in genetic sequences, no formal relationship between DNA sequences and dynamical models of gene expression has been proposed so far. In particular, the formalization of the biological functions of genetic parts has remained elusive. As a result, building models of gene networks encoded in DNA sequences remains a labor-intensive process. This limitation has hampered the development of large families of models needed to analyze phenotypic data generated by libraries of related genetic constructs [Cox, Surette, and Elowitz 2007; Gardner, Cantor, and Collins 2000; Gertz, Siggia, and Cohen 2008; Guet et al. 2002; Murphy, Balázsi, and Collins 2007].

Synthetic biology is likely to be instrumental in refining our understanding of the design of natural biological systems [Drubin, Way, and Silver 2007]. Just like the genetic code was partly elucidated through the de novo chemical synthesis of DNA molecules [Agarwal et al. 1970; Kay 2000], the redesign of genomic sequences will shed a new light on the relations

between structure and function in genetic sequences [Chan, Kosuri, and Endy 2005; Dymond et al. 2009; Gibson et al. 2008]. By considering biological parts as the building blocks of artificial DNA sequences [Endy 2005], designing new parts that do not exist in nature [Cox, Surette, and Elowitz 2007; Gertz, Siggia, and Cohen 2008; Murphy, Balázsi, and Collins 2007], and making parts physically available to the community [Peccoud et al. 2008], synthetic biology calls for a systematic functional characterization of genetic parts [Canton, Labno, and Endy 2008]. These efforts are still limited by the difficulty in expressing how the function of biological parts may be influenced by the structure of the DNA sequence in which they are used. It has been shown that a partial redesign of the genomic sequences of two viruses had a significant effect on the virus fitness even though the redesigns preserved the protein sequences [Chan, Kosuri, and Endy 2005; Coleman et al. 2008]. Just as the context of the expression "bear claw" helps understand its meaning, it is necessary to consider the entire structure of the DNA molecule coding for particular genes to appreciate how those genes contribute to the phenotype.

One possible approach to this problem is to extend the linguistic metaphor used to formulate the central dogma. The notions of genetic code, transcription, and translation are derived from a linguistic representation of biological sequences. Several authors have modeled the structure of various types of biological sequences using syntactic models [Chiang, Joshi, and Searls 2006; Dong and Searls 1994; Gimona 2006; Knudsen and Hein 2003; Rivas and Eddy 2000; Searls 1992; Searls 1997; Searls 2002]. However, these structural models have not yet been complemented by formal semantic models expressing the sequence function. An interesting attempt to use grammars to model the dynamics of gene expression did not rely on a description of the DNA sequence structure. Instead, this grammar described how various inducible or repressible promoters can transition between different states under the control of environmental parameters [Bentolila 1996]. The simple semantic model stored in a knowledge base established a correspondence between the strings generated by the syntax and the physiological state of the cell. The Sequence Ontology [Eilbeck et al. 2005] and the Gene Regulation Ontology [Beisswanger et al. 2008] represent other attempts to associate semantic values with biological sequences. Their controlled vocabularies can be used by software applications to manage knowledge. However, the semantics derived from these ontologies is a semantics of the sequence annotation, not of the sequences themselves.

5.2 Model

Table 5.1: Glossary of specialized terms used throughout this article.

Attribute grammar	An attribute grammar is a context free grammar augmented with attributes, semantic rules, and conditions. Attribute grammars were developed as a means of formalizing the semantics of a context free grammar.
Context free grammar	A context free grammar is a quadruple (V, Σ, P, S) where V is a finite set of non-terminal symbols, Σ (the alphabet) is a finite set of terminal symbols, P is a finite set of rules, and S is a distinguished element of V called the start symbol. A rule P is of the following form $A \rightarrow \omega$ where A is a single non-terminal symbol and ω is a string of terminals and/or non-terminals (possibly empty). The term "context-free" expresses the fact that non-terminals are rewritten without regard to the context in which they occur.
Cusp bifurcation	A codimension 2 bifurcation formed by the tangential meeting of two loci of saddle-node bifurcations. In other words, a cusp bifurcation traces the path of the points bounding a bistable region as they change with changes in two parameters. Bistability is implied within the cusp bounds.
Direct left recursion	A direct left recursion in context free grammar refers to rules of the form $A \rightarrow A\omega$. Parsing left recursion can possibly lead the parser down an infinite branch of the search tree in the corresponding logic program.
PoPS	The measurement of polymerase per second transcribing past a defined point of DNA.
SBML	The Systems Biology Markup Language (SBML) is a machine-readable language, based on XML, for representing models of biochemical reaction networks.
Semantics	Semantics reveals the meaning of syntactically valid strings in a language. For natural languages, this means correlating sentences and phrases with the objects, thoughts, and feelings of our experiences. For programming languages, semantics describes the behavior that a computer follows when executing a program in the language.
Syntax	Syntax refers to the ways symbols may be combined to create well-formed sentences (or programs) in a language. Syntax defines the formal relations between the constituents of a language, thereby providing a structural description of the various expressions that make up legal strings in the language. Syntax deals solely with the form and structure of symbols in a language without any consideration given to their meaning.

We recently described a fairly simple syntactic model of synthetic DNA sequences [Cai et al. 2007] capable of generating a large number of previously published synthetic genetic con-

structs [Elowitz and Leibler 2000; Gardner, Cantor, and Collins 2000; Guet et al. 2002]. We have now enhanced this initial syntactic model with a formal semantic model capable of expressing the dynamics of the molecular mechanisms coded by the DNA sequences. Specialized terms like syntax, semantics, and others are defined in Table 5.1. Our approach uses attribute grammars [Paakki 1995], a theoretical framework developed in the 60s to establish a formal correspondence between the text of a computer program and the series of microprocessor operations it codes for [Knuth 1968; Knuth 1990]. Even though other types of semantic models have been developed since then [Slonneger and Kurtz 1995; Stoy 1977], attribute grammars still represent a good compromise between simplicity and expressivity, an important characteristic to ensure that the framework can be used by non-computer scientists. Attribute grammars make it possible to use well characterized compilation algorithms to translate a DNA sequence into a mathematical model of the molecular interactions it codes for. As the static source code of a program directs the dynamic series of operations carried out by the microprocessor based on user inputs, the compilation process translates the static information of cells coded by DNA sequences into a dynamical model of the development of a phenotype in response to environmental influences [Lewontin 2001].

The translation of a gene network model from a genetic sequence is very similar to the compilation of the source code of a computer program into an object code that can be executed by a microprocessor (Figure 5.1). The first step consists in breaking down the DNA sequence into a series of genetic parts by a program called the lexer or scanner. Since the sequence of a part may be contained in the sequence of another part, the lexer is capable of backtracking to generate all the possible interpretations of the input DNA sequences as a series of parts. All possible combinations of parts generated by the lexer are sent to a second program called the parser to analyze if they are structurally consistent with the language syntax. The structure of a valid series of parts is represented by a parse tree [Cai et al. 2007] (Figure 5.2). The semantic evaluation takes advantage of the parse tree to translate the DNA sequence into a different representation such as a chemical reaction network. The translation process requires attributes and semantic actions. Attributes are properties of individual genetic parts or combinations of parts. Semantic actions are associated with the grammar production rules. They specify how attributes are computed. Specifically, the translation process relies on the semantic actions associated with parse tree nodes to synthesize the attributes of the construct from the attributes of its child nodes, or to inherit the attributes from its parental node. In our implementation, the product of the translation is a mass action model of the network of molecular interactions encoded in the DNA sequence. By using the standardized format of Systems Biology Markup Language (SBML), the model can be analyzed using existing simulation engines [Adalsteinsson, McMillen, and Elston 2004; Griffith et al. 2006; Hoops et al. 2006].

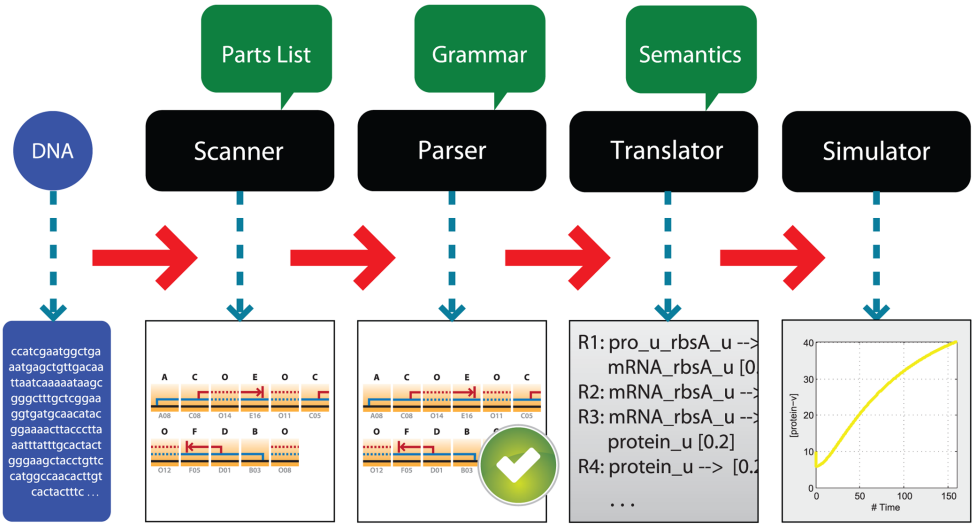


Figure 5.1: Workflow of generating the gene network model encoded in a DNA sequence. The input for this process is a DNA sequence that is first broken down into parts by the scanner. The combination of the parts is validated by the parser according to a syntactic model. After validation by the parser, the sequence is translated by applying semantic actions attached to the rules to transform the series of parts into a set of chemical equations. The resulting equations can then be solved using existing simulation engines. Each step takes the output of the previous step as input, so the workflow can start from any step if the appropriate input is provided.

5.3 Results

5.3.1 Compilation of a DNA sequence

We have developed a simple grammar compact enough to be presented extensively, yet sufficiently complex to represent basic epistatic interactions. The grammar generates constructs composed of one or more gene expression cassettes. The gene expression cassettes are themselves composed of a promoter, cistron, and transcription terminator. Finally, a cistron is composed of a Ribosome Binding Site (RBS) and a coding sequence (gene). The syntax is composed of 12 production rules (P1 to P12) displayed in bold characters in Figure 5.3 where each entry is composed of a rewriting rule (bold), and semantic actions (curly brackets). The symbol ϵrefers to an empty string, [,] to a list, [] to an empty list, and the '+' sign indicates the concatenation operation on two lists. This syntax is comparable to the one described previously [Cai et al. 2007] except that we introduced the extra non-terminal restConstructs to allow the generation of constructs with multiple cassettes without introducing parsing problems due to direct left recursions [Moore 2000].

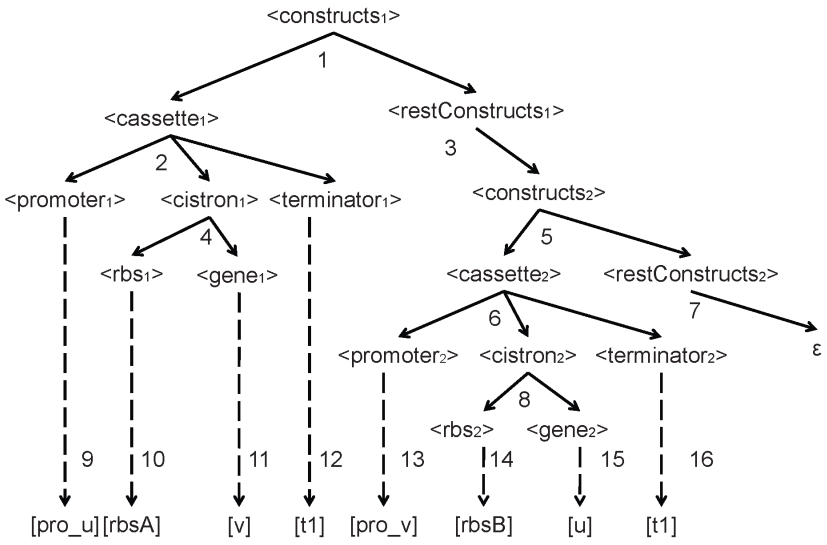


Figure 5.2: Parse tree showing the derivation process of a two-cassette genetic construct. In the derivation tree, terms in <> corresponds to the non-terminals in the grammar, while terms in [ ] are terminals, and the dashed lines indicate the transformation to terminals. The subscripts are used to distinguish different instances of the same category.

The attributes of a part include the kinetic rates related to this part and the interaction information. For example, the attributes of a promoter include a transcription rate along with a list of proteins repressing it and the kinetic parameters of the protein-DNA interactions. For non-terminal variables corresponding to combinations of parts such as cistrons, the attributes include a list of proteins, a list of promoters, and a list of chemical equations. The equation list is used to store the model of the system behavior, while the lists of promoters and proteins are recorded for computing the molecular interactions resulting from the DNA sequence. The complete set of attributes used in this simple grammar is listed in 5.2.

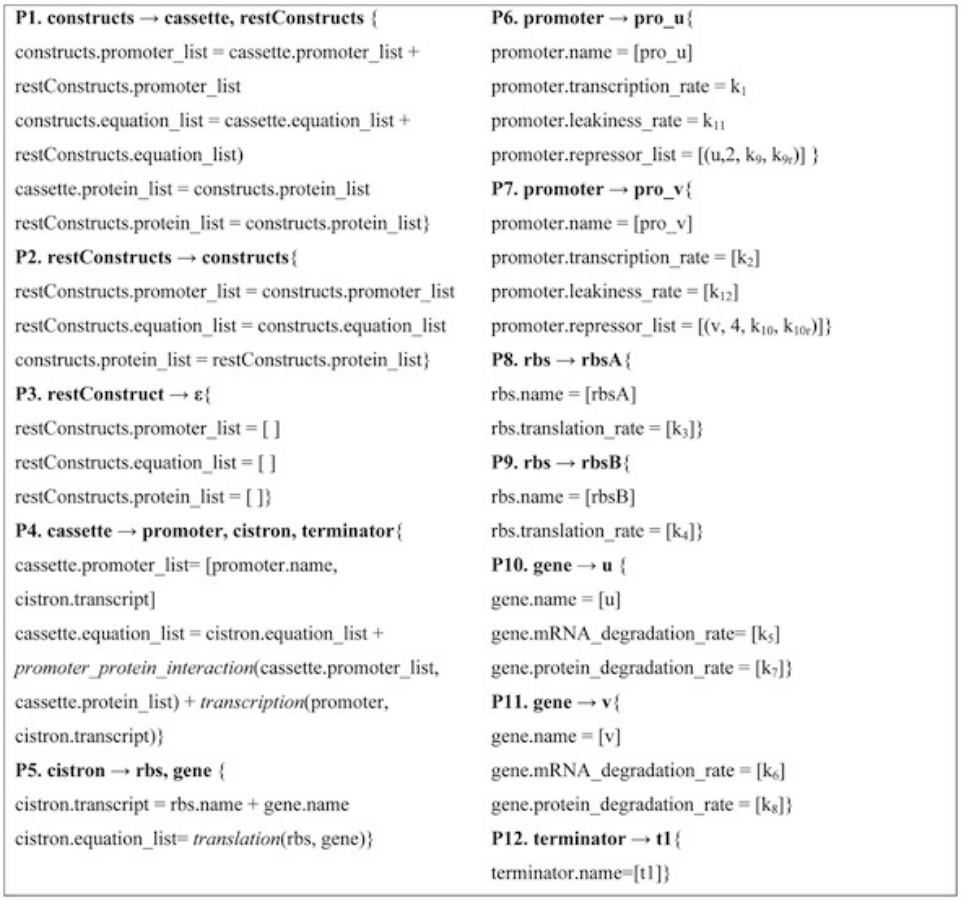


Figure 5.3: An example of attribute grammar.

Table 5.2: Attributes associated with non-terminals.

Non-terminals	Inherited Attribute	Synthesized Attributes
constructs	protein_list	promoter_list, equation_list
cassette	protein_list	promoter_list, equation_list
restConstructs	protein_list	promoter_list, equation_list
cistron	protein_list	transcript, equation_list
promoter	-	name, transcription_rate, leakiness_rate, repressor_list
RBS	-	name, translation_rate
gene	-	name, mRNA_degradation_rate, protein_degradation_rate
terminator	-	name

If many attributes can be computed locally by only considering a small fragment of the DNA sequence, other attributes are global properties of the system. For instance, the computation of protein-DNA interactions requires access to a global list of proteins expressed by the constructs. However, this list is not available until all of the different cassettes have been parsed. The problem is overcome by using a multiple-pass compilation method. In the first pass, the compiler does not do any structural validation but builds the list of proteins in the system and passes the list as an inherited attribute to the second pass. In the second pass, the promoter-protein interactions can be calculated locally at the level of each cassette. Rules P1 to P5 define the structure of a design, while rules P6 to P12 cover the selection of a specific part for each category. In the semantic action, the relation between an attribute and its variable is indicated by a dot and constants are enclosed by brackets. For instance, $gene.mRNA\_degradation\_rate = [k6]$ indicates that the value of the attribute mRNA_degradation_rate of a gene is a constant k6. The attribute repressor_list used in P6 and P7 includes the name of the repressor, the stoichiometry, and the kinetic constants of the forward and reverse reactions of the protein-DNA interaction. Appendix B details the parsing steps and computational dependence of each step. Finally, the equation writing operations are handled by functions typed in italics in Figure 5.3 and defined in Figure 5.4.

P1. constructs → cassette, restConstructs {
constructs.promoter_list = cassette.promoter_list +
restConstructs.promoter_list
constructs.equation_list = cassette.equation_list +
restConstructs.equation_list)
cassette.protein_list = constructs.protein_list
restConstructs.protein_list = constructs.protein_list}
P2. restConstructs → constructs{
restConstructs.promoter_list = constructs.promoter_list
restConstructs.equation_list = constructs.equation_list
constructs.protein_list = restConstructs.protein_list}
P3. restConstruct → ε{
restConstructs.promoter_list = [ ]
restConstructs.equation_list = [ ]
restConstructs.protein_list = [ ]}
P4. cassette → promoter, cistron, terminator{
cassette.promoter_list= [promoter.name,
cistron.transcript]
cassette.equation_list = cistron.equation_list +
promoter_protein_interaction(cassette.promoter_list,
cassette.protein_list) + *transcription*(promoter,
cistron.transcript)}
P5. cistron → rbs, gene {
cistron.transcript = rbs.name + gene.name
cistron.equation_list= *translation*(rbs, gene)}

P6. promoter → pro_u{
promoter.name = [pro_u]
promoter.transcription_rate = k_1
promoter.leakiness_rate = k_{11}
promoter.repressor_list = [(u,2, k_9, k_{9r})] }
P7. promoter → pro_v{
promoter.name = [pro_v]
promoter.transcription_rate = [k_2]
promoter.leakiness_rate = [k_{12}]
promoter.repressor_list = [(v, 4, k_{10}, k_{10r})]}
P8. rbs → rbsA{
rbs.name = [rbsA]
rbs.translation_rate = [k_3]}
P9. rbs → rbsB{
rbs.name = [rbsB]
rbs.translation_rate = [k_4]}
P10. gene → u {
gene.name = [u]
gene.mRNA_degradation_rate= [k_5]
gene.protein_degradation_rate = [k_7]}
P11. gene → v{
gene.name = [v]
gene.mRNA_degradation_rate = [k_6]
gene.protein_degradation_rate = [k_8]}
P12. terminator → t1{
terminator.name=[t1]}

Figure 5.4: Equation generators.

The translation of the DNA sequence into a mathematical model is available as the equation_list attribute of constructs. The model outputs are generated by equations generators, which are purposely decoupled from the semantic actions. The decoupling enables the flexibility of using different equation formats to describe a biological process. The translation of the construct composed of the parts pro_u rbsA gene_v t1 pro_v rbsB gene_u t1 generates the equations displayed in the [Reactions] section of Figure 5.5. Each line is composed of a reaction index (R1 to R12), the chemical equation itself, and one or two reaction parameters depending on the reaction reversibility. The initial values have been computed by assigning 1 to variables representing DNA sequences and prompting the user to set the initial condition of proteins. The scripts and data used in this report are available in `http://files.figshare.com/436347/Dataset_S1.zip`.

[Reactions]

R1: pro_u_rbsA_v -->pro_u_rbsA_v + mRNA_rbsA_v [k_1]

R2: mRNA_rbsA_v --> [k_6]

R3: mRNA_rbsA_v -->mRNA_rbsA_v + protein_v [k_3]

R4: protein_v --> [k_8]

R5: pro_v_rbsB_u -->pro_v_rbsB_u + mRNA_rbsB_u [k_2]

R6: mRNA_rbsB_u --> [k_5]

R7: mRNA_rbsB_u -->mRNA_rbsB_u + protein_u [k_4]

R8: protein_u --> [k_7]

R9: pro_v_rbsB_u +4protein_v <-->pro_v_rbsB_u_x [k_{10}, k_{10r}]

R10: pro_v_rbsB_u_x -->pro_v_rbsB_u_x + mRNA_rbsB_u [k_{12}]

R11: pro_u_rbsA_v +2protein_u <-->pro_u_rbsA_v_x [k_9, k_{9r}]

R12: pro_u_rbsA_v_x -->pro_u_rbsA_v_x + mRNA_rbsA_v [k_{11}]

[InitialValues]

pro_u_rbsA_v= 1

pro_v_rbsB_u = 1

protein_v = user input

protein_u = user input

Figure 5.5: Chemical equations translated from a DNA sequence.

5.3.2 Expressing context-dependencies of parts function

The semantic model presented in the previous section is completely modular since the parameters of the model describing the construct behavior are attributes of individual parts, not of higher order structures. For instance, in the previous model (Figures 5.3, 5.4), translational efficiency is primarily determined by the RBS sequence [Shultzaberger et al. 2001; Vellanoweth and Rabinowitz 1992]. This association between RBS and translation rate was successfully used to design one of the first artificial gene networks [Gardner, Cantor, and Collins 2000] and is still used by many synthetic biology software applications [Hill et al. 2008; Marchisio and Stelling 2008; Pedersen and Phillips 2009; Rodrigo and Jaramillo 2007]. Yet, it is also well known that translation initiation can be attenuated by stable mRNA sec-

ondary structures [De Smit and Van Duin 1994; Kudla et al. 2009; Smit and Van Duin 1990]. This leads to a situation where a translational rate can no longer be considered the attribute of an individual part but needs to be considered as the attribute of a specific combination of parts. This type of context-dependency can naturally be expressed using attribute grammars since the translation reaction is computed at the cistron level, not at the level of individual parts. Rule P5 of Figure 5.3 can be modified by introducing a new function to retrieve the translation rate for specific combination of gene and RBS.

```
P5. cistron --> rbs, gene
{
?cistron.translation_rate = get_translation_rate (rbs, gene)
?cistron.transcript = rbs.name+gene.name
?cistron.equation_list = translation(rbs, gene, cistron.translation_rate)
}

The get_translation_rate function checks for specific cases of interactions
between an RBS and coding sequence first. If none is found, then the
default RBS translation rate is used.

If exists translation_rate(rbs, gene)
?translation_rate = translation_rate(rbs, gene)
else
?translation_rate = translation_rate(rbs)
endif
```

This approach is illustrated in Table 5.3 using previously published data demonstrating the interference between the RBS and coding sequence [Smit and Van Duin 1990]. Specifically, this report provides the relation expression observed in 23 different constructs generated by combining different variants of the RBS and MS2 coat protein gene. This data set has been reorganized in Table 5.3 by sorting the constructs according to the RBS and gene variants they used. Three of the constructs using the WT RBS sequence resulted in a maximum level of expression while the expression of the gene variants ORF4, ORF5, and ORF6 were expressed at a much lower level due to the greater stability of the mRNA secondary structure. A similar pattern is observed for other RBS variants (RBS1, RBS2, RBS3, RBS7). For all of these RBS variants, it is possible to define the translation_rate function by associating the default translation rate with the maximum expression rate. Specific translation rates associated with particular pairs of RBS and gene variants are recorded separately.

5.3.3 Exploration of genetic design space

The semantic model in Figures 5.3 and 5.4 is a compact proof of concept example, but it does not capture a number of features commonly found in actual genetic constructs. In order to

Table 5.3: Context-dependency of experimentally determined translation rates.

heightMutant	RBS	ORF	Expression	Translation rate function
1	RBS WT	ORF WT	100	translation_rate(RBS WT)
6	RBS WT	ORF2	100	translation_rate(RBS WT)
7	RBS WT	ORF3	100	translation_rate(RBS WT)
17	RBS WT	ORF4	3	translation_rate(RBS WT, ORF4)
20	RBS WT	ORF5	6	translation_rate(RBS WT, ORF5)
23	RBS WT	ORF6	0.3	translation_rate(RBS WT, ORF6)
4	RBS1	ORF WT	100	translation_rate(RBS1)
2	RBS1	ORF1	100	translation_rate(RBS1)
3	RBS1	ORF2	100	translation_rate(RBS1)
5	RBS1	ORF3	4	translation_rate(RBS1, ORF3)
14	RBS1	ORF4	< 0.003	translation_rate(RBS1, ORF4)
9	RBS2	ORF WT	100	translation_rate(RBS2)
8	RBS2	ORF1	100	translation_rate(RBS2)
10	RBS2	ORF3	100	translation_rate(RBS2)
12	RBS3	ORF WT	100	translation_rate(RBS3)
11	RBS3	ORF1	20	translation_rate(RBS3, ORF1)
13	RBS3	ORF3	100	translation_rate(RBS3)
15	RBS4	ORF4	0.1	translation_rate(RBS4)
16	RBS5	ORF4	0.05	translation_rate(RBS5)
22	RBS6	ORF WT	0.2	translation_rate(RBS6, ORF WT)
18	RBS6	ORF4	80	translation_rate(RBS6)
21	RBS7	ORF WT	100	translation_rate(RBS7)
19	RBS7	ORF4	100	translation_rate(RBS7)

demonstrate that our approach is capable of modeling more realistic DNA sequences, we have extended this semantic model (Supplementary Materials) to translate the DNA sequences of previously published DNA plasmids that include polycistronic cassettes in different orientations [Gardner, Cantor, and Collins 2000]. This plasmid library was generated by 32 different genetic parts (three promoters: pLtetO-1, pLs1con, ptrc-2; eight RBS: rbsA to rbsH; and four genes: tetR, cIts, lacI, and gfp and one terminator, all in both orientations). The syntax generates 72 different single gene expression constructs in each orientation. By combining two genes repressing each other in a construct, it is possible to make bistable artificial gene networks that are represented in Figure 5.6. These bistable networks can be used as a genetic switch.

To demonstrate the potential use of a semantic model to search for a desirable behavior in a large genetic design space, we have generated the DNA sequences of all 41,472 possible sequences ($722*8$ RBS for the reporter gene) having the same structure as previously described

switches. All sequences were translated into separate model files and a script was developed to perform a bistability analysis of each model. Parameters of the semantic model were obtained by qualitatively matching the experimental results of the six previously published switches [Gardner, Cantor, and Collins 2000] and are summarized in Appendix B . Most of the automatically generated sequences led to inherently non-bistable networks because the necessary repressor/promoter pairs did not match. Since this specific example is particularly well understood, we could have generated a limited number of targeted constructs. Yet, we chose to generate all possible sequences to demonstrate the generality of our approach. In particular, it was important to evaluate the computational cost of generating and translating DNA sequences to ensure that it would not prevent a systematic exploration of more complex design spaces. It takes only minutes to generate 41,472 sequences and translate them into SBML files. Hence, the computational cost of this step is negligible compared to the time required by the simulation of the SBML files.

Bistability was tested numerically by integrating the differential equations until they converged to a steady state starting from two different initial conditions. The two initial conditions started with one protein level very high and the other very low and vice versa. We characterized the bistability by computing the ratio of reporter concentration for the two steady state values. In order to globally verify the behavior of this large population of models, we focused on the 3,072 constructs potentially capable of bistability, 1,408 of which were found to be bistable. We further reduced the number of constructs used to verify the translation process from 3,072 to 384 by assuming that two constructs differing only in the RBS in 5´ of the reporter gene would produce the same ratio of steady state values. Figure 5.6 visualizes the behavior of these 384 constructs. Constructs that are not bistable have a ratio of 1. This ratio gives insight into how the construct is expected to be experimentally detectable. Since most experimental methods cannot give an exact value of protein concentration, a high ratio is desired to rise above experimental noise. Each of the 6 windows is analogous to the previously described two-parameter bifurcation diagram for that pair of repressors [Gardner, Cantor, and Collins 2000]. This gives confidence that both the semantic model of DNA sequences and the compiler used to translate automatically generated DNA sequences give results consistent with manually developed models of this family of gene networks. In the long term, the advantage to our approach over a traditional two-parameter bifurcation is the association of discrete parameter values with specific parts. This will prove particularly valuable when the context-dependencies of parameter values are better documented experimentally.

This example demonstrates the benefit of building a semantic model of synthetic DNA sequences. Even a small library of genetic parts can generate large numbers of artificial gene networks having no more than a few interacting genes. A syntactic model describing how parts can be combined into constructs is a compact representation of the genetic design space generated from the parts library. While it is possible to manually build mathematical models capturing the dynamics of some of these artificial gene networks individually, it becomes desirable to automate the process to ensure the model consistency when building

large families of related models derived from the same parts library. By considering genetic parts as the terminal symbols of an attribute grammar, it becomes possible to automatically generate models of numerous artificial gene networks derived from this parts library and quickly identify the optimal designs [Goler, Bramlett, and Peccoud 2008].

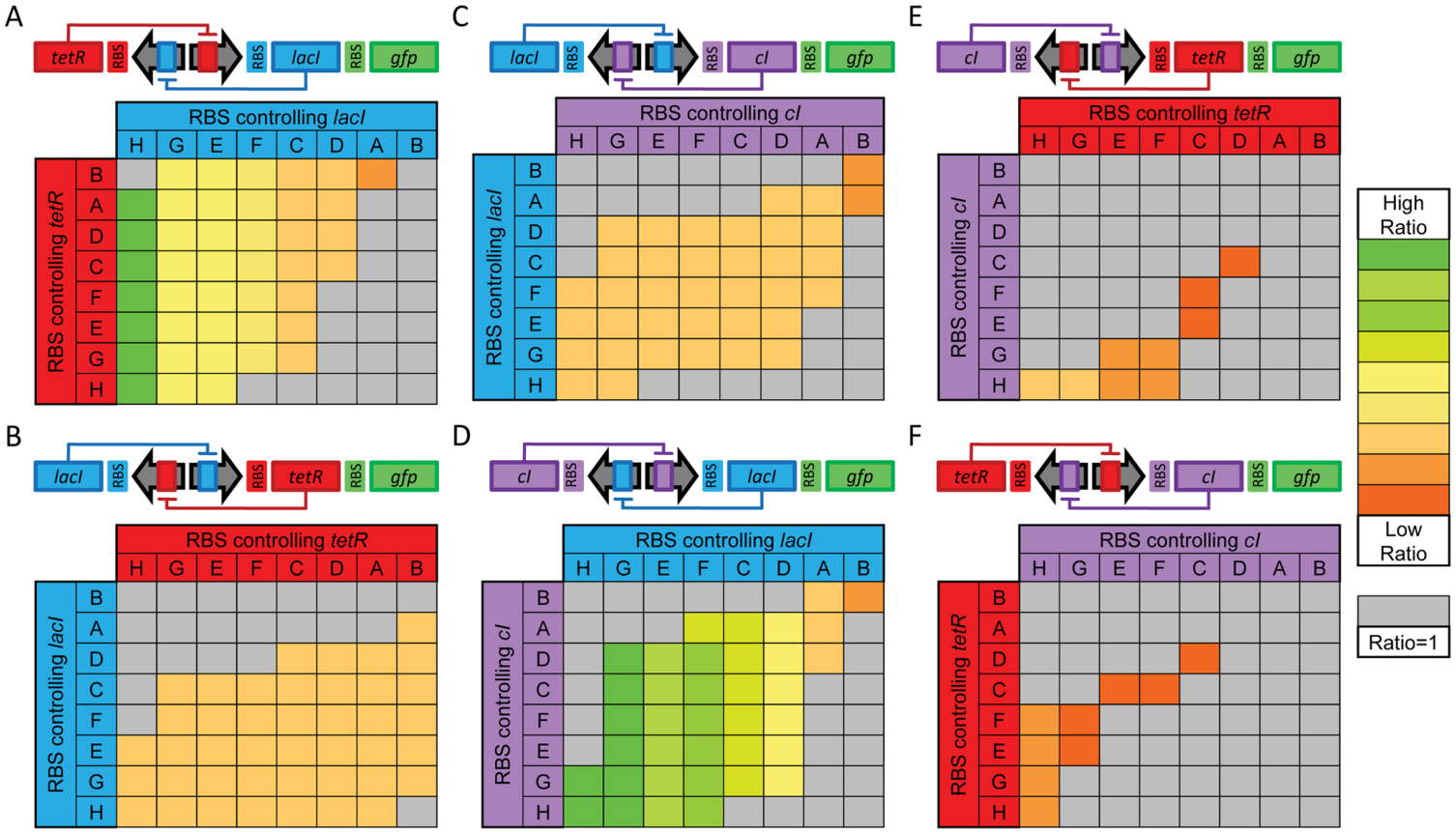
5.4 Discussion

5.4.1 Computer assisted design of synthetic genetic constructs

The parameter values used in the previous example were selected to match an extremely small set of six experimental data points. Although the under-determination of the model does not make it possible to precisely estimate the value of these parameters, the example illustrates how the framework could provide valuable guidance in selecting specific parts for a design. Considering that the exact value of parameters for parts is still a far off perspective, the automatic exploration of the design space presented here will provide useful guidance in construct design. For example, robust constructs from the cusp interior of the tetR/cI and lacI/cI pairings could be built and tested while less robust switches based on the lacI/tetR pairing would be avoided. As more is learned about these parts including the specific rates in different genetic contexts, the predictive ability of such maps will increase. Other motifs could be explored in a similar manner. For example, oscillators [Stricker et al. 2008] could be explored by permuting parts and calculating the model-predicted existence of oscillations

---

Figure 5.6 *(facing page)*: Mapping the behavior of 384 genetic constructs. Each section A to F indicates a different selection of repressors within a toggle switch: (A) tetR and lacI, (B) lacI and tetR, (C) lacI and cI, (D) cI and lacI, (E) cI and tetR, and (F) tetR and cI. Other networks that cannot give rise to bistability (e.g. a construct with tetR as both genes) are excluded as are designs that only vary the GFP RBS (see text). Each pair is explored by varying the RBS (ordered by translational efficiency from low (RBS H) to high (RBS B) as determined by a qualitative fit of the results of Gardner et al. [Gardner, Cantor, and Collins 2000] with consistent letter-based labels) and calculating the detectability ratio, defined as the steady state GFP concentration in the "on" state divided by the concentration in the "off" state. These ratios are displayed using a color map as indicated by the legend to the right. Monostable constructs have a ratio of 1 and are indicated by gray boxes. The ratio gives a measure of how easily the two steady states can be distinguished, which is important due to high experimental noise. Each pane also elucidates the traditional two-parameter bifurcation diagram of each gene pair as the translational rates are varied by changing RBSs. Constructs near the edge of the cusp operate near saddle-node bifurcations and are more prone to noise-induced switching. Thus, constructs from the cusp interior are preferred for robust behavior.


A
tetR
RBS
lacI
gfp
RBS controlling lacI
RBS controlling tetR
B
lacI
tetR
RBS controlling tetR
RBS controlling lacI
C
lacI
cI
RBS controlling cI
RBS controlling lacI
D
cI
lacI
RBS controlling lacI
RBS controlling cI
E
cI
tetR
RBS controlling tetR
RBS controlling cI
F
tetR
cI
RBS controlling cI
RBS controlling tetR
H G E F C D A B
B A D C F E G H
High Ratio
Low Ratio
Ratio=1

as well as their period or amplitude.

The approach presented in this report will be implemented into GenoCAD [Czar, Cai, and Peccoud 2009], the web-based tool we have developed to give biologists access to our syntactic design framework. Through GenoCAD, users will benefit from the syntactic and semantic models of various parts sources (GenoCAD provided library, MIT Registry of Standard Biological Parts, or user created parts library). Initially, users will be able to translate their designs into SBML files that could be imported in SBML-compliant simulation tools (`www.sbml.org/SBML_Software_Guide`) for further analysis. At a later stage, simulation results and more advanced numerical analyses will be seamlessly integrated in GenoCAD's workflow. One of the major obstacles toward the implementation of such semantic models in GenoCAD is the development of a data model allowing users to understand and possibly edit the functional model of the parts they use.

A function description language called Genetic Engineering of living Cells (GEC) was recently introduced to specify the properties of a design [Pedersen and Phillips 2009]. GEC is capable of finding a DNA sequence that implements the desirable phenotypic functions. Several other software applications have been recently released to design biological systems from standardized genetic parts. ASMPART [Rodrigo and Jaramillo 2007], SynBioSS [Hill et al. 2008], a specialized ProMot package [Marchisio and Stelling 2008] and TinkerCell (`www.tinkercell.com`) illustrate this trend. These tools are still exploratory. One of their limitations is the requirement to define parts in a specialized format, such as SBML or Modeling Description Language (MDL). Furthermore, instead of defining parts interactions in the underlying parts data models, these tools rely on the user to manually define them textually [Hill et al. 2008] or graphically [Marchisio and Stelling 2008]. As a result of this specific limitation, several of these tools do not appear suitable for the automatic exploration of a design space. Moreover, they tend to rely on a loosely defined relationship between the structure of the genetic constructs and their behavior. They allow parts to be assembled in any order without regard for biological viability.

Still, the scripts developed to generate our results are of lesser importance than the application of the theory of semantics-based translation using attribute grammars to the translation of DNA sequences into dynamical models representing the molecular interactions they encode. Since this approach is used to develop the compilers of many computer languages [Appel and Palsberg 2007; Slonneger and Kurtz 1995], a wealth of existing theoretical results and software tools can find new applications in the life sciences. For instance, we have implemented semantic models of DNA sequences into two widely used but very different programming environments, Prolog [Bratko 2001] and ANTLR [Parr 2007]. Future research efforts will need to investigate the pros and cons of different compiler generators and different parsing algorithms for analyzing even genome-scale DNA sequences and how they impact the ability of grammars to express various features of DNA sequences. Also, the type of attributes associated with parts is flexible. Here we primarily use mass action kinetic rates as attributes, but we could just as easily have used an emerging synthetic biology unit like polymerase per second (PoPS) [Canton, Labno, and Endy 2008; Kelly et al. 2009].

Ultimately, tools capable of automatically generating models of the behavior of synthetic DNA sequences will be important for the advancement of synthetic biology [Goler, Bramlett, and Peccoud 2008]. However, these tools will need to be able to express that the contribution of a genetic part to the phenotype of an organism depends largely on the local and global context in which it is placed. The interference between RBS and coding sequence is just one example of the biological complexity that computer assisted design applications will have to properly consider.

5.4.2 Functional characterization of genetic parts

Before it will be used to build synthetic genetic systems meeting user-defined specifications, the semantic model of DNA sequences presented in this report will be instrumental in the quantitative characterization of structure-function relationships in synthetic DNA sequences. The vision of applying quantitative engineering methods to biological problems has been recognized as a promising avenue to biological discovery [Drubin, Way, and Silver 2007]. The critical role of artificial gene networks in the characterization of molecular noise affecting the dynamics of gene networks [Raj and Oudenaarden 2008] illustrates the potential of synthetic biology as a route to refine the understanding of basic biological processes.

Ongoing efforts aim to carefully define how parts should fit together syntactically and what attributes are needed to characterize their function. For example, the sequence between the RBS and the start codon has been shown to play an important role in translation rate [Vellanoweth and Rabinowitz 1992]. The question arises whether the RBS should be defined to include the spacing, or if there should be a separate parts category for the spacer. The rapid development of gene synthesis techniques [Czar, Cai, and Peccoud 2009] will make it possible to investigate these questions with a base-level resolution. Beyond libraries of parts for designing expression vectors, similar curation efforts could lead to the identification of parts in genomic sequences, whereby the hypothetical function of these parts as they are expressed in attribute grammars could be tested by genome refactoring [Chan, Kosuri, and Endy 2005].

Acknowledgments

The authors would like to thank Drs. Jacques Cohen and Mark Cooper for their critical reading of an early version of this manuscript, Stephan Hoops for helping us use COPASI in batch mode, and Emily Alberts for her editorial skills.

Author Contributions

Conceived and designed the experiments: JP.
Performed the experiments: YC MWL LA.
Analyzed the data: YC MWL JP. Wrote the paper: YC MWL JP.

References

[1] David Adalsteinsson, David McMillen, and Timothy C Elston. "Biochemical Network Stochastic Simulator (BioNetS): software for stochastic modeling of biochemical networks". In: *BMC bioinformatics* 5.1 (2004), p. 24.

[2] K L Agarwal et al. "Total synthesis of the gene for an alanine transfer ribonucleic acid from yeast". In: *Nature* 227 (1970), pp. 27–34.

[3] Andrew W Appel and Jens Palsberg. *Modern compiler implementation in Java*. Cambridge University Press New York, 2007. ISBN: 8175960728.

[4] William Bains. "The parts list of life". In: *Nature biotechnology* 19.5 (2001), pp. 401–402.

[5] Elena Beisswanger et al. "Gene Regulation Ontology (GRO): design principles and use cases". In: *Studies in health technology and informatics* 136 (2008), p. 9.

[6] Simone Bentolila. "A grammar describing 'biological binding operators' to model gene regulation". In: *Biochimie* 78.5 (1996), pp. 335–350.

[7] Michael A Brasch, James L Hartley, and Marc Vidal. "ORFeome cloning and systems biology: standardized mass production of the parts from the parts-list". In: *Genome research* 14.10b (2004), pp. 2001–2009.

[8] Ivan Bratko. *Prolog: programming for artificial intelligence*. Addison-Wesley, 2001. ISBN: 0201403757.

[9] Yizhi Cai et al. "A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts." In: *Bioinformatics (Oxford, England)* 23.20 (Oct. 2007), pp. 2760–7. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btm446.

[10] Barry Canton, Anna Labno, and Drew Endy. "Refinement and standardization of synthetic biological parts and devices". In: *Nature biotechnology* 26.7 (2008), pp. 787–93. ISSN: 1546-1696. DOI: 10.1038/nbt1413.

[11] Leon Y Chan, Sriram Kosuri, and Drew Endy. "Refactoring bacteriophage T7". In: *Molecular systems biology* 1.1 (2005).

[12] KC Katherine C Chen et al. "Integrative analysis of cell cycle control in budding yeast." In: *Molecular biology of the cell* 15.8 (Aug. 2004), pp. 3841–3862. ISSN: 1059-1524. DOI: 10.1091/mbc.E03.

[13] David Chiang, AK Aravind K Joshi, and David B DB Searls. "Grammatical representations of macromolecular structure". In: *J Comput Biol* 13.5 (June 2006), pp. 1077–1100. DOI: 10.1089/cmb.2006.13.1077.

[14] J Robert Coleman et al. "Virus attenuation by genome-scale changes in codon pair bias". In: *Science* 320.5884 (2008), pp. 1784–1787.

[15] Robert Sidney Cox, Michael G Surette, and Michael B Elowitz. "Programming gene expression with combinatorial promoters". In: *Molecular systems biology* 3.1 (2007).

[16] Michael J Czar, Yizhi Cai, and Jean Peccoud. "Writing DNA with GenoCAD." In: *Nucleic acids research* 37.Web Server issue (July 2009), W40–7. ISSN: 1362-4962. DOI: 10.1093/nar/gkp361.

[17] Maarten H De Smit and Jan Van Duin. "Control of translation by mRNA secondary structure in Escherichia coli. A quantitative analysis of literature data". In: *Journal of molecular biology* 244 (1994), p. 144.

[18] Shan Dong and David B DB Searls. "Gene structure prediction by linguistic methods". In: *Genomics* 23.3 (1994), pp. 540–551.

[19] David A Drubin, Jeffrey C Way, and Pamela A Silver. "Designing biological systems". In: *Genes & development* 21.3 (2007), pp. 242–254.

[20] Jessica S Dymond et al. "Teaching synthetic biology, bioinformatics and engineering to undergraduates: the interdisciplinary Build-a-Genome course". In: *Genetics* 181.1 (2009), pp. 13–21.

[21] Ulrike S Eggert, Timothy J Mitchison, and Christine M Field. "Animal cytokinesis: from parts list to mechanisms". In: *Annu. Rev. Biochem.* 75 (2006), pp. 543–566.

[22] Karen Eilbeck et al. "The Sequence Ontology: a tool for the unification of genome annotations". In: *Genome Biol* 6.5 (2005), R44.

[23] M B Elowitz and S Leibler. "A synthetic oscillatory network of transcriptional regulators." In: *Nature* 403.6767 (Jan. 2000), pp. 335–8. ISSN: 0028-0836. DOI: 10.1038/35002125.

[24] Drew Endy. "Foundations for engineering biology". In: *Nature* 438.7067 (2005), pp. 449–53. ISSN: 1476-4687. DOI: 10.1038/nature04342.

[25] Douglas S Falconer, Trudy F C Mackay, and Richard Frankham. "Introduction to Quantitative Genetics (4th edn)". In: *Trends in Genetics* 12.7 (1996), p. 280.

[26] Nicholas C Fitzkee et al. "Are proteins made from a limited parts list?" In: *Trends in biochemical sciences* 30.2 (2005), pp. 73–80.

[27] T S Gardner, C R Cantor, and J J Collins. "Construction of a genetic toggle switch in Escherichia coli." In: *Nature* 403.6767 (Jan. 2000), pp. 339–42. ISSN: 0028-0836. DOI: 10.1038/35002131.

[28] Jason Gertz, Eric D Siggia, and Barak A Cohen. "Analysis of combinatorial cis-regulation in synthetic and genomic promoters". In: *Nature* 457.7226 (2008), pp. 215–218.

[29] Daniel G Gibson et al. "One-step assembly in yeast of 25 overlapping DNA fragments to form a complete synthetic Mycoplasma genitalium genome". In: (2008).

[30] Mario Gimona. "Protein linguistics - a grammar for modular protein assembly?" In: *Nat Rev Mol Cell Biol* 7.1 (Jan. 2006), pp. 68–73. ISSN: 1471-0072. DOI: 10.1038/nrm1785.

[31] JA Jonathan A Goler, Brian W BW Bramlett, and Jean Peccoud. "Genetic design: rising above the sequence". In: *Trends in biotechnology* 26.10 (Oct. 2008), pp. 538–544. ISSN: 0167-7799. DOI: 10.1016/j.tibtech.2008.06.003.

[32] Mark Griffith et al. "Dynamic partitioning for hybrid simulation of the bistable HIV-1 transactivation network." In: *Bioinformatics (Oxford, England)* 22.22 (Nov. 2006), pp. 2782–9. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btl465.

[33] Călin C Guet et al. "Combinatorial synthesis of genetic networks". In: *Science* 296.5572 (2002), pp. 1466–1470.

[34] U Güldener et al. "CYGD: the comprehensive yeast genome database". In: *Nucleic acids research* 33.suppl 1 (2005), pp. D364–D368.

[35] Anthony D Hill et al. "SynBioSS: the synthetic biology modeling suite." In: *Bioinformatics (Oxford, England)* 24.21 (2008), pp. 2551–3. ISSN: 1460-2059. DOI: 10.1093/bioinformatics/btn468.

[36] Stefan Hoops et al. "COPASI—a complex pathway simulator". In: *Bioinformatics* 22.24 (Dec. 2006), pp. 3067–3074. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btl485.

[37] Lily E Kay. *Who Wrote the Book of Life?: History of the Genetic Code*. Stanford University Press, 2000. ISBN: 0804734178.

[38] Evelyn Fox Keller. *The century of the gene*. Harvard University Press, 2001. ISBN: 0674039432.

[39] Evelyn Fox Keller and David Harel. "Beyond the gene". In: *PLoS One* 2.11 (2007), e1231.

[40] Jason R Kelly et al. "Measuring the activity of BioBrick promoters using an in vivo reference standard". In: *Journal of biological engineering* 3.1 (2009), p. 4.

[41] Bjarne Knudsen and Jotun Hein. "Pfold: RNA secondary structure prediction using stochastic context-free grammars". In: *Nucleic acids research* 31.13 (2003), pp. 3423–3428.

[42] Donald E Knuth. "Semantics of context-free languages". In: *Mathematical systems theory* 2.2 (1968), pp. 127–145.

[43] Donald E Knuth. "The Genesis of Attribute Grammars". In: *Proceedings of the international conference on Attribute grammars and their applications*. Springer, 1990, pp. 1–12. ISBN: 3540531017.

[44] Grzegorz Kudla et al. "Coding-Sequence Determinants of Gene Expression in Escherichia coli". In: *science* 324.April (2009), pp. 255–258.

[45] Richard C Lewontin. *The triple helix: Gene, organism, and environment*. Harvard University Press, 2001. ISBN: 0674006771.

[46] Michael Lynch and Bruce Walsh. "Genetics and analysis of quantitative traits". In: (1998).

[47] M a Marchisio and J Stelling. "Computational design of synthetic gene circuits with composable parts." In: *Bioinformatics (Oxford, England)* 24.17 (2008), pp. 1903–10. ISSN: 1460-2059. DOI: 10.1093/bioinformatics/btn330.

[48] RC Moore. "Removing left recursion from context-free grammars". In: *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. 2000.

[49] Michael Mueller, Lennart Martens, and Rolf Apweiler. "Annotating the human proteome: beyond establishing a parts list". In: *Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics* 1774.2 (2007), pp. 175–191.

[50] K.F. Kevin F Murphy, Gábor Balázsi, and J.J. James J.J. James J Collins. "Combinatorial promoter design for engineering noisy gene expression". In: *Proceedings of the National Academy of Sciences* 104.31 (2007), p. 12726.

[51] Jukka Paakki. "Attribute Grammar Paradigms Language Implementation — A High-Level Methodology in". In: *Grammars* 27.June (1995), pp. 196–255.

[52] Terence Parr. "The Definitive ANTLR Reference: Building Domain-Specific Languages (Pragmatic Programmers)". In: *Pragmatic Bookshelf, May* (2007).

[53] Jean Peccoud et al. "Targeted development of registries of biological parts". In: *PLoS One* 3.7 (Jan. 2008), e2671. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0002671.

[54] Michael Pedersen and Andrew Phillips. "Towards programming languages for genetic engineering of living cells." In: *Journal of the Royal Society, Interface / the Royal Society* 6 Suppl 4.April (Aug. 2009), S437–50. ISSN: 1742-5662. DOI: 10.1098/rsif.2008.0516.focus.

[55] Michal Rabani, Michael Kertesz, and Eran Segal. "Computational prediction of RNA structural motifs involved in posttranscriptional regulatory processes". In: *Proceedings of the National Academy of Sciences* 105.39 (2008), pp. 14885–14890.

[56] Arjun Raj and Alexander van Oudenaarden. "Nature, nurture, or chance: stochastic gene expression and its consequences". In: *Cell* 135.2 (2008), pp. 216–226.

[57] Stephen A Ramsey et al. "Dual feedback loops in the GAL regulon suppress cellular heterogeneity in yeast". In: *Nature genetics* 38.9 (2006), pp. 1082–1087.

[58] Elena Rivas and Sean R Eddy. "The language of RNA: a formal grammar that includes pseudoknots". In: *Bioinformatics* 16.4 (Jan. 2000), pp. 334–340.

[59] Guillermo Rodrigo and Alfonso Jaramillo. "Computational design of digital and memory biological devices." In: *Systems and synthetic biology* 1.4 (Dec. 2007), pp. 183–95. ISSN: 1872-5325. DOI: 10.1007/s11693-008-9017-0.

[60] David B Searls. "The linguistics of DNA". In: *American Scientist* 80.6 (Jan. 1992), pp. 579–591.

[61] D.B. David B Searls. "Linguistic approaches to biological sequences". In: *Bioinformatics* 13.4 (1997), p. 333. ISSN: 1367-4803.

[62] D.B. David B Searls. "The language of genes". In: *Nature* 420.6912 (2002), pp. 211–217.

[63] Eran Segal et al. "A genomic code for nucleosome positioning". In: *Nature* 442.7104 (2006), pp. 772–778.

[64] Ryan K Shultzaberger et al. "Anatomy of Escherichia coli ribosome binding sites". In: *Journal of molecular biology* 313.1 (2001), pp. 215–228.

[65] Maxine Singer and Paul Berg. "George Beadle: from genes to proteins". In: *Nature Reviews Genetics* 5.12 (2004), pp. 949–954.

[66] Kenneth Slonneger and Barry L Kurtz. *Formal syntax and semantics of programming languages*. Addison-Wesley Reading, 1995. ISBN: 0201656973.

[67] Maarten H de Smit and J Van Duin. "Secondary structure of the ribosome binding site determines translational efficiency: a quantitative analysis". In: *Proceedings of the National Academy of Sciences* 87.19 (1990), pp. 7668–7672.

[68] C Neal Stewart Jr. "Plant functional genomics: beyond the parts list". In: *Trends in plant science* 10.12 (2005), p. 561.

[69] Joseph E Stoy. *Denotational semantics: the Scott-Strachey approach to programming language theory*. MIT press, 1977. ISBN: 0262191474.

[70] Jesse Stricker et al. "A fast, robust and tunable synthetic gene oscillator." In: *Nature* 456.7221 (Nov. 2008), pp. 516–9. ISSN: 1476-4687. DOI: 10.1038/nature07389.

[71] A.H. Sturtevant. *A History of Genetics.* Cold Spring Harbor Laboratory Press, 2001. ISBN: 0879696079.

[72] Edward L Tatum. "A Case History in Biological Research Chance and the exchange of ideas played roles in the discovery that genes control biochemical events". In: *Science* 129.3365 (1959), pp. 1711–1715.

[73] Marcel Tigges et al. "A tunable synthetic mammalian oscillator". In: *Nature* 457.7227 (2009), pp. 309–312.

[74] Robert Luis Vellanoweth and Jesse C Rabinowitz. "The influence of ribosomebindingsite elements on translational efficiency in Bacillus subtilis and Escherichia coli in vivo". In: *Molecular microbiology* 6.9 (1992), pp. 1105–1114.

[75] George Von Dassow et al. “The segment polarity network is a robust developmental module”. In: *Nature* 406.6792 (2000), pp. 188–192.

[76] Siyuan Wang, Yuping Zhang, and Qi Ouyang. “Stochastic model of coliphage lambda regulatory network”. In: *Physical Review E* 73.4 (2006), p. 41922.

[77] Shuai Weng et al. “Saccharomyces Genome Database (SGD) provides biochemical and structural information for budding yeast proteins”. In: *Nucleic acids research* 31.1 (2003), pp. 216–218.

Chapter 6

Design of Languages for Systems and Synthetic Biology to Translate Genetic Designs into Mathematical Models

Abstract

Context We propose to encompass context-dependent genomic information and mathematical models in a logical and structured fashion through the use of attribute grammars. Attribute Grammars (AG) provide means to compute a mathematical model –possibly encoding phenotypic traits– from a genetic construct. AGs are a framework for biological Domain Specific Languages (DSLs) that, unlike current formalisms to associate a genotype to a phenotype (databases, ontologies), can confer predictive powers. Hence, the language's genetic constructs can be studied *in silico*, an approach which is increasingly essential as synthetic biology becomes more complex. It also makes it possible to predict phenotypes of mutants based on their genome sequences.

Methods DNA compilers based on such AGs can analyze a genetic construct and output the corresponding mathematical model according to the language semantics. We propose a methodology for the development of DSLs to design and analyze genetic constructs. Because development of those languages is dependent on current biological understanding of the modeled mechanisms, it can be incrementally refined for the target biological application. We implemented the workflow to design a DSL, generate the DNA compiler, use the language to design genetic constructs, and analyze constructs *in silico* by running time course simulations of the produced mathematical model in GenoCAD –a point-and-click CAD tool

for synthetic biology relying on formal language theory. Using GenoCAD, a Context-Free Grammar (CFG) can be designed using a drag-and-drop interface; for AGs, attributes can also be specified, and a library of preset functions can be used to make declarations for the semantic actions, or to set attribute values. Resulting biological languages are stored in a relational database. Then, we propose to generate the semantic DNA compilers on-the-fly from user specifications to analyze their designs.

Results We illustrated our workflow with three Synthetic Biology Markup Language (SBML) grammars. In the first grammar, rate laws are modeled by Wilson Cowan equations. This language allows users to design and analyze gene regulatory networks with up to three interacting proteins inserted into plasmids, and we reproduced both the genetic toggle switches and the repressilator with the same library of genetic parts. In the second grammar, we showed we can easily change the rate laws and the granularity of design by generating Mass Action equations instead. Then, we design an application-specific language to design AND gates and layer them to make a 3-input AND gate. We also showed that other modeling approaches than Ordinary Differential Equations (ODEs) could be used in the DSL. We wrote an attribute grammar that outputs a boolean logic model of the gene regulatory network in GinML format and reproduced the simplified cI/cro system of the lambda bacteriophage. Finally, we point out that by re-using the work from system biologists, these DSLs can also model natural genomes.

6.1 Introduction

6.1.1 Biological Languages to Express Genetic Constructs

Synthetic DNA molecules are often constructed through the assembly of known pieces of DNA to obtain a particular behavior. The process of designing a new biological molecule goes through three main stages: design, analyze and fabrication [Lux et al. 2012a]. For the complexity of genetic constructs to increase [Purnick and Weiss 2009], it is key to use computers to help with the design and automation of the transition from the design to the analyze stage. Predictive languages are needed for design exploration in the field of synthetic biology.

Formal languages, that can integrate both syntax and semantics, appears to be a promising modeling approach for Genotype-to-Phenotype predictive maps –genotype being the syntax and phenotype being the meaning–.
David Searls was a pioneer in introducing semantics elements to the syntax of DNA molecules; he proposed a simple system to model the production of proteins from a gene [Searls 1988]. In 2007, formal grammars were shown to guide the design of synthetic DNA made out of standard biological parts [Cai et al. 2007]. Synthetic DNA molecules were chosen to demon-

strate the possibility to define design rules as they are syntactically simpler than natural sequences.

A Context-Free Grammar (CFG) can be defined as a tuple $G : G =< N, T, P, S >$. N is the set of Nonterminals, T is the set of terminals, and $V = T \cup N$ is the vocabulary of the languages defined by the grammar. P is the set of production rules of the form $A \rightarrow \alpha$ where A is a Nonterminal and α is a string of $V*$. S is a nonterminal from which rules start to be applied in order to form a word of grammar. Cai and his colleagues [Cai et al. 2007] applied this formal definition to make synthetic biology sentences, and should be understood as follows: the biological parts (name or DNA sequence) are the terminal words of the language. Those parts belong to categories (promoters, for example), which are, in fact, non-terminals. There are also non-terminals that do not represent any part but are used by the rules, as intermediary design steps for clarity. The grammar rules state how these parts may be associated.

The powerful semantic extension of CFG, Attribute Grammars (AG), have then been used to derive the corresponding mathematical models out of the DNA molecules made of parts [Cai et al. 2009]. A tree illustrating the use of attribute grammars to analyze genetic designs can be seen in Figure 6.1. Attribute grammars are a standard tool used to define the syntax of a language and implement the semantics of a language by associating attributes to the vocabulary and semantic actions to the rules [Knuth 1990]. Attributes are divided into inherited attributes, which are attributes that are passed to a rule, as opposed to synthesized attributes, which are returned from a rule; neither the start symbol nor the terminal symbols have inherited attributes. Semantic actions are used to compute the value of the left side attributes as a function of the ones encountered on the right side. In the 2009 we demonstrated that the language represents gene expression mechanisms (DNA and mRNA molecules, and their protein products), including their regulation through protein-promoter interactions, as mass-action equations [Cai et al. 2009]. The example attribute grammar outputted a SBML file for each valid design compiled. It is then possible to evaluate their likely phenotype after time course simulations. By these means, a combinatorial exploration of the designs makes it possible to screen for a particular genotype using an objective function. From this preliminary work, it is possible to outline a generic mapping of Attribute Grammar components and their correspondence in Biology as shown in Table 6.1.

6.1.2 Computer-Assisted Design of Biological Molecules: Towards *in silico* Simulation

Over the past 10 years, several synthetic biology applications have been developed. The field has been in need for a framework to design biological molecules; CAD systems have been developed to help with the management and virtual assembly of genetic parts [Lux et al. 2012b]. More recently, greater interest has been focused on the ability to simulate the designed molecules *in silico*. Genetic programming languages, and, in particular, Domain-

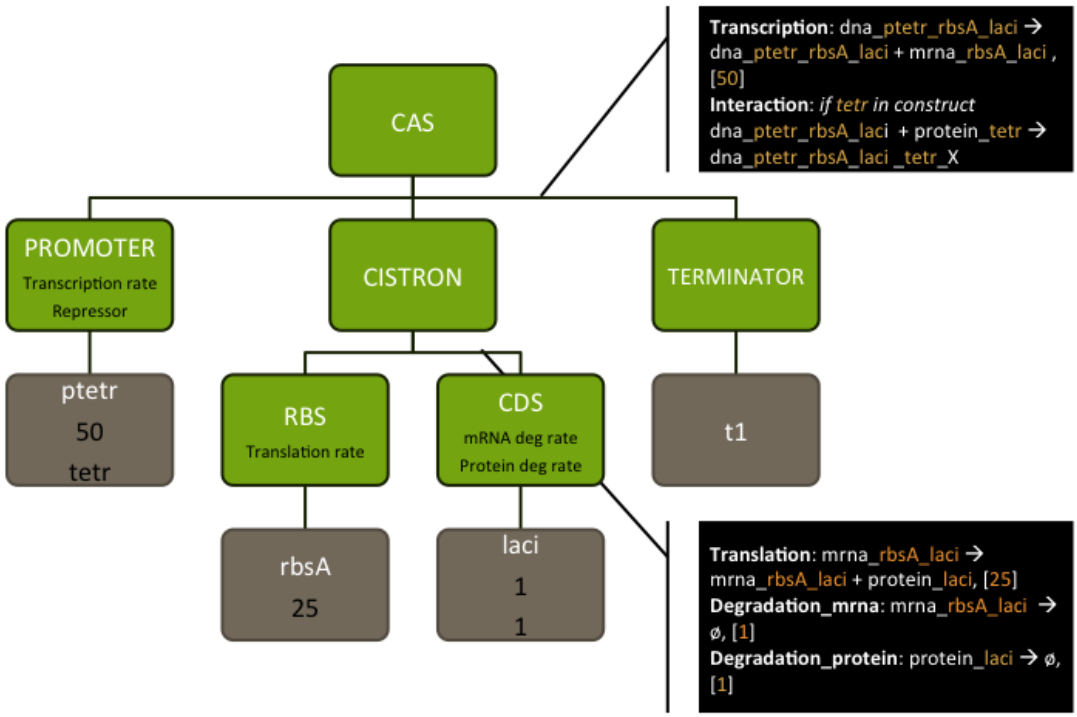


Figure 6.1: Tree based on an Attribute Grammar for Biological Language. This tree is a simple example of a genetic cassette. The start symbols is 'CAS'. It shows how the semantic elements of an attribute grammar can be used to generate the equations that correspond to the example construct. The nonterminal, categories, have attributes. The terminals, genetic parts, have values for the attributes of the category they correspond to. Rules may have semantic actions (the black boxes). The parsing of a construct is made top to bottom, left to right. After the rule is applied, when going back up the tree, the semantic action is computed using the attribute collected below –the synthesized attributes.

Table 6.1: Attribute Grammars in the Context of Synthetic Biology.

Formal definition	Semantic	In the synthetic biology context
N, a finite set of non-terminals	Attributes (synthesized or inherited)	Parts categories
T, a finite set of terminals	Attribute values	Genetic Parts
P, a finite relation from V to $(N \cup T)*$	Semantic actions (context-depend declarations)	Design Rules for layout of DNA molecule
$S \in V$, the start symbol	DNA Compiler code	Entire molecule

Specific Languages (DSL), have been proposed by multiple research groups. These DSLs specify genetic constructs, and resulting equations are derived from each constituent module part-equation; Eugene, GEC and Asmparts are examples of applications that support this approach [Bilitchenko et al. 2011; Pedersen and Phillips 2009; Rodrigo and Carrera 2008]. However, merely concatenating the equations of the constituent parts fails to convey conceptually the expression of a gene working within a system. There is no contextual computational aspect in this approach, which makes it impossible to uncover the non-hard-coded, nonlinear relationships between genotype and phenotype [Peccoud et al. 2004]; still, logical gates can be assembled into larger systems and simulated. Although these DSLs are still useful, they fail to convey all the facets of the engineered DNA system, especially context-sensitivity.

The idea of using formal languages in synthetic biology CAD tools appears to have many advantages. First, the concept of grammar rules is something that any user is familiar with from their experience with spoken languages. The analogy is intuitive.
GenoCAD is a web-based application that uses the generative power of formal language to guide its users through the design of synthetic DNA molecules, much like composing a sentence, ensuring its biological correctness. In order for the end user to use such abstract concepts, the synthetic biology group at Virginia Bioinformatics Institute has been developing GenoCAD since 2007 as a user-friendly software tool geared towards molecular biologists. GenoCAD can be accessed freely online at `www.genocad.org` or installed on a server, as the code is open source. This forward CAD methodology not only allows beginners to navigate within a genetic design workspace, but also allows more advanced biologists to design mutants by applying their own design rules as introduced in Chapter 4.

GenoCAD's grammars describe the functional structure of a biological sequence. Indeed, formal languages have been proven to be suitable for engineering biology. Examples of Context Free Grammars (CFG) have been shown to represent the syntax of genetic constructs based on standardized genetic parts [Cai, Wilson, and Peccoud 2010; Cai et al. 2007; Czar, Cai, and Peccoud 2009]. Additionally, we showed in 2009 that by using attribute grammars we can translate DNA sequences into mathematical models capable of predicting the phenotype they encode and that such grammars help explore the design space before performing any

wet lab work [Cai et al. 2009]. Through the implementation of attribute grammar support, GenoCAD now provides a customizable platform that supports all the steps from DNA design to simulation.

Even though most similar applications allow for personalization by the users, such as the addition or modification of mathematical models, implementing changes requires mastering programming skills to write plugins in the language the software is using; for example, the TinkerCell visual platform allows users to make changes to the predefined processes, but those changes must be implemented using C, C++, Python, Octave or Ruby [Chandran, Bergmann, and Sauro 2009]. Admittedly, within GenoCAD, it would be impossible to define an attribute grammar that could satisfy the needs of every biologist, so biologists should be able to define their own grammars and their own mathematical models.

6.1.3 Creating Tools for the Design of Biological Languages and their Use in CAD Software for Biology

Domain-Specific Languages that derive a mathematical model from a genetic design are narrowing the design space and possibilities of designs in this still-to be standardized field. Many Domain-Specific Languages should be proposed to define a design space depending on the target organism (bacteria, plants), the target application (gene therapy, biobrick assembly) or the institutional use (education, intellectual properties).

In this paper, we present a methodology for defining new biological semantic languages based on a logical framework that is driven by the output format.
Technically, the DSL can then be used, not only to design molecules (by derivation), but also to analyze them (by compilation). We map DNA molecular syntax to mathematical models through these attribute grammars via the creation of a translation tool, referred to in this paper as a DNA compiler. Guided by the attribute grammar, we extract contextual information about the genetic design, then translate it into a mathematical model.
In the methods section we explain the workflow for the definition and usage of a GenoCAD attribute grammar fitted to a particular project, and how we generate a compiler to analyze the designs for a given set of mathematical models.
In the results section, we will illustrate our system's capabilities with a grammar representing Gene Regulatory Networks (GRN), in which a gene's products, or proteins, can be activated or repressed by other components of the network.
This fairly high-level approach allows for the redesign of many synthetic biology constructs [Andrianantoandro et al. 2006; Bashor et al. 2010; Hasty et al. 2001; Purnick and Weiss 2009]. We then show that we can model the mechanisms of gene expression more precisely using our approach than was possible with earlier technologies accounting for cis and trans interactions. This approach can be generalized to produce various mathematical models (ODEs, discrete, etc.), and can handle more or less detailed DNA molecule constructions.

6.2 Methods

6.2.1 Defining a Language for Systems and Synthetic Biology

While it has been shown that Attribute Grammars can support the mapping of genetic information of a DNA molecule to its corresponding mathematical model within a cell compartment in the case of a mass action ODEs of gene interaction sample model [Cai et al. 2009], it is important to understand how to use them to define a wide range of Domain Specific Languages to analyze and predict the behavior of genetic constructs.

The development of a new syntax goes through a top-down process where larger DNA sequences are broken down into categories corresponding to increasingly smaller DNA sequences. The grammar is therefore built by developing an abstraction hierarchy describing a set of DNA sequences generated by the CFG. Each step requires first defining parts categories, then production rules describing how these categories may be combined. As categories and rules are added to the grammar, it is convenient to identify three groups of categories. Rewritable categories are categories that are used at least once on the left side of a production rule; for example, a cassette can be rewritten as a promoter, cistron, and terminator ($CAS \rightarrow PRO, CIS, TER$), so the cassette is a rewritable category. Terminal categories are used only on the right side of rewriting rules; in the previous example, the terminator could be a terminal category as long as there are no rules that rewrite it to another set of subcategories. Finally, orphan categories correspond to categories that have not been used in any production rule. A well-designed grammar should not have any orphan categories or rules. Once parts categories and production rules have been defined, defining T (the set of parts containing DNA segments) completes the CFG. Either adding or importing at least one part for each terminal category achieves this.

The semantics of an attribute grammar is set by semantic actions and attributes. To define a grammar's specific G2P semantic actions, one can use a meta-language. The meta-language is a library of functions that will make the declarations into the target modeling language using the parameters passed as arguments. The library fitted to the output format should be developed ahead of time, although we developed some for the most common system biology modeling languages.

A good practice consists in identifying the components of the model and that each category carries a list of those as attributes (Figure 6.2 provides a logical mapping for generating SBML models). Systematically, semantic actions concatenate each type of attribute lists from the right to the corresponding attribute on the left to pass the semantics up the tree. That is, each declaration is collected in its corresponding list and passed up the tree; declarations are synthesized attributes of the left hand side. The synthesized attributes of terminal symbols are assumed to be externally defined. Additionally, an attribute for the name of the construct is added so that all declarations can be uniquely designated. The name is formed as the concatenation of the categories names below, originally synthesized from the

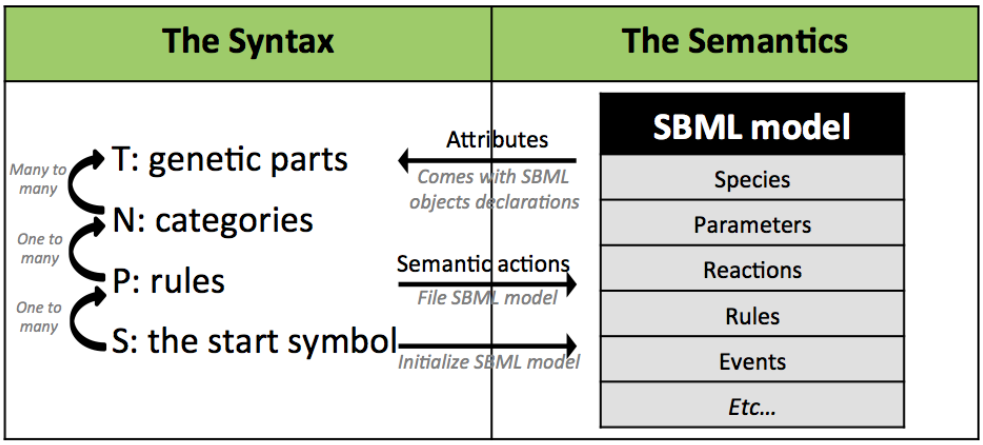


Figure 6.2: Attribute Grammar for Predictive Biological Language. We used the SBML model example to explain the logic of developing an Attribute Grammar. The syntax describes the genetic sequences that can be made by narrowing the possibilities of the spacial structure of genetic elements and the semantics are used to generate the corresponding SBML model by declaring SBML objects that are encountered during the parsing. Each category and genetic part come with attribute names and lists of Species, Parameters, Rules, Events, Trans, etc. Genetic parts set these lists to empty although they may also come with SBML object declarations –for instance, parameter values–. The semantic actions of rules may define new SBML objects, often a new reaction or rule. Each of the attribute lists from the right hand side of the rule get concatenated into the corresponding attribute list of the left side. That way, the semantic elements computing during the parsing are passed up the tree. At the end, the attributes of S that have been synthesized contain the SBML model that represent the design analyzed.

terminals chosen.

A rule's semantic action may be used to declare elements given a context: a reaction, its resulting new biological entity, etc. They provide context-dependency at the local level and can model cis-interactions by declaring elements of the model only for a particular syntax rule.
Trans-equations are handled differently because, at the time when the trans-equation is declared in the semantic actions, the interacting species is unknown and therefore cannot be named. Indeed, it is an indirect reference to elements that may or may not be present in the system; it is only after the entire construct is analyzed that the interacting species may be found to be present. For the syntax of the grammar to convey certain meanings, it may be useful to subcategorize some categories into functional ones. For example, adding a rule such as $Coding\_Sequence \rightarrow TET|LAC|CI$ allows sub-families of functional parts to be created under a generic category of Coding_Sequence, yet still be a syntax unit as any Coding_Sequence could be. Hence, as the syntax called for a sub-functional category supporting the need for keywords to appear in the species name, trans-equations can be declared with these key words and the type of entities (PROT, DNA, etc.). When a trans interaction is possible, a special library (see Appendix C) allows for the declaration of trans reactions and specifies the coupled type and key. At the end, for each trans declaration, species will be searched in order to find functional matches and the corresponding equation will be written if needed.

6.2.1.1 DNA Compilation

A compiler is an application that transforms a computer program written in a programming language, the source code, into an object code written in a target language that can be interpreted or executed. It is possible to apply the concept of compilation to genetic information and develop compilers to translate DNA sequences composed of well-identified genetic parts into a file providing a mathematical representation of the network of molecular interactions encoded in the DNA sequence. The translation of an entire DNA sequence generated by a grammar is the synthesized attributes of the start category. In the end, all declarations have been collected from the design and can be interpreted to create the modeling file. For trans reactions, it may require processing lists to be cross-analyzed according to the participating parts' attributes. This final processing step is part of the DNA compilation process and requires the construction of a DNA compiler for these Attribute Grammars that maps genetic information to a mathematical representation.
We perform our compilation in a single-pass implementation [Koskimies 1984], which results in a smaller, faster, and simpler compiler. Before parsing the construct, we make some initialization calls to open and write the beginning of the target file, including all top-defined elements such as SBML's units. Then the construct is parsed according to the AG specified. Going up the tree, the attributes integrate the semantics calculated from the categories of the construct parsed. When reaching the top, the evaluation of synthesized attributes of

the start symbol may be written to a file from the attributes' lists declaration according to the library. The trans-interaction declarations must be rewritten according to the species declared by looking at the leaves that contain the corresponding key words. The output of this analysis is written to the target file. Finally, functions to finish and close the target file are called.

6.2.2 GenoCAD, Designing Biology as a Language and Designing a Language for Biology

In order to enable the development of new languages applicable to different biological domains, it is necessary to automate the process of deriving a compiler corresponding to the latest iteration of a language under development. The development of an attribute grammar is an iterative process, during which time the attributes of a part category may be changed, or a semantic rule may be modified. In order to ensure quick iterations of the modeling cycle, it is necessary to generate the compiler on-the-fly using the latest modifications to the language, then use the newly generated compiler to translate a number of test cases (genetic constructs or yeast mutants), and, finally, feed the resulting model files into a simulator to compare simulated and actual phenotypes.

GenoCAD's workflow has been divided into a three-step process to integrate simulation and language definition: the parts, the design and the simulation (see also Figure 6.3). In the meantime, GenoCAD has been modified behind the scenes to support attribute grammars in addition to the context free grammars described above. Yet, our user-friendly view of a Computer-Assisted Design (CAD) tool for synthetic biology is bringing us face to face with some computer science issues, namely, an attribute grammar editor and the generation of attribute grammar-based compilers from the database.

GenoCAD does not only allow to design biological molecules based on a given Domain-Specific Language but now, we also provide tools for the design of languages for biology to customize the software.
It started with the implementation of a Context-free Grammar editor (presented in Chapter 4). In order to add new semantic features in GenoCAD, an intuitive attribute grammar editor had to be developed.
Oliveira and his colleagues have been working on visually defining Attribute Grammars and propose a language called Visual Lisa [Oliveira et al. 2009]. Looking at visual AG editor which could be more accessible to GenoCAD's users, this is the only example found. Once a grammar is entered, it is converted to be used with Lisa, a compiler-compiler. However, using Visual Lisa in the context of GenoCAD does not appear to be an option: the visual symbols represent nonterminal, inherited or synthesized attributes, etc. Using these symbols requires an understanding of attribute grammars, which should not be required for GenoCAD. At present in GenoCAD, from a CFG, a user may add synthesized attributes to categories and semantic actions to rules by referring to attributes. Attribute values can be set to parts.

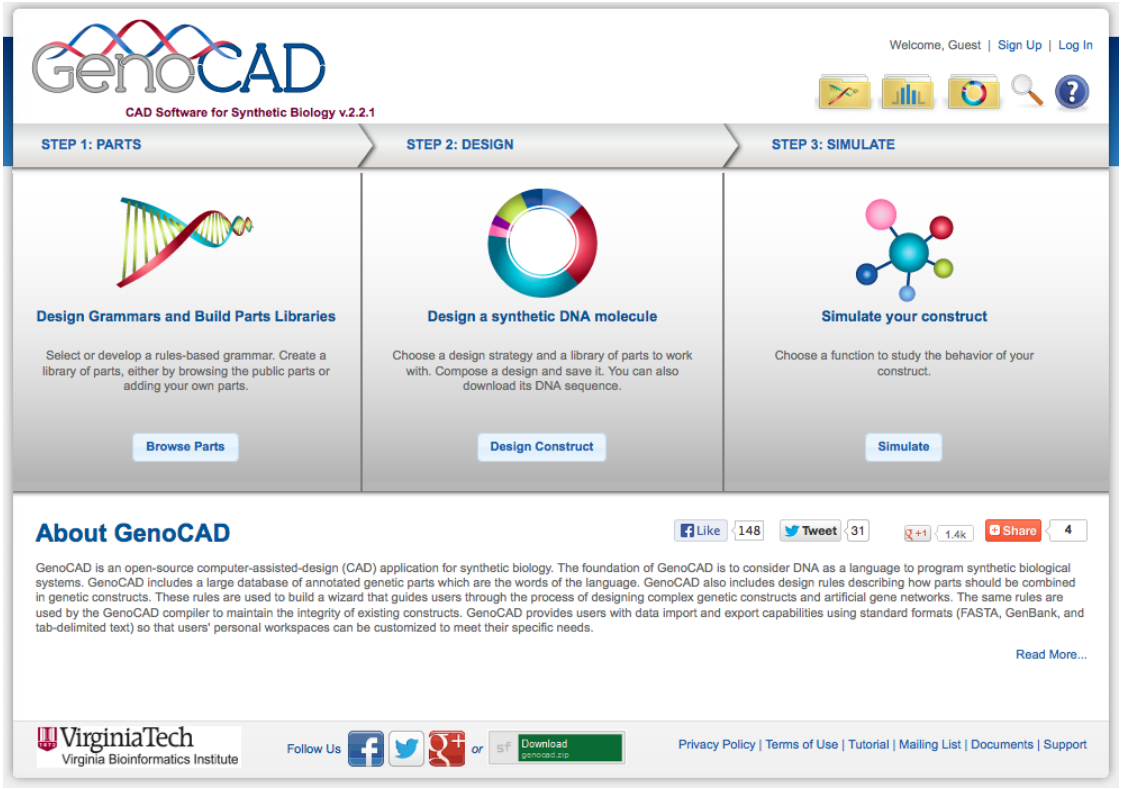


Figure 6.3: GenoCAD Workflow. Using GenoCAD from an end-user perspective is a three-step process. In Step 1, a grammar can be defined to solve a particular problem or organism. Then genetic parts are added and organized as project libraries and grammar rules can be defined. In Step 2, following grammar rules, the user can create a design out of the library's parts. In Step 3, the underlying model can be deduced from the semantic language and the user can run time courses of his design model or download the SBML file (screenshot of the `GenoCAD.org` homepage on 07/12/2013).

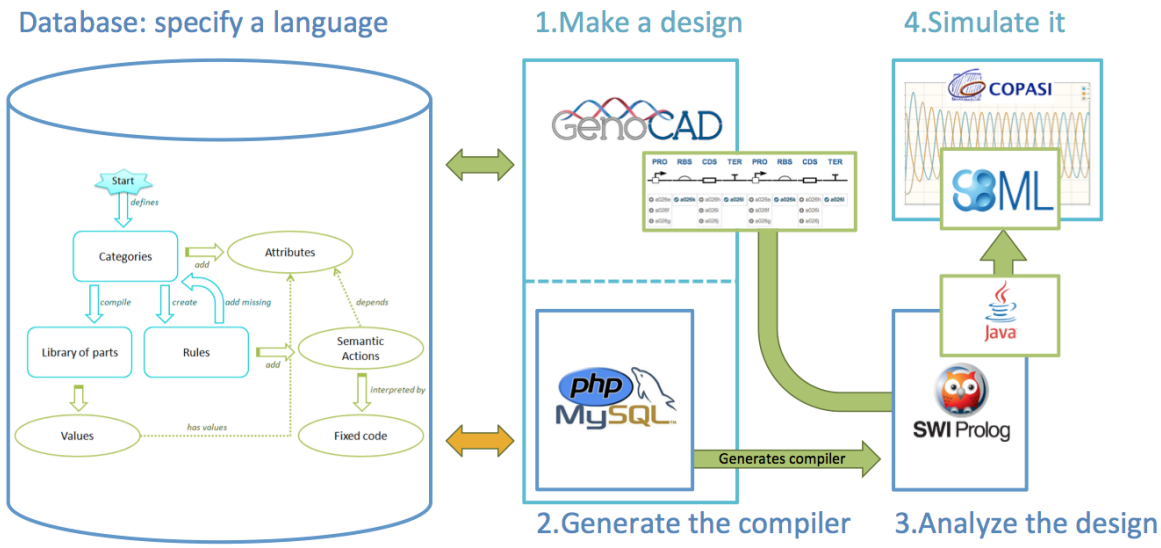


Figure 6.4: Behind the GenoCAD Workflow. Compilation of a design occurs after the on-the-fly generation of a semantic DNA compiler based on the user's attribute grammar as stored in the database. The compiler generated is used to compile the user's design and the SBML target file computed is returned to the user or to Copasi for time course simulation.

6.2.2.1 Data Model

Since the compilation process is designed to be fairly flexible, we store the attribute grammar specifications so that the language can be easily changed. Indeed, new genetic parts with their attribute values declarations can be added to the Vocabulary, new syntax rules can be added to enable alternative constructs possibly mapped to a semantic action, and new categories and their attributes can be added to allow for a more detailed syntax. Since the semantics rely on function calls, a library of function declarations of standard modeling practices can be built and reused across grammars. As a result, semantic DNA compilation can be adapted for each project. A data model to store attribute grammars has been developed and is represented as a database diagram in the Figure C.1 of Appendix C, and an overview can be seen in Figure 6.4; the database serves as the backbone for generating semantic DNA compilers.

6.2.2.2 Compiler Generation for Attribute Grammars

We have described how GenoCAD support the definition of new attribute grammars. On the other end, we explain the need for semantic DNA compilers based on the grammars to obtain a mathematical representation of a design. Therefore, there is a need for these compilers to be generated on-the-fly to encompass the latest modifications.

An AG is not directly turned into a compiler. Hence, Domain Specific Languages for Attribute Grammars have been built; examples of domain-specific languages include Olga [Jourdan et al. 1990a], LISA [Gray and Wu 2005] and Silver [Van Wyk et al. 2008]. Once the input language is specified, platforms can generate the corresponding compiler. FNC-2 [Jourdan et al. 1990b] uses OLGA, and Silver and LISA have their own input language. All fulfill the needs for engineering domain specific languages, as synthetic biology is in need for. However, in GenoCAD, we ultimately aim for the intuitive design of attribute grammar based biological languages with no need to either understand the definition of an attribute grammar nor having programming skills, which would be required by any of these compiler-compiler systems.

6.2.2.3 The Prolog Skeleton of an Attribute Grammar Parser

We chose to implement the compilers in Prolog –we used SWI-Prolog [Wielemaker et al. 2012]–, often used in Computational Linguistics, because of its convenience for knowledge declaration as facts, especially for defining a new attribute grammar, and its rule-like syntax, which is simpler for compiler generation. Indeed, the grammar rules are written using the Definite Clause Grammar (DCG) syntax, which was developed in the spirit of attribute grammars [Pereira and Warren 1980]. Looking first at the context free issue, the fact that prolog supports the definite clause grammar (DFG) syntax, which is extremely close the Backus Normal Form (BNF) of the grammars rules oriented our research in this direction. Finally, based on this paper [Sierra and Fernández-Valmayor 2006], we saw that attributes could be considered as arguments of the vocabulary-"functor", and semantic actions can be specified between brackets after a DCG rule. This allowed us to remain extremely close to the direct attribute grammar specifications, offering prolog's DCG parsing algorithm, top-down depth-first.
Grammar rules can be associated with semantic actions as an instance of Prolog code that is executed after the rule is applied. Syntactically, each rule's semantic actions are written at the end of a rule between braces. For simplicity, they can be rendered as function calls using the accessed attributes' values, and while the function declarations in Prolog must be coded by hand, they are generic enough that a library can be built to reuse them. When a design is submitted for analysis, the compiler is generated against the latest language specifications from the database. We use PHP and MySQL to query the database, process the grammar as previously described, and write the compiler file in SWI-Prolog. The generation of the compiler corresponding to a user-defined attribute grammar is not straightforward. Because of parsing algorithm and Prolog constraints, it is necessary to pre-process the rules before writing the compiler.

The first step in the process of developing the compiler is the removal of orphan rules, which would cause a Prolog error. An orphan rule is a rule that could never be used because there is no situation where it would be called because the left-side category of the rule never appears on the right side of any other rule and that category has no parts; the only exception to this

definition is the start category. Once a rule is identified as an orphan, it is removed from the set of grammar rules; however, this may result in new orphan rules. Therefore, the search for orphan rules continues recursively until no orphans can be found.

The second preprocessing step is the detection of direct left-recursive rules. Left recursive rules are those where the nonterminal on the left side appears first on the right side in the same rule, for example $A \rightarrow A\alpha|\beta$. A top-down parser cannot handle direct left-recursive rules [Frost, Hafiz, and Callaghan 2007]. It is possible to rewrite the direct left recursive rules into indirect ones by replacing them with $A \rightarrow \beta A'$ and $A' \rightarrow \epsilon|\alpha A'$ [Moore 2000]. Unfortunately, there is no solution for rewriting left recursive rules in the case of 'general' attribute grammars [Lohmann, Riedewald, and Stoy 2004] so we only perform this step to check a design validity against a context free grammar. Although at this point we do not check, adding extra steps (for example, $A \rightarrow BC$ and $B \rightarrow A$) would still cause the compiler to fail. In practice, once language designers become aware of this issue, it is not difficult to redesign the language syntax to avoid this problem without any loss of expressivity.

After the preprocessing, the compiler is ready to be written. First, the compiler starts with a function declaration that initiates the parsing from the Start symbol, along with the design as represented by a list of parts. Second, the rules for the associated attribute grammar are fetched and rewritten if necessary. The attributes are placed as arguments of the functor-category. However, we do verify how many times a category appears in the rule in order to suffix the ID of the attributes so that they can be uniquely identified in the semantic action. If a given rule has a semantic action, it is retrieved from the database and placed between brackets at the end of the rule to which it belongs. The semantic actions that indicate that the attributes of the left-hand side are the aggregation of the corresponding attributes of the categories from the right-hand side are generated. The fixed code to analyze the intermediary code, that is, to write the code corresponding to the declaration, which is based on predefined functions, can be fetched as is. Finally, the genetic parts are written, linked to their categories, and declared with their attribute values. The input of the compiler will be the list of parts that makes the design.

6.3 Results

6.3.1 Biological Languages to Compute SBML Models

The meaning of a DNA sequence is a biological function generally represented as a network of molecular interactions regulating the cell physiological processes. The Systems Biology Markup Language (SBML) has evolved into a community standard suitable to represent molecular networks using a variety of mathematical models.
In order to facilitate the development of a broad range of languages suitable to represent the dynamics of various genetic systems, we rely on generic synthesized attributes that remain

independent of the language that manipulates them. Since attributes are used to produce an SBML model [Hucka et al. 2003], it is natural that these attributes correspond to various components of an SBML file, such as a list of species, reactions, rules, events and parameters associated with a particular sequence. Each grammar also contains unit definitions, as well as a compartment definition, associated with the S rule(s).

A generic attribute grammar template to translate genetic sequences into SBML files has been outlined, but each grammar can encode a specific modeling approach. In order to develop a semantic model of DNA sequences from an existing CFG, we rely on a set of chemical equation prototypes corresponding to various types of molecular interactions between reactants, modifiers and products. For instance, the transcription of DNA into an mRNA molecule is commonly represented by a chemical equation expressing that a DNA molecule catalyzes the production of an RNA molecule. This template is represented by replacing the name of specific molecular species in the chemical equation by strings composed of a prefix indicating the type of molecule as in transcription: $DNA \rightarrow DNA + RNA$. This template expresses that the DNA molecule on the left side is the same as the DNA molecule on the right side of the equation: DNA is a modifier and RNA is a product. Similarly the template can be used to model the translation of an mRNA sequence into a protein (PROT). Protein-protein complex formation can be represented by ppi: $PROT_1 + PROT_2 \leftrightarrow COMP$ as the reversible association of two different proteins, the reactants, into a complex, the product. Degradation reactions are expressed using the $\varnothing$ symbol on the right side. Kinetic laws are used to specify the rates of molecular interactions using one or more parameters and the concentration of molecular species; mass action or Hill kinetics are examples of kinetic laws commonly used in systems and synthetic biology. The declaration of kinetic laws corresponds to the list of functions that can be declared in an SBML file. A Kinetic Law must be chosen to model a prototype chemical equation, therefore requiring a set of rates for the model.

The API of the library we developed for SBML models is in the Appendix. It actually helps in generating a java file, using the libSBML library [Bornstein et al. 2008]. There is a direct translation of the java file into an SBML file, and the libSBML API has the advantage of allowing for easy modification to follow the evolution of the SBML language.

6.3.1.1 Wilson Cowan Rate Laws

In Gene Regulatory Networks (GRNs), the nodes are the genes and the edges indicate a regulation, positive or negative, of the synthesis of distant nodes. Particular topologies, or motifs, can have interesting dynamical behavior, such as oscillations or logic gates [Alon 2007; Milo et al. 2002; Tyson and Novák 2010]. Therefore, GRNs can be a basis to the modular approach of synthetic biology based on re-usable components [Alon 2003; Andrianantoandro et al. 2006]. Hence, we developed an Attribute Grammar to design GRNs with up to three network components. Although relatively simple, this grammar allows for de-

signing $3^9 = 19,683$ motifs.

We implemented an attribute grammar to design GRNs. The rules allow for the construction of one to three genetic cassettes, starting with an element InitialConditions InitialConditions may contain the parameters for the initial concentration value of the possible proteins of the systems. Each of the cassettes are made up of a combination of the promoter region to initiate the transcription, with a ribosome-binding site, that initiates the translation (PBS); a coding sequence (CDS), that will be transcribed; and a transcription terminator (TER). The PBS can fall into one of three sub-categories by adding rules $PBS \rightarrow PCI|PLA|PTE$, to get a subset of parts repressible by CI, LAC, and TETR proteins, respectively.

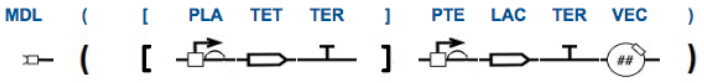
The semantic layer of the example Attribute Grammar specifies how the gene expression is regulated. In a biochemical network, the transcriptional regulation happens through RNA and protein products that have been expressed. These products then regulate the transcription of DNA sequences influencing the rate at which mRNA, the protein pre-products, are transcribed. The mathematical model we use to model this behavior comes from Tyson and Novàk, and relies on Wilson-Cowan equations to express protein levels [Tyson and Novák 2010]. The general form of the Wilson-Cowan nonlinear ODEs is presented in 6.1 where x_i is the level or activity of protein i, $0 \leq x_i \leq 1$, and $F(\sigma W)$ is a sigmoidal function that varies from 0 (when $W \ll \frac{-1}{\sigma}$) to 1 (when $W \gg \frac{1}{\sigma}$). The parameter σ controls the steepness of the sigmoidal function at its inflection point. $W_i = \omega_{i0} + \sum_{j=1}^{N} \omega_{ij} X_j, i = 1, ..., N$ is the net effect on protein i of all proteins in the network. The coefficient ω_{ij} is less than 0 if protein j inhibits the expression of protein i, more than 0 if protein j activates protein i, or equal to 0 if there is no effect of protein j on protein i. In addition, γ indicates the time-scale and ω_{i0} determines the offset for each species ($F \approx 0$ if $\omega_{ij} \ll 0$ and $X_j = 0$ or $F \approx 1$ if $\omega_{ij} \gg 0$ and $X_j = 0$).

$$\frac{dX}{dt} = \gamma * (F(\sigma W) - X), where F(\sigma W) = \frac{1}{1 + e^{-\sigma W}} \tag{6.1}$$

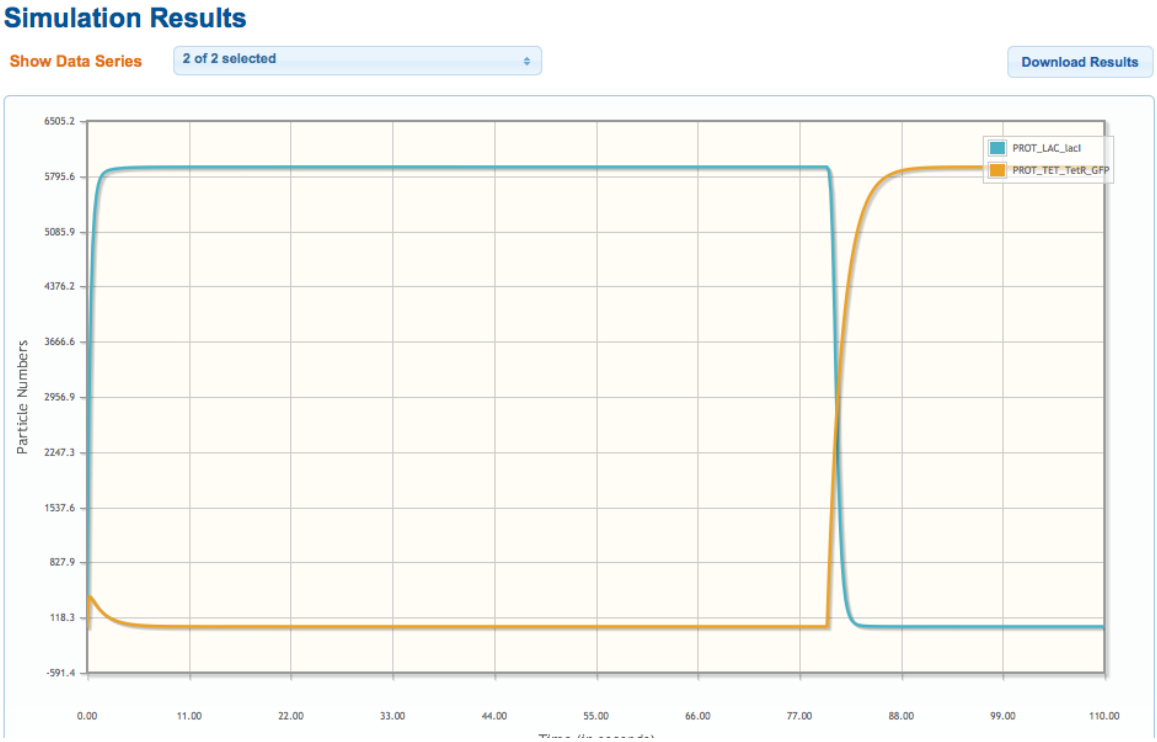
This system has the great advantage of being amenable to all the powerful analytical and simulation tools of nonlinear ODEs; yet, when σ is greater than 1, Equation (6.1) behaves like a piecewise linear ODE.

In the attribute grammar, we declare the part promoter i with σ_i, and ω_{i0} and their values in the attribute list parameters; the gene i has a parameter attribute declaration of γ_i. The category InitialConditions contains all protein initial condition values in its parameters lists. The semantic actions associated with CAS rules store, on one hand, the proteins that will be produced in the system in the species list, and, on the other hand, a Wilson-Cowan equation for protein levels in the reactions list. It depends on W_i, which indicated the regulation. Hence, a rule for W_i is declared in the semantic action linked to the PBS via the subcategory rule, and designates the trans element through the keywords, based on a naming convention ('PROT' is a type of species, 'LAC', 'TET', and 'CI' are keys used in the species names). Upon parsing completion, W_i is computed according to the species being declared after the construct analysis. The grammar is presented in the Table C of Appendix C.

In order to test the example grammar and the generation of the compiler, we examined a genetic toggle switch and an oscillator [Elowitz and Leibler 2000; Gardner, Cantor, and Collins 2000]. We searched for parameters that satisfied the dynamic constraints of the two examples in reusing parts and the rates in two different constructs. We provide a single standard genetic parts libraries with parameter values such that they display the expected behaviors, whether they are assembled as a genetic toggle switch or an oscillator in Figure 6.5 and 6.6. The SBML files and compiled java files along with the scripts used can be downloaded at `http://web.figshare.com/download/file/1118192` along with the compiler generated in GenoCAD and the grammar that can be imported in `GenoCAD.org`.

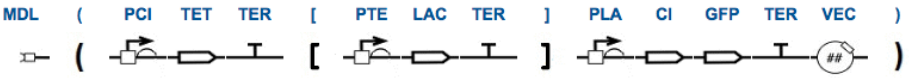


(a) Switch design

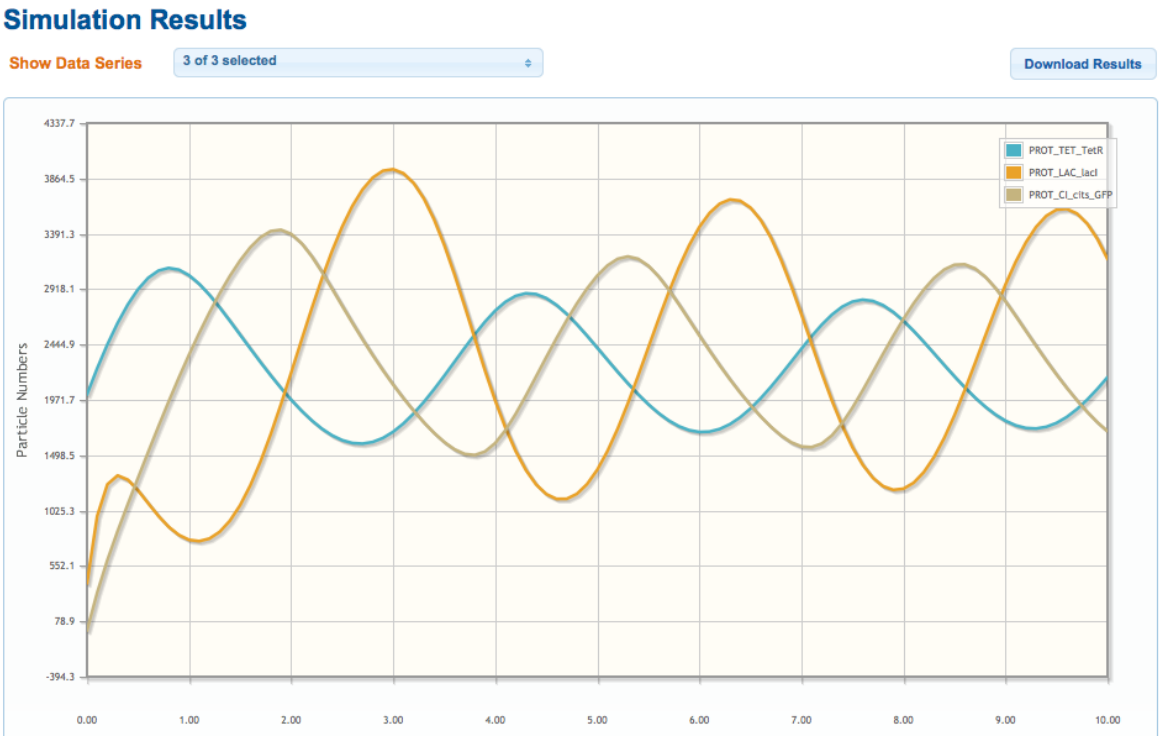


(b) Time Course in hours for the Switch

Figure 6.5: Wilson Cowan Design: Switch. Design of a genetic switch with LacI and TetR proteins using the grammar presented in Table C of Appendix C and its time course simulation using Copasi in GenoCAD from the SBML files generated by our compiler. We used the Wilson Cowan rate law attribute grammar to output the SBML files, and the library of parts we developed to design the examples and to generate the compiler. The MDL parts for switches have events to make the switches change state according to the experiment [Gardner, Cantor, and Collins 2000] –here: at t=20h, aTc is added then removed 20 hours later. IPTG is added at t=80h and removed at t=100h. As expected, we can observe that it takes about 3 hours for the switch to change from one stable steady state to the other.



(a) Oscillator design



(b) Time Course in hours for the Oscillator

Figure 6.6: Wilson Cowan Design: Oscillator. Design of an Oscillator using the grammar presented in Table C of Appendix C and its time course simulation for 10 hours using Copasi in GenoCAD from the SBML files generated by our compiler. We used the Wilson Cowan rate law attribute grammar to output the SBML files, and the library of parts we developed to design the examples and to generate the compiler. We obtained a similar period of oscillations as in the experimental results from Figure 2-c in [Elowitz and Leibler 2000].

6.3.1.2 Mass Action Rate Laws

In a second experiment, we re-use our SBML library to implement a mass-action model of the genetic constructs. Hence, we copied the Wilson Cowan syntax and refined the grammar by separating the PBS into its separate promoter (PRO) and ribosome binding site (RBS) components, and introduced a cistron element (CIS) to the cassette that includes both the RBS and the CDS; thus, the new attribute grammar allows for the distinction between transcription and translation. We chose to model the production and degradation of mRNA

and Proteins with mass action rate laws. Hence, the semantic action corresponding to the cassette declares the transcription in its reactions, and the cistron rules declare the translations. Promoter parts come with a transcription rate in their parameters lists; similarly, RBS parts have a translation rate. Along with the rules $PRO \rightarrow PCI|PLA|PTE$, we declare the possible trans interactions with the key word corresponding to the group of proteins that interacts with it. The grammar is described in Figure C in Appendix C.
This other modeling approach shows how easy it is to model different levels of granularity of the genetic sequence, which can be analyzed immediately thanks to the AG compiler generated. The grammar files for importing in GenoCAD and the compiler generated can be found at `http://web.figshare.com/download/file/1118193`.

6.3.1.3 Design of an Application Specific Language: Layered AND Gates

To increase the complexity of genetic constructs, Voigt and his group proposed to layer AND gates in *E. coli*: the output of the first gate becomes one of the inputs of the following one [Moon et al. 2012]. Such behavior can be modeled *in silico*; we were interested in developing a library and define design principles for layering logic gates as an illustration of the possibility to define an application-specific language using our methodology.
In order to develop the layering language, we first designed a language for the AND gates themselves. The construction of the AND gate consists in designing three plasmids: one for the transcription factor (activator) under the transcriptional control of a regulated promoter, the plasmid with the chaperone under the control of another regulated promoter, and the output plasmid with the core promoter that controls the expression of the rfp reporter.
To generate the mathematical model, we used the SBML API. The semantic actions indicate the production of the chaperone or activator proteins under the control of their promoters. The transfer function of the output promoter uses the TRANS library to indicate that it will be bounded by the complex of the two activators and one chaperone. The Figure 6.7 shows the final syntactic structure of a single AND gate and the main semantics actions that are used to compute the SBML model of the gate.

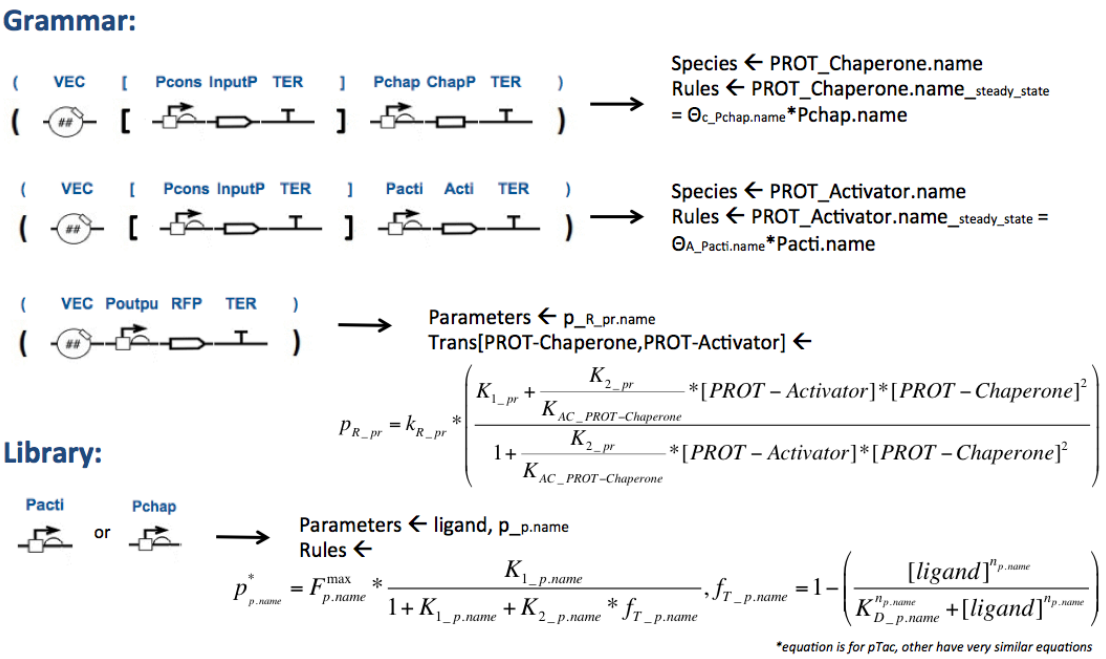


Figure 6.7: AND Gate Structure and its Semantics. The construct layout of an AND gate is made out of three plasmids. The first one carries the Chaperone protein. The semantic action of this plasmid rule declares the protein species using the 'PROT' and 'Chaperone' keywords in the name. The level of expression of the protein depends of the downstream promoter chosen. A rule is declared to compute the protein level at its steady-state. These input promoters come with their own rule to compute the value of 'p_Pchap.name'. The second plasmid contains the Activator coding sequence. Similarly, the keywords 'PROT' and 'Activator' are systematically used in the name of the species that is declared in the semantic action. The 'Pacti_p.name' rule for each of the possible promoters in the library comes in their rules lists. Finally, on the third plasmid, there is the promoter used for the output of the gate. Its transfer-function is given by the trans rule 'pr_poutput.name'. The trans API is used to look for the PROT-Chaperone and PROT-Activator that are actually present in the final species list. The equation of the 'pr_poutput.name' rule is computed according to the matches.

We designed and analyzed both the *Salmonella* and *Shigella* parts gates. In Figure 6.8, we showed the AND gate behavior by computing the transfer function values of the output promoter for different values of the inputs.

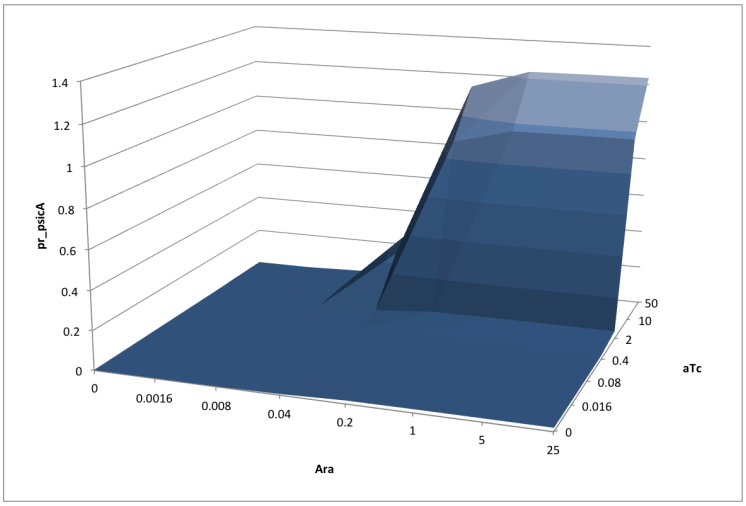


Figure 6.8: Computed Transfer Function of psicA in the Design of the AND Gate with the *Salmonella* Parts Library. We designed an AND gate with the AND gate language described previously and the *Salmonella* parts. We obtained the SBML file from GenoCAD thanks to the generated compiler. We imported it in Copasi to calculate pr_psicA for various values of Ara (0, 0.0016, 0.008, 0.04, 0.2, 1, 5, 25 nM) and aTc (0, 0.016, 0.08, 0.4, 2, 10, 50 ng.ml-1) taken from the experiment (Figure 3 in [Moon et al. 2012]). We plotted them to show the transfer function of the AND gate designed.

In a second time, we added rules to layer the AND gates. The output promoter of the first AND gate will be used as the promoter regulating the transcription factor of the second AND gate. We modified the middle plasmid, the second one. Now, it carries an entire AND gate. The proteins making the second AND gate are designated as intermediary chaperone and activator. We used a naming convention to compute the transfer function of the promoters regulated by the trans action of the chaperones and transcription factors. The last plasmid computes the output of the gate. It contains the rfp reporter under the control of a promoter repressed by the chaperone proteins on the first plasmid and activator proteins –which are the output of the AND gate on the second plasmid– see Figure 6.9.

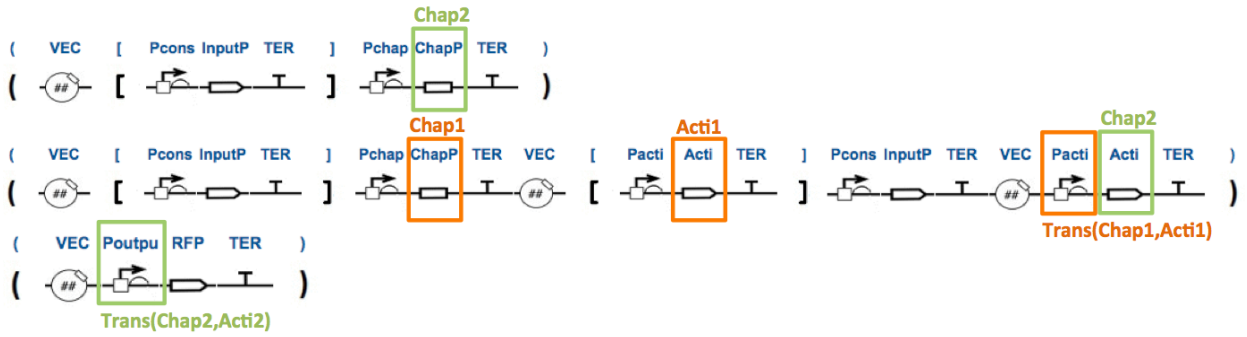


Figure 6.9: Concept to Layer AND Gates. The 3 input AND gate is designed on three different plasmids. The semantic actions that compute the values of the promoters repressed by the transcription factors and chaperones uses different keywords to impose the layered AND gate layout.

We designed a 3-input AND gate using the parts proposed in the paper [Moon et al. 2012] and obtained the corresponding equations, listed in Figure 6.10, by generating the SBML file directly from GenoCAD interface. We verified its 3 input AND gate behavior by computing the transfer function of the output promoter for the different input combinations 6.11.

$$PROT\_Chaperone\_invF = Oc\_PROT\_Chaperone\_invF * p\_pTet$$

$$PROT\_ChapI\_mxiE = Oc\_PROT\_Chaperone\_mxiE * p\_pTac$$

$$PROT\_ActiI\_ipgC = Oa\_PROT\_Activator\_ipgC * p\_pBAD$$

$$PROT\_Activator\_SicAs = Oa\_PROT\_Activator\_SicAs * pr\_pipaHs$$

$$p\_pTet = Fmax\_pTet * \frac{K1\_pTet}{1 + K1\_pTet + 2 * K2\_pTet * ft\_pTet + K2\_pTet^2 * ft\_pTet^2}$$

$$ft\_pTet = 1 - \frac{aTc^{n\_pTet}}{Kd\_pTet^{n\_pTet} + aTc^{n\_pTet}}$$

$$pr\_pipaHs = kr\_pipaHs *$$
$$\frac{K1\_pipaHs + \frac{K2\_pipaHs}{Kac\_PROT\_ChapI\_mxiE} * PROT\_ChapI\_mxiE^2 * PROT\_ActiI\_ipgC}{1 + K1\_pipaHs + \frac{K2\_pipaHs}{Kac\_PROT\_ChapI\_mxiE} * PROT\_ChapI\_mxiE^2 * PROT\_ActiI\_ipgC}$$

$$p\_pTac = Fmax\_pTac * \frac{K1\_pTac}{1 + K1\_pTac + K2\_pTac * ft\_pTac}$$

$$ft\_pTac = 1 - \frac{iPTG^{n\_pTac}}{Kd\_pTac^{n\_pTac} + iPTG^{n\_pTac}}$$

$$p\_pBAD = Fmax\_pBAD *$$
$$\frac{K1\_pBAD + K2\_pBAD * ftl\_pBAD}{1 + K1\_pBAD + K2\_pBAD * ftl\_pBAD + K3\_pBAD * (1 - ftl\_pBAD)}$$

$$ftl\_pBAD = \frac{Ara^{n\_pBAD}}{Kd\_pBAD^{n\_pBAD} + Ara^{n\_pBAD}}$$

$$pr\_psicA = kr\_psicA *$$
$$\frac{K1\_psicA + \frac{K2\_psicA}{Kac\_PROT\_Chaperone\_invF} * PROT\_Chaperone\_invF^2 * PROT\_Activator\_SicAs}{1 + K1\_psicA + \frac{K2\_psicA}{Kac\_PROT\_Chaperone\_invF} * PROT\_Chaperone\_invF^2 * PROT\_Activator\_SicAs}$$

Figure 6.10: Computed Equations for the Layered AND Gates. After the language for layering AND gates has been defined, we generated the compiler and compiled the list of parts that correspond to the layered AND gates with the *Salmonella* and *Shigella* parts. We obtained a SBML file and used Copasi to display the differential equations. The first AND gate keywords are Chaperone and Activator. They are used to compute the trans-equation for the intermediary output promoter. We used different keywords for the second AND gates chaperone and transcription factor proteins in order to compute the trans-function of the promoter controlling the expression of rfp.

The grammar is described in details in Section C in Appendix C.The compiler and scripts along with the SBML files generated can be downloaded at `http://web.figshare.com/`

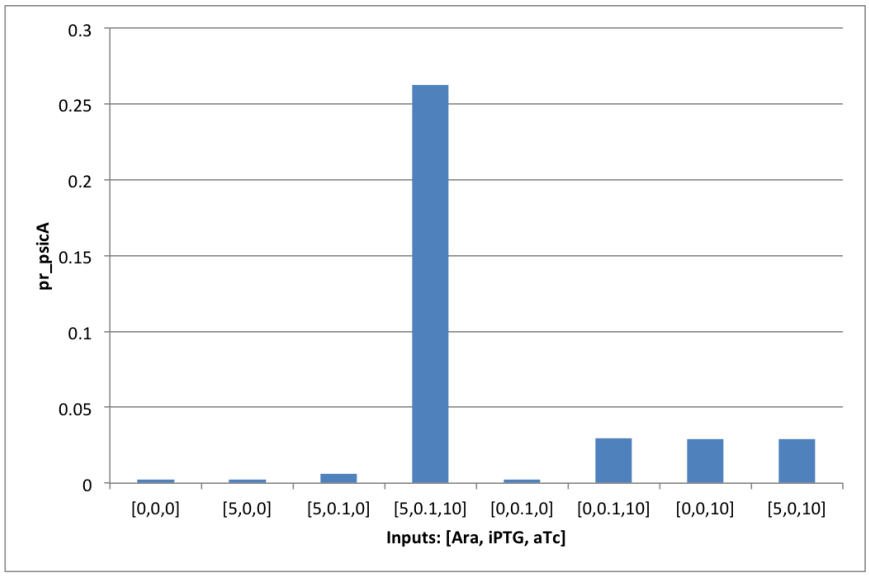


Figure 6.11: Verification of the 3-input AND gate. We used the SBML model generated for the 3-input AND gate design in GenoCAD by our compiler to compute pr_psicA for all possible input values and verified the 3-input AND gate behavior. We see that the value of pr_psciA is high only in the case that the three inputs are added to the system.

`download/file/1149454`.

6.3.2 Discrete Boolean Networks Language

We have proposed Attribute Grammars to create SBML models of synthetic biology constructs based on Wilson-Cowan rate laws or mass action. Different kinetic laws have been used but both fall under the continuous modeling approach. Yet, the design of language for Genotype to Phenotype Mapping can also model constructs with a discrete approach. Logical models are simple and the qualitative features found in the discrete system would be found in a continuous system [Glass 1975]. The discrete boolean modeling approach as presented in this paper [Snoussi 2007] presents interesting features for synthetic biologists because it can give clues about the dynamic features of a system without the need for quantitative parameters.

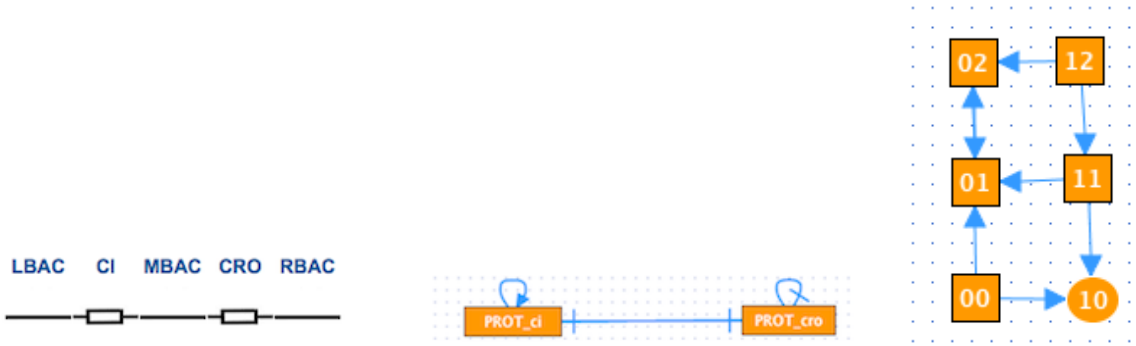
In a discrete gene regulatory network (dGRN), nodes, the proteins of the system designed, are related by arrows, which represent interactions (direct or indirect between the proteins). The sign of the arrow, negative or positive, respectively indicate repression or activation. The concentration of proteins is a set of discrete states, from 0 to the Max Value that is set for each node. A resource for a node is the presence of an activator or the absence of an inhibitor. Each edge also carries a threshold, which indicates the minimum state for the

interacting node to become a resource.

In order to design a language that would model the dynamics of a synthetic DNA molecule as a dGRN, we propose to re-use the previous Wilson Cowan grammar's syntax for simplicity. In this version, each category has attributes for its Name, a list of Nodes and a list of Resources to declare possible interaction edges. In addition, when designing constructs with two or three Cassettes, we add an inherited attribute to the Cassette to contain the high level of expression –MaxVal, set to two or three, respectively.

We output a GinML file ["GINML: Towards a GXL Based Format for Logical Regulatory Graphs and Dynamical Graphs"], which is an XML DSL for regulatory graphs. These types of files can be simulated with GinSIM [Gonzalez et al. 2006]. As previously seen, it required the development of a library to declare the elements that make the file, in particular nodes and edges, included in the annex. We also re-used the TRANS library to declare the edges only if necessary. The essential characteristics of the grammar are described in Section C of Appendix C and the grammars files can be found at `http://web.figshare.com/download/file/1118750`.

We were able to generate the GinML files corresponding to the two or three repressing genes from the previous grammar and open then with GinSIM for simulation. In order to show that dGRN can be more realistic, we added a rule from the Start to represent the lambda bacteriophage genome, and explicitly mapped the gene coding sequence from the CI and CRO proteins. A simplified view of the choice between lyse or becoming lysogenic is controlled by the on or off state of the gene cI which actives its synthesis. Indirectly CI is negatively controlled by CRO, and CRO has a negative autoregulation. The presence of CI has a negative impact on the synthesis of CRO. According to René Thomas' paper [Thieffry and Thomas 1995], it can be modeled with two variables. We introduced a rule $S \rightarrow LBAC, CI, MBAC, CRO, RBAC$ to the dGRN attribute grammar, derived from the genetic map of the virus genome. We added relevant parts and used the GinML library to specify the semantics. The GinML file for the cI/cro sytem can now be generated from the lambda bacteriophage genome constructs with the dGRN attribute grammar and using GinSIM we can simulate our construct as shown in Figure 6.12. With the latest example, we see that once a language is described, it is easy to model different biological molecules using the same libraries, and, most importantly, that our method is scalable to larger systems –here, the virus genome based on systems biology approaches.



(a) Lambda bacteriophage - GenoCAD design

(b) Lambda bacteriophage network generated

(c) GinSim simulation

Figure 6.12: Lambda Bacteriophage cI/cro Design with Discrete Boolean Network. The compilation of the lambda bacteriophage genome provides qualitative models for Lambda Regulation with two-variables cI/cro using the grammar presented in Table C of Appendix C. (a) is the lambda bacteriophage design in GenoCAD, and (b) and (c) are screenshots after the GinML file obtained from compiling the design with the generated compiler was imported in GinSim. The simulation parameters were set according to [Thieffry and Thomas 1995].

6.4 Discussion

Attribute grammars (AG) can formally link phenotypic information, in the form of mathematical models, to genetic information. As the basis of a genetic compilation process that interprets DNA designs into mathematical models, we showed that those AG based compilers can be automatically generated from users' specifications.
Using the methodology presented to design a new biological language based on attribute grammars, it becomes easy to develop Domain Specific Languages to target organisms, applications, institutions, etc., that are needed in synthetic biology CAD software. We illustrated how application specific biological engineering concepts can be integrated at the grammar level and then applied to design genetic constructs and obtain their equations automatically, as demonstrated with the AND gate examples.

In addition to the compiler generation and attribute grammar storage features that have been presented in this paper, GenoCAD would benefit from further development a graphical interface for the semantics of the attribute grammar editor. Indeed, the target users would not need to use Prolog at all; rather, they would "draw" the grammar's syntax and reuse code from a library of functions (rate laws, for instance).

The relationships between genotypes and phenotypes are now often derived from non-linear dynamical systems [Alberch 1991; Pigliucci 2010]. Hence, obtaining the expected phenotype

encoded in a genotype requires computation [Kell 2002]; a formalism to predict a phenotype needs be developed as new type of nonlinear Genotype to Phenotype maps. Such maps are needed in CAD software for engineering biology in order to design more and more complex constructs. The work described in this paper outlines an answer to this missing formalism and helps filling the the gap between genotypic information, as understood by molecular biologists, and phenotypes in the form of mathematical models.

Although this paper proposes simple grammars to illustrate these concepts, the approach can be scaled up to handle larger systems including natural genomes, such as presented in the yeast genome example (in Appendix C.0.1), as well as a large variety of modeling approaches. Not only will this approach speed up the development of biological models by easily enabling the automatic testing of mutant behaviors –each of them could have a target behavior to check– but it will immediately integrate any changes in the model by automatically generating the compiler for each test design. It will also benefit molecular biologists by facilitating the translation of their data to a format acceptable by most modelers.

Supplementary Materials

The generated compiler files are given with scripts to compile the presented constructs. When implemented in GenoCAD, the Attribute Grammar file is also attached and can be uploaded and tested in GenoCAD directly.
The API of the SBML and GinML libraries developed along with some keynotes about our use of Swi-Prolog can be found in Section C.0.1 of Appendix C.

Acknowledgements

Kathy Chen, John Tyson and Neil Adames for their inputs on the Cell Cycle grammar.
François Képès for inviting LA to the thematic school in which I attended a modeling workshop by Gilles Bernot, the inspiration for studying discrete modeling.

Funding

This work was supported by the National Science Foundation [Grant EF-0850100 to JP].

Authors

Laura Adam, Matthew W. Lux, Mandy L. Wilson, Tian Hong, Jean Peccoud

Affiliations:

LA, MWL, MLW, JP: Virginia Bioinformatics Institute, Virginia Tech, Blacksburg VA 24061 (USA)
TH: Department of Biological Sciences, Virginia Tech, Blacksburg VA 24061 (USA) MLW, LA, JP: Virginia Bioinformatics Institute, Virginia Tech, Blacksburg VA 24061, USA

References

[1] P Alberch. "From genes to phenotype: dynamical systems and evolvability." In: *Genetica* 84.1 (Jan. 1991), pp. 5–11. ISSN: 0016-6707.

[2] U Alon. "Biological networks: the tinkerer as an engineer." In: *Science (New York, N.Y.)* 301.5641 (Sept. 2003), pp. 1866–7. ISSN: 1095-9203. DOI: 10.1126/science.1089072.

[3] Uri Alon. "Simplicity in biology." In: *Nature* 446.7135 (Mar. 2007), p. 497. ISSN: 1476-4687. DOI: 10.1038/446497a.

[4] Ernesto Andrianantoandro et al. "Synthetic biology : new engineering rules for an emerging discipline". In: *Molecular Systems Biology* (2006), pp. 1–14. DOI: 10.1038/msb4100073.

[5] Caleb J Bashor et al. "Rewiring Cells: Synthetic Biology as a Tool to Interrogate the Organizational Principles of Living Systems." In: *Annual review of biophysics* (Feb. 2010). ISSN: 1936-1238. DOI: 10.1146/annurev.biophys.050708.133652.

[6] Lesia Bilitchenko et al. "Eugene–a domain specific language for specifying and constraining synthetic biological parts, devices, and systems." In: *PloS one* 6.4 (Jan. 2011), e18882. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0018882.

[7] Benjamin J Bornstein et al. "LibSBML: an API library for SBML." In: *Bioinformatics (Oxford, England)* 24.6 (Mar. 2008), pp. 880–1. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btn051.

[8] Yizhi Cai, Mandy L. Wilson, and Jean Peccoud. "GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs." In: *Nucleic acids research* 38.8 (May 2010), pp. 2637–44. ISSN: 1362-4962. DOI: 10.1093/nar/gkq086.

[9] Yizhi Cai et al. "A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts." In: *Bioinformatics (Oxford, England)* 23.20 (Oct. 2007), pp. 2760–7. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btm446.

[10] Yizhi Cai et al. "Modeling structure-function relationships in synthetic DNA sequences using attribute grammars." In: *PLoS computational biology* 5.10 (Oct. 2009), e1000529. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1000529.

[11] Deepak Chandran, Frank T Bergmann, and Herbert M Sauro. "TinkerCell: modular CAD tool for synthetic biology." In: *Journal of biological engineering* 3 (Jan. 2009), p. 19. ISSN: 1754-1611. DOI: 10.1186/1754-1611-3-19.

[12] Claudine Chaouiya, AG González, and Denis Thieffry. "GINML: Towards a GXL Based Format for Logical Regulatory Graphs and Dynamical Graphs". In: *Citeseer* i (), pp. 0–1.

[13] Michael J Czar, Yizhi Cai, and Jean Peccoud. "Writing DNA with GenoCAD." In: *Nucleic acids research* 37.Web Server issue (July 2009), W40–7. ISSN: 1362-4962. DOI: 10.1093/nar/gkp361.

[14] M B Elowitz and S Leibler. "A synthetic oscillatory network of transcriptional regulators." In: *Nature* 403.6767 (Jan. 2000), pp. 335–8. ISSN: 0028-0836. DOI: 10.1038/35002125.

[15] RA Frost, Rahmatullah Hafiz, and PC Callaghan. "Modular and efficient top-down parsing for ambiguous left-recursive grammars". In: *... International Conference on Parsing ...* June (2007), pp. 109–120.

[16] T S Gardner, C R Cantor, and J J Collins. "Construction of a genetic toggle switch in Escherichia coli." In: *Nature* 403.6767 (Jan. 2000), pp. 339–42. ISSN: 0028-0836. DOI: 10.1038/35002131.

[17] Leon Glass. "Combinatorial and topological methods in nonlinear chemical kinetics". In: *The Journal of Chemical Physics* 63.4 (1975), p. 1325. ISSN: 00219606. DOI: 10.1063/1.431518.

[18] a Gonzalez Gonzalez et al. "GINsim: a software suite for the qualitative modelling, simulation and analysis of regulatory networks." In: *Bio Systems* 84.2 (May 2006), pp. 91–100. ISSN: 0303-2647. DOI: 10.1016/j.biosystems.2005.10.003.

[19] J Gray and H Wu. "Automatic generation of language-based tools using the LISA system". In: 152.2 (2005). DOI: 10.1049/ip-sen.

[20] Jeff Hasty et al. "Designer gene networks: Towards fundamental cellular control." In: *Chaos (Woodbury, N.Y.)* 11.1 (Mar. 2001), pp. 207–220. ISSN: 1089-7682. DOI: 10.1063/1.1345702.

[21] M. Hucka et al. "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models". In: *Bioinformatics* 19.4 (Mar. 2003), pp. 524–531. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg015.

[22] M. Jourdan et al. "The OLGA attribute grammar description language: Design, implementation and evaluation". In: *Attribute Grammars and their Applications* (1990), pp. 222–237.

[23] Martin Jourdan et al. "Design, implementation and evaluation of the FNC-2 attribute grammar system". In: *Proceedings of the ACM SIGPLAN 1990 conference on Programming language design and implementation - PLDI '90* (1990), pp. 209–222. DOI: 10.1145/93542.93568.

[24] Douglas B Kell. "Genotype-phenotype mapping: genes as computer programs." In: *Trends in genetics : TIG* 18.11 (Nov. 2002), pp. 555–9. ISSN: 0168-9525.

[25] Donald E Knuth. "The Genesis of Attribute Grammars". In: *Proceedings of the international conference on Attribute grammars and their applications*. Springer, 1990, pp. 1–12. ISBN: 3540531017.

[26] Kai Koskimies. "A specification language for one-pass semantic analysis". In: *ACM SIGPLAN Notices* 19.8 (1984), pp. 179–189.

[27] W Lohmann, G Riedewald, and M Stoy. "Semantics-preserving Migration of Semantic Rules During Left Recursion Removal in Attribute Grammars". In: *Electronic Notes in Theoretical Computer Science* 110 (Dec. 2004), pp. 133–148. ISSN: 15710661. DOI: 10.1016/j.entcs.2004.06.006.

[28] Matthew W Lux et al. "Genetic design automation: engineering fantasy or scientific renewal?" In: *Trends in biotechnology* 30.2 (Feb. 2012), pp. 120–6. ISSN: 1879-3096. DOI: 10.1016/j.tibtech.2011.09.001.

[29] Matthew W Lux et al. "Genetic design automation: engineering fantasy or scientific renewal?" In: *Trends in biotechnology* 30.2 (Feb. 2012), pp. 120–6. ISSN: 1879-3096. DOI: 10.1016/j.tibtech.2011.09.001.

[30] R Milo et al. "Network motifs: simple building blocks of complex networks." In: *Science (New York, N.Y.)* 298.5594 (Oct. 2002), pp. 824–7. ISSN: 1095-9203. DOI: 10.1126/science.298.5594.824.

[31] Tae Seok Moon et al. "Genetic programs constructed from layered logic gates in single cells." In: *Nature* 491.7423 (Nov. 2012), pp. 249–53. ISSN: 1476-4687. DOI: 10.1038/nature11516.

[32] RC Moore. "Removing left recursion from context-free grammars". In: *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. 2000.

[33] Nuno Oliveira et al. "VisualLISA: Visual programming environment for attribute grammars specification". In: *2009 International Multiconference on Computer Science and Information Technology* (Oct. 2009), pp. 691–698. DOI: 10.1109/IMCSIT.2009.5352765.

[34] Jean Peccoud et al. "The selective values of alleles in a molecular network model are context dependent." In: *Genetics* 166.4 (Apr. 2004), pp. 1715–25. ISSN: 0016-6731.

[35] Michael Pedersen and Andrew Phillips. "Towards programming languages for genetic engineering of living cells." In: *Journal of the Royal Society, Interface / the Royal Society* 6 Suppl 4.April (Aug. 2009), S437–50. ISSN: 1742-5662. DOI: 10.1098/rsif.2008.0516.focus.

[36] Fernando C.N. Pereira and David H.D. Warren. "Definite clause grammars for language analysis—A survey of the formalism and a comparison with augmented transition networks". In: *Artificial Intelligence* 13.3 (May 1980), pp. 231–278. ISSN: 00043702. DOI: 10.1016/0004-3702(80)90003-X.

[37] Massimo Pigliucci. "Genotype-phenotype mapping and the end of the 'genes as blueprint' metaphor." In: *Philosophical transactions of the Royal Society of London. Series B, Biological sciences* 365.1540 (Feb. 2010), pp. 557–66. ISSN: 1471-2970. DOI: 10.1098/rstb.2009.0241.

[38] P.E.M. Priscilla E M Purnick and Ron Weiss. "The second wave of synthetic biology: from modules to systems". In: *Nature Reviews Molecular Cell Biology* 10.6 (June 2009), pp. 410–422. ISSN: 1471-0080. DOI: 10.1038/nrm2698.

[39] Guillermo Rodrigo and Æ Javier Carrera. "Asmparts : assembly of biological model parts". In: *Vital And Health Statistics. Series 20 Data From The National Vitalstatistics System Vital Health Stat 20 Data Natl Vital Sta* (2008). DOI: 10.1007/s11693-008-9013-4.

[40] D Searls. "Representing genetic information with formal grammars". In: *Proceedings of the 7th National Conference on Artificial Intelligence* (1988).

[41] JL Sierra and A Fernández-Valmayor. "A prolog framework for the rapid prototyping of language processors with attribute grammars". In: *Electronic Notes in Theoretical ...* 164.2 (Oct. 2006), pp. 19–36. ISSN: 15710661. DOI: 10.1016/j.entcs.2006.10.002.

[42] El Houssine Snoussi. "Qualitative dynamics of piecewise-linear differential equations : a discrete mapping approach Qualitative dynamics of piecewise-linear differential equations : a discrete mapping approach". In: June 2013 (2007), pp. 37–41.

[43] D Thieffry and R Thomas. "Dynamical behaviour of biological regulatory networks—II. Immunity control in bacteriophage lambda". In: *Bulletin of Mathematical Biology* (1995).

[44] John J Tyson and Béla Novák. "Functional motifs in biochemical reaction networks." In: *Annual review of physical chemistry* 61 (Mar. 2010), pp. 219–40. ISSN: 1545-1593. DOI: 10.1146/annurev.physchem.012809.103457.

[45] Eric Van Wyk et al. "Silver: an Extensible Attribute Grammar System". In: *Electronic Notes in Theoretical Computer Science* 203.2 (Apr. 2008), pp. 103–116. ISSN: 15710661. DOI: 10.1016/j.entcs.2008.03.047.

[46] Jan Wielemaker et al. "SWI-Prolog". In: *Theory and Practice of Logic Programming* 12.1-2 (2012), pp. 67–96. ISSN: 1471-0684.

Chapter 7

Strengths and limitations of the federal guidance on synthetic DNA

Published in:

Adam, L., Kozar, M., Letort, G., Mirat, O., Srivastava, A., Stewart, T., Wilson, M. L., et al. (2011). Strengths and limitations of the federal guidance on synthetic DNA. Nature Biotechnology, 29(3), 208–10. doi:10.1038/nbt.1802

To the Editor:

The December issue included a report summarizing the first reactions of the gene synthesis industry to the publication of the US government Screening Framework Guidance for Providers of Synthetic Double-Stranded DNA [Eisenstein 2010]. Some of the questions raised by the federal guidance had already been exposed in your columns [Fischer and Maurer 2010; LaVan and Marmon 2010], but none of these previous comments relied on a bioinformatics analysis of the screening protocol proposed by the US government. Here we present the preliminary results of an implementation of this protocol with the hope of documenting the strengths and limitations of the federal guidance.

This document outlines a minimal DNA sequence screening protocol that providers of gene synthesis [Czar et al. 2009] services are encouraged to follow before fulfilling an order. The objective of the protocol is to identify sequences of concern of any length that are specific to 'select agents or toxins' (SAT) listed on the National Select Agent Registry (`http://www.selectagents.gov/`). It starts by translating the nucleotide sequence ordered by the customers into each of six possible reading frames. Both the nucleotide and amino acid sequences must then be divided into fragments that are individually aligned against GenBank using a local sequence alignment algorithm. Alignment results are interpreted

using the 'best match' criterion, a procedure designed to identify sequences specific to SATs without relying on a curated database of sequences of concern.
Although the federal guidance gives a general method for the automatic identification of potentially dangerous sequences, few instructions are given concerning the exact implementation of the method. Here we describe an interpretation of the method that is amenable to implementation in software (Figure 7.1).

The input DNA sequence to be screened first undergoes a six-frame translation. The resulting six-amino-acid sequences and the two original DNA sequences correspondingto the two strands of the query sequence are then divided into 66 amino acids (aa) and 200-bp fragments, respectively. When the sequence length is not a multiple of 200 bp or 66 aa, a new subsequence is created using the last 200 bp or 66 aa of the sequence. This subsequence overlaps the last subsequence resulting from the initial fragmentation, but it ensures that the entire sequence is screened.
All of these fragments are then analyzed individually to determine if they should be flagged. They are first aligned against GenBank using BLAST [Camacho et al. 2009]. The best matches are extracted among the BLAST results by selecting the alignments with the highest percent identity over the entire 200-bp fragment (query coverage of 100%). To determine if a best match corresponds to a SAT, the information in the GenBank reference page is cross-referenced with a keyword list. For toxins, keywords include alternative names of the toxin, the names of enzymes that are associated with the production and function of the toxin, and the names of organisms that uniquely produce the toxin. For organisms and viruses, keywords include alternative species names, the names of diseases associated with the entries and any toxins or pathogenic agents uniquely produced by the entry. Two keyword lists were developed. The restricted keyword list has 86 records, whereas the extended keyword list has 340 keywords. If every best match is to a SAT, then the fragment is considered a hit.
A sequence can be fragmented such that a 200-bp region of SAT could unequally straddle two contiguous fragments. To alleviate this issue, the algorithm creates a new 200-bp (of 66 aa) fragment when it detects the presence of an alignment to a SAT longer than 100 bp or 33 aa on either extremity of the subsequence. This new subsequence is composed of the SAT region from the initial fragment and a region from the appropriate adjacent fragment of a length such that the sum of both regions is equal to 200 bp or 66 aa. Every new extended subsequence is compared with GenBank to identify its best matches, as previously described. This thorough analysis is fairly computationally expensive because screening a 1-kb sequence requires at least 40 sequence alignments (two DNA and six protein alignments for each 200-bp fragment). Sequences of several kilobases can be analyzed in a few minutes on a dedicated server or high-end workstation, which should be compatible with the operational constraints of the gene synthesis industry.
The draft guidance published in 2009 focused exclusively on sequences longer than 200 bp, but the final version has removed this exclusion. This decision is unfortunate. Screening short sequences creates all sorts of bioinformatics complications that can affect the quality of the results. The best-match method has been designed to screen long sequences and

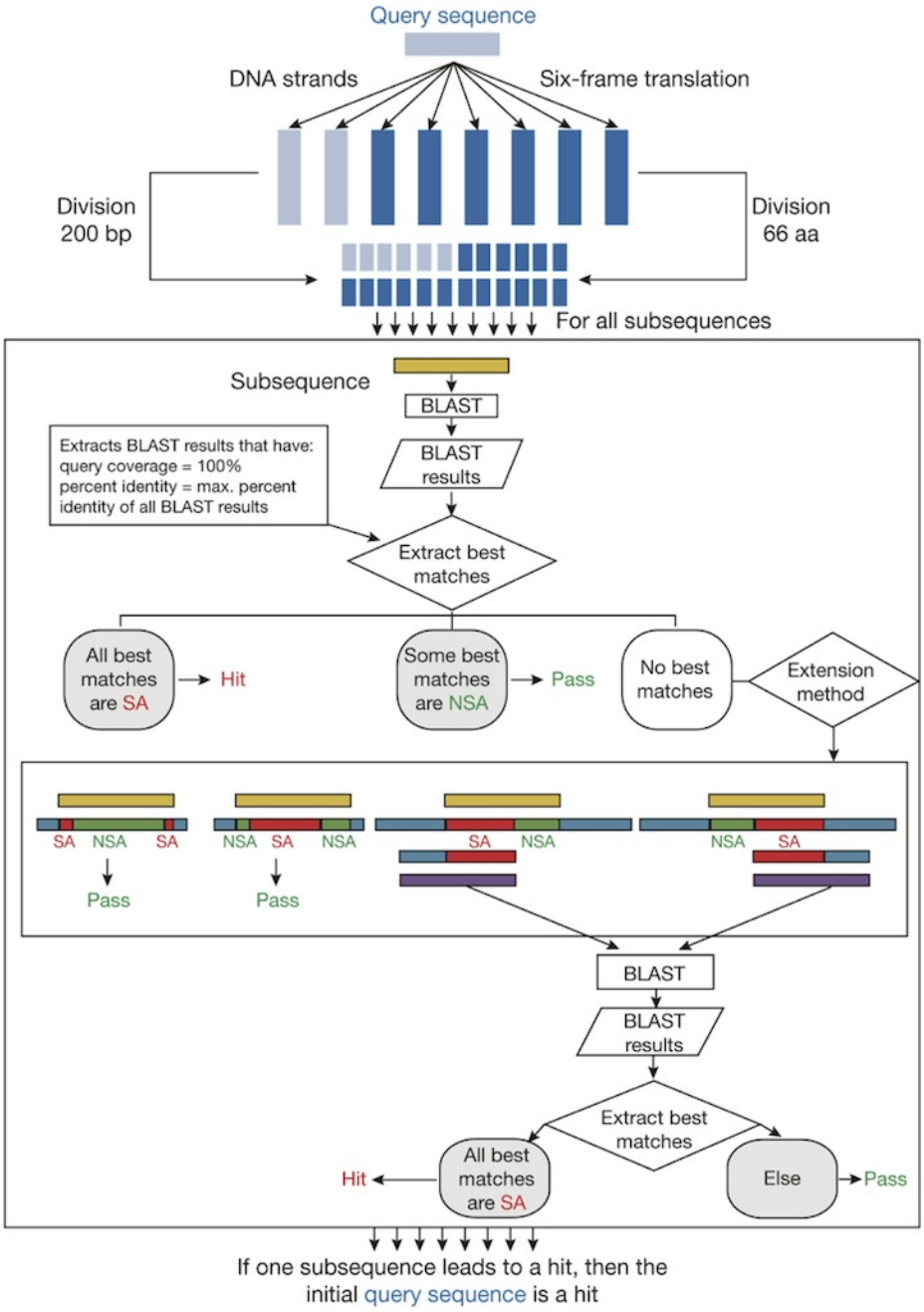


Figure 7.1: Sequence screening algorithm. The query sequence first undergoes a six frame translation, then the amino acid sequences and nucleotide sequences are fragmented into the appropriate size. The subsequences are then aligned using BLAST against GenBank and the nature of the best matches is determined. If there is no best match but there are sequences of concern with query coverage $> 50\%$, then the alignment extension occurs. The algorithm is repeated on the extended sequences to determine whether original query sequence is a hit to a SAT.

is not suitable for screening short sequences. Furthermore, by removing the 200-bp limit, the guidance is somewhat inconsistent. Short sequences are more likely to be ordered as oligonucleotides than double-stranded DNA, but screening oligonucleotide orders is outside the scope of the guidance. For all these reasons, we decided to keep the 200-bp restriction in our implementation of the guidance.

To evaluate the performance of this protocol, we developed a test suite of sequences annotated as either SAT (75 sequences) or non-SAT (100 sequences) after manually reviewing alignment results for each sequence. The accuracy of the screen can be estimated by comparing the screen output with the test sequence annotations. Not surprisingly, the performance of the screening protocol depends on the content of the keyword database. The number of false negatives, sequences of concern that are undetected, is minimized when using the extended keyword list (25 false negatives with the limited keyword list versus 1 false negative with the extended keyword list). Because the outcome of the screen is so dependent on the keywords used to analyze alignment results, it would be useful to develop a standardized list of keywords acceptable to all constituencies. Beyond its application in this particular context, these keyword lists are a prerequisite to the development of a sequence-based classification system of SATs [Wadman 2009].

Moreover, we screened the GenoCAD [Cai, Wilson, and Peccoud 2010; Czar, Cai, and Peccoud 2009] parts database. This data set includes 1,258 sequences longer than 200 bp that mimic the order books of gene synthesis companies. The screen returned 32 hits (2.54%). For most hits, the human review did not uncover any significant relation to SATs beyond some local homology between one of many fragments and a SAT sequence. Even so, we found one GenoCAD part closely related to the YopH protein from Yersinia pestis (gi|14488772). This protocol is extremely effective at detecting sequences of concern embedded into larger sequences because each 200-bp fragment is analyzed individually. The six-frame translation also ensures that redesigned sequences which take advantage of the degeneracy of the genetic code are easily detected by the protocol. However, it proved difficult to design test sequences by introducing mutations in SAT sequences found in GenBank as there is no simple way to determine if such sequences should be detected or not as the biological activity of these sequences is unknown. It would therefore be useful to develop large and realistic training sets that could be used to assess the performance of software implementations of the guidelines recommended by the government.

Before the publication of the federal guidelines, the International Association−Synthetic Biology (IASB; Heidelberg, Germany) published "Code of conduct for best practices in gene synthesis" and the International Gene Synthesis Consortium (IGSC; San Francisco) released their "Harmonized screening protocol." Several important differences between the protocols can be confusing to the public and the gene synthesis industry [Fischer and Maurer 2010]. Table 7.1 shows that the industry is advocating a global analysis of the sequence, leaving the responsibility of interpreting the results to a human operator.

The federal protocol advocates a more granular approach that requires breaking down sequences into smaller fragments analyzed individually. This high-resolution screen can detect local features of a sequence that may be undetected if the sequence is analyzed globally in

Table 7.1: Comparison of sequence screening protocols.

Recommendation	IASB	IGSC	US
Fragment double-stranded DNA sequence	No	No	200bp
Screen six-frame translation of DNA sequence	No	Yes	Yes
Screen against curated sequence database	No	Yes	Optional
Defined criteria to identify sequence as a hit	No	No	Best match
Requires human element in screening procedure	Yes	Yes	No

one pass. Since it is not practical to manually review the results of all the sequence alignments performed by the federal protocol, the federal document provides objective criteria to identify what should be further investigated. This automatic classification of sequences of concern is both a strength and a weakness. On the one hand, it makes it possible to objectively assess the performance of the screen, something that is not possible when the results of sequence alignment are interpreted by human operators. On the other hand, the intrinsic limitations of the best-match method may overlook patterns that human operators would detect. Furthermore, determined individuals could take actions before placing an order to ensure that their order does not raise a red flag. In its defense, the government standard has always been described as a bare minimum that does not prevent the use of complementary approaches such as the ones proposed by the industry. In the long term, the security of gene synthesis may not lie as much in standards as in the availability of biosecurity software applications inspired by computer security solutions. Such biosecurity tools would rely on rapidly evolving models of biosecurity threats to provide human operators with the information they need to quickly and efficiently screen all synthetic DNA sequences at the different steps of the design and fabrication process. The wide adoption of such tools would be objective evidence that the community is developing a culture of responsibility, which is unanimously regarded as the best protection against this new biological threat [Bennett et al. 2009; LaVan and Marmon 2010].

Acknowledgments

This work was supported by National Science Foundation Award no. 1060776 to Virginia Tech. L.A. was supported by a graduate fellowship from the Science Applications International Corporation (SAIC). M.K., A.S., O.M., G.L. and T.S. were supported by undergraduate research fellowships from the MITRE Corporation.

Competing Financial Interests

The authors declare no competing financial interests.

References

[1] Gaymon Bennett et al. "From synthetic biology to biohacking: are we prepared?" In: *Nature biotechnology* 27.12 (Dec. 2009), pp. 1109–11. ISSN: 1546-1696. DOI: 10.1038/nbt1209-1109.

[2] Yizhi Cai, Mandy L. Wilson, and Jean Peccoud. "GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs." In: *Nucleic acids research* 38.8 (May 2010), pp. 2637–44. ISSN: 1362-4962. DOI: 10.1093/nar/gkq086.

[3] Christiam Camacho et al. "BLAST+: architecture and applications." In: *BMC bioinformatics* 10 (Jan. 2009), p. 421. ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-421.

[4] Michael J Czar, Yizhi Cai, and Jean Peccoud. "Writing DNA with GenoCAD." In: *Nucleic acids research* 37.Web Server issue (July 2009), W40–7. ISSN: 1362-4962. DOI: 10.1093/nar/gkp361.

[5] Michael J Czar et al. "Gene synthesis demystified." In: *Trends in biotechnology* 27.2 (Feb. 2009), pp. 63–72. ISSN: 0167-7799. DOI: 10.1016/j.tibtech.2008.10.007.

[6] Michael Eisenstein. "Synthetic DNA firms embrace hazardous agents guidance but remain wary of automated 'best-match'." In: *Nature biotechnology* 28.12 (Dec. 2010), pp. 1225–6. ISSN: 1546-1696. DOI: 10.1038/nbt1210-1225.

[7] Markus Fischer and Stephen M Maurer. "Harmonizing biosecurity oversight for gene synthesis." In: *Nature biotechnology* 28.1 (Jan. 2010), pp. 20–2. ISSN: 1546-1696. DOI: 10.1038/nbt0110-20.

[8] David A LaVan and Louis M Marmon. "Safe and effective synthetic biology." In: *Nature biotechnology* 28.10 (Oct. 2010), pp. 1010–2. ISSN: 1546-1696. DOI: 10.1038/nbt1010-1010.

[9] Meredith Wadman. "US drafts guidelines to screen genes". In: *Nature* December (2009), pp. 2–5. ISSN: 1744-7933. DOI: 10.1038/news.2009.1117.

Chapter 8

Conclusion

This thesis aims to be an interdisciplinary project that uses formal language theory in a biological context. In particular, we provided examples of applications in the field of synthetic biology. We also hope to demonstrate the importance of considering the implications of our research; here, the biosecurity issues raised by DNA synthesis.

Use of Formal Languages in Biology for Making Phenotypic Predictions

The nature of this thesis fits under the several efforts to develop languages to design and simulate biological constructs. Unlike other synthetic biology CAD software, we use formal languages which allow the computation of context dependencies. We showed that attribute grammars can be the template for Genotype to Phenotype maps. Designs based on such languages can be studied *in silico*, after a mathematical representation of the genetic construct has been computed. In order to use this formalism, we proposed a mechanism of DNA compilation that uses Prolog.

We realized that there is a need for users to define their own project-specific language. We worked on an editor, which includes a storage database for Attribute Grammars and the definition of libraries for target modeling language declarations. We illustrated that with SBML and GinML. As a result, there is a need for the generation of attribute grammar-based DNA compilers that take into account the latest language modifications. This entire algorithm that generates SBML files is being integrated into GenoCAD, allowing the users to use this theory with only a minimal understanding of compilation or formal languages.

GenoCAD Improvements

In order to integrate these new features into GenoCAD, user interviews were conducted in 2009 as a starting point. Some of the interviewees were not familiar with the software but were willing to learn through workshops since the idea of using grammars to develop DNA sequences sounded promising to them. Users asked for detailed descriptions in order to choose a grammar for a design and to make it easy to exchange between tools.

1. Improvements to parts management. Search functionalities have been developed, and import/export with the GenBank format is now available. SBOL visual language can be used to design constructs. Therefore, the collaboration and cross applications requested have been covered. User personal spaces have been developed and parts or designs can be saved (and "cloned").

2. Tutorials have been developed and given over the past years allowing for constant user feedback through direct interaction. The tutorials can be found at `http://peccoud.vbi.vt.edu/genocad-training-set-i/`. I gave several workshops to teach people how to use GenoCAD and design their grammars.

3. Redesigning of the workflow and the interface of GenoCAD into a three steps process for clarity, and also in order to integrate the simulation functionalities through the generation of a semantic DNA compiler

   Implementation the generation of attribute grammar based compiler

   Design of a data model to store the attribute grammars

   Finally, the use of Grammars is only possible if users are able to fit them to their needs. Public Grammars remain available and can also be used as templates for design new grammars. A grammar editor has been implemented.

Societal Impacts of Designing DNA Molecules

Since we are helping people to design DNA sequences, we wonder how the growing accessibility of biotechnologies raise a number of emerging security concerns. My research has focused on the development of publicly available tools to facilitate the engineering of genetic constructs. As these tools advance, organisms could be designed to perform increasingly specific purposes by people with decreasing expertise. Biothreats purposefully targeted for use against humans, animals, or plants could have devastating consequences. How can we ensure that our tools do not enable the design of organisms for malicious intent?
With such questions in mind, when the US federal guidance on screening DNA synthesis orders was drafted, we decided to address the inherent dangers of my research field. DNA synthesis technologies are widely used and extremely useful; however, by using the same

technologies, ill-intentioned individuals could obtain genetic material from dangerous organisms, challenging the existing biocontainment measures. Hence, gene synthesis is considered a dual-use technology. We implemented the guidance's algorithm that can search a DNA sequence for subsequences (over 200bp) that are uniquely related to Select Agents and Toxins sequences and characterize its efficiency in mitigating the risks.

Direction of Work

There are a few directions foreseen to build upon this work.
On the one hand, the design of biological languages can ease the usage of ontologies and databases for information extraction; grammar inference approaches could also be used. On the other hand, the presented languages can also be further developed. In particular, for the yeast cell cycle, it would be interesting to model mutants, and try to make predictions to be validated in the lab.
GenoCAD would benefit from a better analysis of a natural DNA sequence when comparing them to the database of parts. Integrating sequence alignment with a margin of error as part of the lexical analysis would allow sequenced data to be uploaded and analyzed for an explanation about their phenotype thanks to the Domain Specific Languages.
The systematic check of design molecules should be implemented in GenoCAD. It would be interesting to implement the algorithm using exact sequence alignments and compare performances with the current implementation that uses Blast. The use of grammars could be integrated in such an algorithm to take into account the complicated construction/deconstruction tricks that could be used in design molecules.

Appendix A

Appendix for Chapter 2

The Sequence Ontology is used in SBOL to specify the types of DnaComponents. Below are some samples of these types which are commonly used in synthetic biology designs. For example, the value of DC.type MUST be a valid SO URI such as those listed below. Additional Sequence Ontology types can be found at the website (`http://www.sequenceontology.org/`) the namespace used for SO URIs is "`http://purl.obolibrary.org/obo/`" these URIs are maintained as Persistent Uniform Resource Locators (PURLs) by the OBO Foundry (`http://obolibrary.org/about.shtml`) and can be used to retrieve additional information about the term.

Promoter	A regulatory_region composed of the TSS(s) and binding sites for TF_complexes of the basal transcription machinery. SO URI: http://purl.obolibrary.org/obo/SO_0000167 SO Name: promoter
Operator	A regulatory element of an operon to which activators or repressors bind, thereby effecting translation of genes in that operon. SO URI: http://purl.obolibrary.org/obo/SO_0000057 SO Name: operator
CDS	A contiguous sequence which begins with, and includes, a start codon, and ends with, and includes, a stop codon. SO URI: http://purl.obolibrary.org/obo/SO_0000316 SO Name: cds
5' UTR	A region at the 5' end of a mature transcript (preceding the initiation codon) that is not translated into a protein. SO URI: http://purl.obolibrary.org/obo/SO_0000204 SO Name: five_prime_utr
Terminator	The sequence of DNA located either at the end of the transcript that causes RNA polymerase to terminate transcription. SO URI: http://purl.obolibrary.org/obo/SO_0000141 SO Name: terminator
Insulator	A transcriptional cis regulatory region that, when located between a CM and a gene's promoter, prevents the CRM from modulating that genes expression. SO URI: http://purl.obolibrary.org/obo/SO_0000627 SO Name: insulator
Origin of Replication	The origin of replication; starting site for duplication of a nucleic acid molecule to give two identical copies. SO URI: http://purl.obolibrary.org/obo/SO_0000296 SO Name: ori
Primer Binding Site	Non-covalent primer binding site for initiation of replication, transcription, or reverse transcription. SO URI: http://purl.obolibrary.org/obo/SO_0005850 SO Name: primer_binding_site
Restriction Enzyme Recognition Site	Represents a region of a DNA molecule which is a nucleotide region (usually a palindrome) that is recognized by a restriction enzyme. SO URI: http://purl.obolibrary.org/obo/SO_0001687 SO Name: restriction_enzyme_recognition_site

Appendix B

Appendix for Chapter 5

No.	Semantic Actions	Dependence
9	promoter1.name = [pro_u]	N/A
10	rbs1.name = [rbsA]	N/A
11	gene1.name = [v]	N/A
12	terminator1.name = [t1]	N/A
13	promoter2.name = [pro_v]	N/A
14	rbs2.name = [rbsB]	N/A
15	gene2.name = [u]	N/A
16	terminator2.name = [t1]	N/A
4	cistron1.transcript = rbs1.name + gene1.name = [rbsA_v] cistron1.equation_list = *translation*(rbsA, v)	10, 11
8	cistron2.transcript = rbs2.name + gene2.name = [rbsB_u] cistron2.equation_list = *translation*(rbsB, u)	14, 15
7	restConstruct2.equation_list = [ ]	N/A
2	cassette1.promoter_list = [promoter.name, cistron.transcript] = [pro_u, rbsA_v] cassette1.equation_list = *promoter_protein_interaction*(cassette1.promoter_list, cassette1.protein_list) + *transcription*(promoter, cistron1.transcript) + cistron1.equation_list = *promoter_protein_interaction*([pro_u, rbsA_v], cassette1.protein_list) + *transcription*(pro_u, rbsA_v) + *translation*(rbsA, v)	4, 9, 12
6	cassette2.promoter_list = [promoter.name, cistron.transcript] = [pro_v, rbsB_u] cassette2.equation_list = *promoter_protein_interaction*(cassette2.promoter_list, cassette2.protein_list) + *transcription*(promoter, cistron2.transcript) + cistron2.equation_list = *promoter_protein_interaction*([pro_v, rbsB_u], cassette2.protein_list) + *transcription*(pro_v, rbsB_u) + *translation*(rbsB, u)	8, 13, 16
5	construct2.equation_list = cassette2.equation_list + restConstructs2.equation_list = *translation*(rbsB, u) + *transcription*(pro_v, rbsB_u) + *promoter_protein_interaction*([pro_v, rbsB_u], cassette2.protein_list)	6, 7
3	restConstructs1.equations_list = construct2.equation_list = *translation*(rbsB, u) + *transcription*(pro_v, rbsB_u) + *promoter_protein_interaction*([pro_v, rbsB_u], cassette2.protein_list)	5
1	cassette1.protein_list = constructs1.protein_list = [protein_u, protein_v] cassette2.protein_list = constructs1.protein_list = [protein_u, protein_v] constructs1.equation_list = cassette1.equation_list + restConstructs.equation_list = *transcription*(pro_u, rbsA_v) + *translation*(rbsA, v) + *transcription*(pro_v, rbsB_u) + *translation*(rbsB, u)+ *promoter_protein_interaction*([pro_u, rbsA_v], [protein_u, protein_v]) + *promoter_protein_interaction*([pro_v, rbsB_u], [protein_u, protein_v])	2, 3

Figure B.1: Computation dependence corresponding to the derivation tree in Figure 5.2. The computation starts from the leaves of the tree, and the semantic values computed are transferred to upstream nodes. The computation of each node cannot proceed until all of its sub-trees are computed. For example, the computation of semantic values of (2) is pending until its subtrees (3) and (4) are computed.

Part Name	Part Type	Associated Attribute	Attribute Value
ptrc2	Promoter	promoter.name	ptrc2
		promoter.transcription_rate	25
		promoter.leakiness_rate	.25
		promoter.repressor_list	[[lacI, 4, 0.001, 1], [lacIrc, 4, 0.001, 1]]
pls1con	Promoter	promoter.name	pls1con
		promoter.transcription_rate	50
		promoter.leakiness_rate	0.00833333
		promoter.repressor_list	[[cIts, 2, 0. 1, 1], [cItsrc, 2, 0.1, 1]]
pltet01	Promoter	promoter.name	tetR
		promoter.transcription_rate	10
		promoter.leakiness_rate	0.1
		promoter.repressor_list	[[tetR, 2, 0.1, 1], [tetRrc, 2, 0. 1, 1]]
ptrc2rc	Reverse Promoter	reversePromoter.name	ptrc2rc
		reversePromoter.transcription_rate	25
		reversePromoter.leakiness_rate	.25
		reversePromoter.repressor_list	[[lacI, 4, 0.001, 1], [lacIrc, 4, 0.001, 1]]
pls1conrc	Reverse Promoter	reversePromoter.name	pls1conrc
		reversePromoter.transcription_rate	50
		reversePromoter.leakiness_rate	0.00833333
		reversePromoter.repressor_list	[[cIts, 2, 0. 1, 1], [cItsrc, 2, 0.1, 1]]
pltet01rc	Reverse Promoter	reversePromoter.name	tetRrc
		reversePromoter.transcription_rate	10
		reversePromoter.leakiness_rate	0.1
		reversePromoter.repressor_list	[[tetR, 2, 0.1, 1], [tetRrc, 2, 0. 1, 1]]
rbsA	RBS	rbs.name	rbsA
		rbs.translation_rate	25
rbsB	RBS	rbs.name	rbsB
		rbs.translation_rate	50
rbsC	RBS	rbs.name	rbsC
		rbs.translation_rate	10
rbsD	RBS	rbs.name	rbsD
		rbs.translation_rate	12.5
rbsE	RBS	rbs.name	rbsE
		rbs.translation_rate	6.25
rbsF	RBS	rbs.name	rbsF
		rbs.translation_rate	7
rbsG	RBS	rbs.name	rbsG
		rbs.translation_rate	5
rbsH	RBS	rbs.name	rbsH
		rbs.translation_rate	2
rbsArc	Reverse RBS	reverseRBS.name	rbsA
		reverseRBS.translation_rate	25
rbsBrc	Reverse RBS	reverseRBS.name	rbsB
		reverseRBS.translation_rate	50
rbsCrc	Reverse RBS	reverseRBS.name	rbsC
		reverseRBS.translation_rate	10
rbsDrc	Reverse RBS	reverseRBS.name	rbsD
		reverseRBS.translation_rate	12.5
rbsErc	Reverse RBS	reverseRBS.name	rbsE
		reverseRBS.translation_rate	6.25

rbsFrc	Reverse RBS	reverseRBS.name	rbsF
		reverseRBS.translation_rate	7
rbsGrc	Reverse RBS	reverseRBS.name	rbsG
		reverseRBS.translation_rate	5
rbsHrc	Reverse RBS	reverseRBS.name	rbsH
		reverseRBS.translation_rate	2
lacI	Gene	gene.name	lacI
		gene.mRNA_degradation_rate	1
		gene.protein_degradation_rate	0.1
gfpmut3	Gene	gene.name	gfpmut3
		gene.mRNA_degradation_rate	1
		gene.protein_degradation_rate	0.1
cIts	Gene	gene.name	cIts
		gene.mRNA_degradation_rate	1
		gene.protein_degradation_rate	0.1
tetR	Gene	gene.name	tetR
		gene.mRNA_degradation_rate	1
		gene.protein_degradation_rate	0.1
lacIrc	Reverse Gene	reverseGene.name	lacIrc
		reverseGene.mRNA_degradation_rate	1
		reverseGene.protein_degradation_rate	0.1
gfpmut3rc	Reverse Gene	reverseGene.name	gfpmut3rc
		reverseGene.mRNA_degradation_rate	1
		reverseGene.protein_degradation_rate	0.1
cItsrc	Reverse Gene	reverseGene.name	cItsrc
		reverseGene.mRNA_degradation_rate	1
		reverseGene.protein_degradation_rate	0.1
tetRrc	Reverse Gene	reverseGene.name	tetRrc
		reverseGene.mRNA_degradation_rate	1
		reverseGene.protein_degradation_rate	0.1
b0010	Terminator	none	
b0012	Terminator	none	
b0016	Terminator	none	
b0010rc	Reverse Terminator	none	
b0012rc	Reverse Terminator	none	
b0016rc	Reverse Terminator	none	

Figure B.2: List of parts used in the "exploration of genetic space" section and values of associated attributes.

Appendix C

Appendix for Chapter 6

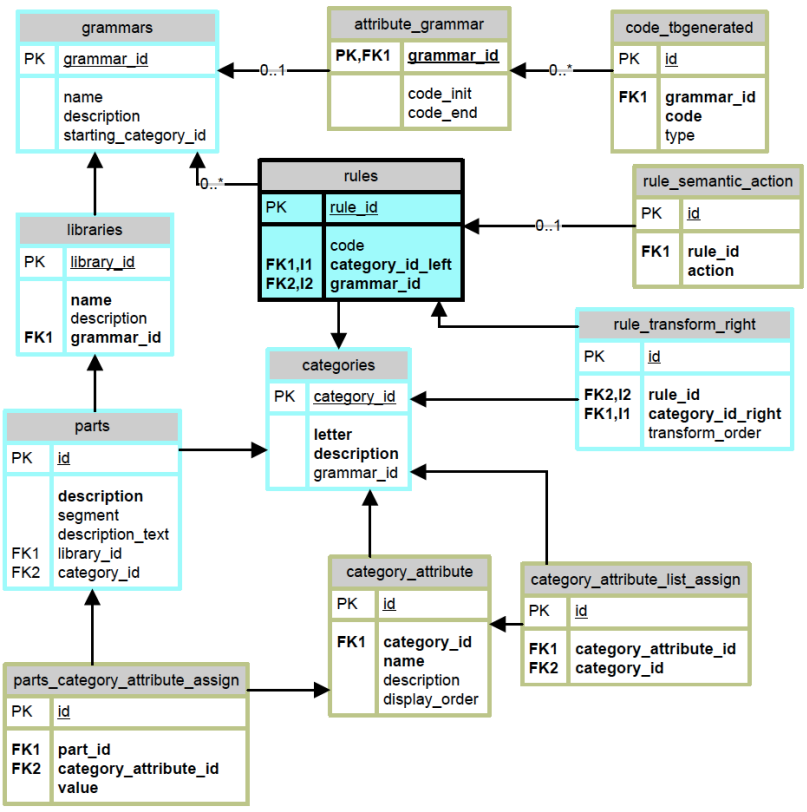


Figure C.1: Simple data model for storing Attribute Grammars.

Wilson-Cowan Attribute Grammar

Rules	**Reactions**
2CAS --> CAS CAS | [CAS] CAS | [CAS CAS]	
CDS --> CI | TET | LAC	
3CAS --> CAS CAS CAS | [CAS CAS CAS] | CAS [CAS] CAS	
CAS --> PBS CDS TER	reaction(wilson_cowan_*<CAS.Construct>*, [], [], [PROT_*<CDS.Construct>*], "g_*<CDS.Construct>**(1E-20/(1+power(ExponentialE,-s__*<PBS.Construct>**W__<PBS.Construct>))-PROT_*<CDS.Construct>*)")
PBS --> PCI	
PBS --> PLA	
PBS --> PTE	
FP --> GFP	
CAS --> PBS CDS FP TER	reaction(wilson_cowan_*<CAS.Construct>*, [], [], [PROT_*<CDS.Construct>*_<FP.Construct>], "g_*<CDS.Construct>**(1E-20/(1+power(ExponentialE,-s__*<PBS.Construct>**W__<PBS.Construct>))-PROT_*<CDS.Construct>*_<FP.Construct>)")
SPL -->) (CAS VEC SPL |)	
S --> MDL (2CAS VEC SPL	
CAS --> PBS FP TER	reaction(wilson_cowan_*<CAS.Construct>*, [], [], [PROT_*<FP.Construct>*], "g_*<FP.Construct>**(1E-20/(1+power(ExponentialE,-s__*<PBS.Construct>**W__<PBS.Construct>))-PROT_*<FP.Construct>*)")

Rules	**Species**
2CAS --> CAS CAS | [CAS] CAS | [CAS CAS]	
CDS --> CI | TET | LAC	
[CAS CAS CAS] | CAS [CAS] CAS	
CAS --> PBS CDS TER	species(PROT_<*CDS.Construct*>, init_<*CDS.Construct*>.getValue())
PBS --> PCI	
PBS --> PLA	
PBS --> PTE	
FP --> GFP	
CAS --> PBS CDS FP TER	species(PROT_<*CDS.Construct*>_<FP.Construct>, 0)
SPL -->) (CAS VEC SPL |	
S --> MDL (2CAS VEC	
CAS --> PBS FP TER	species(PROT_<*FP.Construct*>, init_<*CDS.Construct*>.getValue())

Rules	**Trans**
[CAS] CAS | [CAS CAS]	
CDS --> CI | TET |	
CAS | [CAS CAS CAS] | CAS [CAS] CAS	
CAS --> PBS CDS	
PBS --> PCI	rule(W_<*PCI.Construct*>, TRANS{[PROT-CI], '+W_PROT-CI_<*PCI.Construct*>*PROT-CI', 'w0_<*PCI.Construct*>'})
PBS --> PLA	rule(W_<*PLA.Construct*>, TRANS{[PROT-LAC], '+W_PROT-LAC_<*PLA.Construct*>*PROT-LAC', 'w0_<*PLA.Construct*>'})
PBS --> PTE	rule(W_<*PTE.Construct*>, TRANS{[PROT-TET], '+W_PROT-TET_<*PTE.Construct*>*PROT-TET', 'w0_<*PTE.Construct*>'})
FP --> GFP	
CAS --> PBS CDS FP	
SPL -->) (CAS VEC	
S --> MDL (2CAS	
CAS --> PBS FP TER	

Rules	**Parameters**
2CAS --> CAS CAS | [CAS] CAS | [CAS CAS]	
CDS --> CI | TET | LAC	
3CAS --> CAS CAS CAS | [CAS CAS CAS] | CAS [CAS] CAS	
CAS --> PBS CDS TER	
PBS --> PCI	parameter(W_<*PCI.Construct*>)
PBS --> PLA	parameter(W_<*PLA.Construct*>)
PBS --> PTE	parameter(W_<*PTE.Construct*>)
FP --> GFP	
CAS --> PBS CDS FP TER	
SPL -->) (CAS VEC SPL |)	
S --> MDL (2CAS VEC SPL	
CAS --> PBS FP TER	

		Parameters	Events
	init_switch_LAC_CI	parameter('init_LAC_lacI',1E-21),parameter('init_TET_TetR',5E-21),parameter('init_CI_cits',0	event('add_IPTG',[['W_PROT_LAC_lacI_placI',0],['W_PROT_LAC_lacI_GFP_placI',0]],'"geq(time,20)"'),event('phase1',[['W_PROT_LAC_lacI_placI',-1.5E20],['W_PROT_LAC_lacI_GFP_placI',-1.5E20]],'"geq(time,40)"'),event('heat',[['PROT_CI_cits',0],['PROT_CI_cits_GFP',0]],'"geq(time,80)"')
	init_switch_TET_CI	parameter('init_LAC_lacI',1E-21),parameter('init_TET_TetR',5E-21),parameter('init_CI_cits',0)	event('add_aTc',[['W_PROT_TET_TetR_pTetR',0],['W_PROT_TET_TetR_GFP_pTetR',0]],'"geq(time,20)"'),event('phase1',[['W_PROT_TET_TetR_pTetR',-1.5E20],['W_PROT_TET_TetR_GFP_pTetR',-1.5E20]],'"geq(time,40)"'),event('heat',[['PROT_CI_cits',0],['PROT_CI_cits_GFP',0]],'"geq(time,80)"')
	init_switch_LAC_TET	parameter('init_LAC_lacI',1E-21),parameter('init_TET_TetR',5E-21),parameter('init_CI_cits',0)	event('add_aTc',[['W_PROT_LAC_lacI_placI',-1.5E20],['W_PROT_TET_TetR_pTetR',0],['W_PROT_LAC_lacI_GFP_placI',-1.5E20],['W_PROT_TET_TetR_GFP_pTetR',0]],'"geq(time,20)"'),event('phase1',[['W_PROT_LAC_lacI_placI',-1.5E20],['W_PROT_TET_TetR_pTetR',-1.5E20],['W_PROT_LAC_lacI_GFP_placI',-1.5E20],['W_PROT_TET_TetR_GFP_pTetR',-1.5E20]],'"geq(time,40)"'),event('add_IPTG',[['W_PROT_LAC_lacI_placI',0],['W_PROT_TET_TetR_pTetR',-1.5E20],['W_PROT_LAC_lacI_GFP_placI',0],['W_PROT_TET_TetR_GFP_pTetR',-1.5E20]],'"geq(time,80)"'),event('phase2',[['W_PROT_LAC_lacI_placI',-1.5E20],['W_PROT_TET_TetR_pTetR',-1.5E20],['W_PROT_LAC_lacI_GFP_placI',-1.5E20],['W_PROT_TET_TetR_GFP_pTetR',-1.5E20]],'"geq(time,100)"')
MDL	**Init_osci**	parameter('init_LAC_lacI',1E-21),parameter('init_TET_TetR',5E-21),parameter('init_CI_cits',0)	
VEC	**E_coli_vector**		
GFP	**GFP**		
PTE	**pTetR**	parameter('s_pTetR',8),parameter('w0_pTetR',0.5)	
PLA	**placI**	parameter('s_placI',8),parameter('w0_placI',0.5)	
PCI	**pcI**	parameter('s_pcI',8),parameter('w0_pcI',0.5)	
TER	**B0010**		
LAC	**lacI**	parameter('g_LAC_lacI',3),parameter_notconstant('W_PROT_LAC_lacI_placI',-1.5E20),parameter_notconstant('W_PROT_LAC_lacI_GFP_placI',-1.5E20)	
TET	**tetR**	parameter('g_TET_TetR',0.6),parameter_notconstant('W_PROT_TET_TetR_pTetR',-1.5E20),parameter_notconstant('W_PROT_TET_TetR_GFP_pTetR',-1.5E20)	
CI	**cIts**	parameter('g_CI_cits',0.6),parameter_notconstant('W_PROT_CI_cits_pcI',-1.5E20),parameter_notconstant('W_PROT_CI_cits_GFP_pcI',-1.5E20)	

Mass-Action Attribute Grammar

Rules	**Species**
2CAS --> CAS CAS | [CAS] CAS | [CAS CAS]	
CDS --> CI | TET | LAC	
3CAS --> CAS CAS CAS | [CAS CAS CAS] | CAS [CAS] CAS	
CAS --> PRO CIS TER	species_amount(DNA_*<PRO.Construct>*,1)
PRO --> PCI	
PRO --> PLA	
PRO --> PTE	
FP --> GFP	
CAS --> PRO CIS TER	species(PROT_*<CDS.Construct>*_<FP.Construct>, 0)
SPL -->) (CAS VEC SPL |)	
S --> MDL (2CAS VEC SPL	
CIS --> RBS CDS	species(mRNA_*<CIS.Construct>*,0),species(PROT_*<CDS.Construct>*,init_*<CDS.Construct>*.getValue())
CIS --> RBS CDS FP	species(mRNA_*<CIS.Construct>*,0),species(PROT_<CDS.Construct>_*<FP.Construct>*,0)
CIS --> RBS FP	species(mRNA_*<CIS.Construct>*,0),species(PROT_*<FP.Construct>*,0)

Rules	**Reactions**
2CAS --> CAS CAS | [CAS] CAS | [CAS CAS]	
CDS --> CI | TET | LAC	
3CAS --> CAS CAS CAS | [CAS CAS CAS] | CAS [CAS] CAS	
CAS --> PRO CIS TER	reaction(Transcription_<*CAS.Construct*>, [*DNA*_<*PRO.Construct*>], [], [mRNA_<*CIS.Construct*>], "k_transcription_<*PRO.Construct*>")
PRO --> PCI	
PRO --> PLA	
PRO --> PTE	
FP --> GFP	
CAS --> PRO CIS TER	
SPL -->) (CAS VEC SPL |)	
S --> MDL (2CAS VEC SPL	
CIS --> RBS CDS	reaction(Translation_<*CIS.Construct*>, [mRNA_<*CIS.Construct*>], [], [PROT_<*CDS.Construct*>], "k_translation_<*RBS.Construct*>"),reaction(Degradation_mRNA_<*CIS.Construct*>, [], [*mRNA*_<*CIS.Construct*>], [], "-k_degradation_<*RBS.Construct*>*mRNA_<*CIS.Construct*>"),reaction(Degradation_PROT_<*CDS.Construct*>, [], [PROT_<*CDS.Construct*>], [], "-k_degradation_<*CDS.Construct*>*PROT_<*CDS.Construct*>")
CIS --> RBS CDS FP	reaction(Translation_<*CIS.Construct*>, [mRNA_<*CIS.Construct*>], [], [PROT_<*CDS.Construct*>_<*FP.Construct*>], "k_translation_<*RBS.Construct*>"),reaction(Degradation_mRNA_<*CIS.Construct*>, [], [*mRNA*_<*CIS.Construct*>], [], "-k_degradation_<*RBS.Construct*>*mRNA_<*CIS.Construct*>"),reaction(Degradation_PROT_<*CDS.Construct*>_<*FP.Construct*>, [], [PROT_<CDS.Construct>_<*FP.Construct*>], [], "-k_degradation_<*CDS.Construct*>*PROT_<*CDS.Construct*>_<*FP.Construct*>")
CIS --> RBS FP	reaction(Translation_<*CIS.Construct*>, [mRNA_<*CIS.Construct*>], [], [PROT_<*FP.Construct*>], "k_translation_<*RBS.Construct*>"),reaction(Degradation_mRNA_<*CIS.Construct*>, [], [*mRNA*_<*CIS.Construct*>], [], "-k_degradation_<*RBS.Construct*>*mRNA_<*CIS.Construct*>"),reaction(Degradation_PROT_<*FP.Construct*>, [], [PROT_<*FP.Construct*>], [], "-k_degradation_<*FP.Construct*>*PROT_<*FP.Construct*>")

Rules	**Trans**
2CAS --> CAS CAS | [CAS] CAS | [CAS CAS]	
CDS --> CI | TET | LAC	
3CAS --> CAS CAS CAS | [CAS CAS CAS] | CAS [CAS] CAS	
CAS --> PRO CIS TER	
PRO --> PCI	TRANSfor_each_species{PROT-CI,parameter(k_binding_PROT-CI_<*PCI.Construct*>,1)}, TRANSfor_each_species{PROT-CI,parameter(k_release_PROT-CI_<*PCI.Construct*>,0.1)}, reaction_rev(interaction_CI_<*PCI.Construct*>,[],TRANSspecies{PROT-CI},TRANSspecies_and_declare{[PROT-CI],PROT-CI_DNA_<*PCI.Construct*>_x,0},Math_Interaction)
PRO --> PLA	TRANSfor_each_species{PROT-LAC,parameter(k_binding_PROT-LAC_<*PLA.Construct*>,1)}, TRANSfor_each_species{PROT-LAC,parameter(k_release_PROT-LAC_<*PLA.Construct*>,0.1)}, reaction_rev(interaction_LAC_<*PLA.Construct*>,[],TRANSspecies{PROT-LAC},TRANSspecies_and_declare{[PROT-LAC],PROT-LAC_DNA_<*PLA.Construct*>_x,0},Math_Interaction)
PRO --> PTE	TRANSfor_each_species{PROT-TET,parameter(k_binding_PROT-TET_<*PTE.Construct*>,1)}, TRANSfor_each_species{PROT-TET,parameter(k_release_PROT-TET_<*PTE.Construct*>,0.1)}, reaction_rev(interaction_TET_<*PTE.Construct*>,[],TRANSspecies{PROT-TET},TRANSspecies_and_declare{[PROT-TET],PROT-TET_DNA_<*PTE.Construct*>_x,0},Math_Interaction)
FP --> GFP	
CAS --> PRO CIS TER	
SPL -->) (CAS VEC SPL |)	
S --> MDL (2CAS VEC SPL	
CIS --> RBS CDS	
CIS --> RBS CDS FP	
CIS --> RBS FP	

Category	*PartsID*	**Parameters**
	Init_osci	parameter('init_PROT_LAC_lacI',0),parameter('init_PROT_CI_cIts',50),parameter('init_PROT_TET_tetR',100)
VEC	**E_coli_vector**	
GFP	**GFP**	parameter('k_degradation_GFP',0.1)
PTE	**pTetR**	parameter('k_transcription_pTetR',10)
PLA	**placI**	parameter('k_transcription_placI',25)
PCI	**pcI**	parameter('k_transcription_pcI',50)
TER	**B0010**	
RBS	**Strong_RBS**	parameter('k_translation_Strong_RBS',25),parameter('k_degradation_Strong_RBS',1)
LAC	**lacI**	parameter('k_degradation_LAC_lacI',0.1)
TET	**tetR**	parameter('k_degradation_TET_tetR',0.1)
CI	**cIts**	parameter('k_degradation_CI_cIts',0.1)

Discrete Gene Regulatory Networks

Rules	*Categories*	**Nodes**
2CAS --> CAS CAS | [CAS CAS] |	**2CAS**	
	CAS-o1	
	CAS	
3CAS --> CAS CAS CAS | [CAS CAS CAS] |	**3CAS**	
	CAS-o2	
	CAS-o1	
	CAS	
CAS --> PBS CDS TER	**CAS**	node(PROT_*<CDS.name>*, 0, *<CAS.MaxVal>*)
	PBS	
	CDS	
	TER	
PBS --> PCI | PCA | PTE	**PBS**	
	PCI | PLA | PTE	
S --> (2CAS VEC) | (3CAS VEC)	**S**	
	2CAS | 3CAS	
	VEC	
S --> LBAC CI MBAC CRO RBAC	**S**	node(PROT_*<CI.name>*, 0, *1*), node(PROT_<CRO.name>, 0, 2)
	CI	
	CRO	

Rules	*Categories*	**MaxVal**
2CAS --> CAS CAS | [CAS CAS] | [CAS] CAS	**2CAS**	n/a
	CAS-o1	2
	CAS	2
3CAS --> CAS CAS CAS | [CAS CAS CAS] | CAS [CAS] CAS	**3CAS**	n/a
	CAS-o2	3
	CAS-o1	3
	CAS	3
CAS --> PBS CDS TER	**CAS**	
	PBS	n/a
	CDS	n/a
	TER	n/a
PBS --> PCI | PCA | PTE	**PBS**	n/a
	PCI | PLA | PTE	n/a
S --> (2CAS VEC) | (3CAS VEC)	**S**	n/a
	2CAS | 3CAS	n/a
	VEC	n/a
S --> LBAC CI MBAC CRO RBAC	**S**	n/a
	CI	n/a
	CRO	n/a

Rules	*Categories*	**Ressources**
2CAS --> CAS CAS | [CAS CAS] | [CAS] CAS	**2CAS**	
	CAS-o1	
	CAS	
3CAS --> CAS CAS CAS | [CAS CAS CAS] | CAS [CAS] CAS	**3CAS**	
	CAS-o2	
	CAS-o1	
	CAS	
CAS --> PBS CDS TER	**CAS**	*<PBS.Ressources>*=edge(From, Sign, Val); TRANSfor_each_species{*<From>*,edge(*<From>*,PROT_*<CDS.Name>*,*<Sign>*,*<Val>*)}
	PBS	
	CDS	
	TER	
PBS --> PCI | PCA | PTE	**PBS**	
	PCI | PLA | PTE	
S --> (2CAS VEC) | (3CAS VEC)	**S**	
	2CAS | 3CAS	
	VEC	
S --> LBAC CI MBAC CRO RBAC	**S**	<CI.Ressources>=edge(From, Sign, Val),edge(From2, Sign2, Val2); TRANSfor_each_species{<From>,edge(<From>,PROT_<CI.Name>,<Sign>,<Val>)},TRANSfor_each_species{<From2>,edge(<From2>,PROT_<CI.Name>,<Sign2>,<Val2>)}, <CRO.Ressources>=edge(From, Sign, Val),edge(From2, Sign2, Val2); TRANSfor_each_species{<From>,edge(<From>,PROT_<CRO.Name>,<Sign>,<Val>)},TRANSfor_each_species{<From2>,edge(<From2>,PROT_<CRO.Name>,<Sign2>,<Val2>)}
	CI	
	CRO	

Category	*PartsID*	**Reactions**
VEC	**E_coli_vector**	
PTE	**pTetR**	edge('PROT-tetR','negative',1)
PLA	**placI**	edge('PROT-lacI','negative',1)
PCI	**pcI**	edge('PROT-cIts','negative',1)
TER	**B0010**	
CDS	**tetR01**	
	lacI01	
	cIts01	
CI	**ci**	edge('PROT-cro','negative',1),edge('PROT-ci','positive',1)
CRO	**cro**	edge('PROT-ci','negative',1),edge('PROT-cro','negative',2)

Layered AND gates Attribute Grammar

Grammar rules and semantic actions from the generated compiler

```
check_grammar(Tokens) :- init_sbml,c117413963(S1, S2, S3, S4, S5, S6, S7,
    Tokens, []) ,%Construct, Parameters, Species, Reactions, Events, Trans, Rules

      list_to_set(S2,Species),
      list_to_set(S3,Parameters),
      list_to_set(S4,Reactions),
      list_to_set(S5,Events),
      list_to_set(S6,Trans),
      list_to_set(S7,Rules),
      nl,nl,
      process_list(Parameters),nl,
      process_list(Species),nl,
      process_list(Rules),nl,
      process_list(Reactions),nl,
      process_list(Events),nl,
      analyze_trans(Trans,Species),nl,
      end_sbml .


%S --> 2AND
c117413963(A4647, A4648, A4649, A4650, A4651, A4652, A4653) -->
    c117413978(A4709, A4710, A4711, A4712, A4713, A4714, A4715), {
%%name
A4709=A4647,

%%pass list attributes
append([A4710], A4648),
append([A4711], A4649),
append([A4712], A4650),
append([A4713], A4651),
append([A4714], A4652),
append([A4715], A4653)}.

%2AND --> Chap TFplas OutPlas
c117413978(A4709, A4710, A4711, A4712, A4713, A4714, A4715) -->
    c117413977(A4702, A4703, A4704, A4705, A4706, A4707, A4708),
    c117413973(A4675, A4676, A4677, A4678, A4679, A4680, A4681),
    c117413986(A4764, A4765, A4766, A4767, A4768, A4769, A4770), {
%%name
```

```
atomic_list_concat([A4702,'_', A4675,'_', A4764], A4709),

%%lists
append([A4703, A4676, A4766], A4710),

append([A4704, A4677, A4765], A4711),

append([A4708, A4681, A4767], A4715),

append([A4705, A4678, A4769], A4712),

append([A4706, A4679, A4768], A4713),

append([A4707, A4680, A4770], A4714)}.

%TFplas --> ( VEC [ Pcons InputP TER ] Pacti Acti TER )
c117413973(A4675, A4676, A4677, A4678, A4679, A4680, A4681) -->
    c117413966(Name117413966), c117413979(Name117413979, A4716, A4717, A4718,
    A4719, A4720, A4721, A4722), c117413964(Name117413964),
    c117413971(Name117413971, A4661, A4662, A4663, A4664, A4665, A4666, A4667),
    c117413976(Name117413976, A4695, A4696, A4697, A4698, A4699, A4700, A4701),
    c117413970(Name117413970o1, A4654o1, A4655o1, A4656o1, A4657o1, A4658o1,
    A4659o1, A4660o1), c117413965(Name117413965), c117413980(Name117413980,
    A4723, A4724, A4725, A4726, A4727, A4728, A4729), c117413984(Name117413984,
    A4751, A4752, A4753, A4754, A4755, A4756, A4757), c117413970(Name117413970,
    A4654, A4655, A4656, A4657, A4658, A4659, A4660), c117413967(Name117413967), {
%%name
atomic_list_concat([A4716,'_', A4661,'_', A4695,'_', A4654o1,'_', A4723,'_',
    A4751,'_', A4654], A4675),

%%pass lists
atomic_list_concat(['PROT_Activator_', A4751],Species_Prot),
append([[species(Species_Prot,0)], A4717, A4662, A4696, A4655o1, A4724, A4752,
    A4655], A4676),

atomic_list_concat(['"Oa_PROT_Activator_', A4751,'*p_', A4723,'"'],Rule_math),

append([A4719, A4664, A4698, A4657o1, A4726, A4754, A4657], A4678),

append([A4718, A4663, A4697, A4656o1, A4725, A4753, A4656], A4677),

append([[rule(Species_Prot,Rule_math)], A4722, A4667, A4701, A4660o1, A4729,
    A4757, A4660], A4681),

append([A4720, A4665, A4699, A4658o1, A4727, A4755, A4658], A4679),
```

```
append([A4721, A4666, A4700, A4659o1, A4728, A4756, A4659], A4680)}.

%Chap --> ( VEC [ Pcons InputP TER ] Pchap ChapP TER )
c117413977(A4702, A4703, A4704, A4705, A4706, A4707, A4708) -->
    c117413966(Name117413966), c117413979(Name117413979, A4716, A4717, A4718,
    A4719, A4720, A4721, A4722), c117413964(Name117413964),
    c117413971(Name117413971, A4661, A4662, A4663, A4664, A4665, A4666, A4667),
    c117413976(Name117413976, A4695, A4696, A4697, A4698, A4699, A4700, A4701),
    c117413970(Name117413970o1, A4654o1, A4655o1, A4656o1, A4657o1, A4658o1,
    A4659o1, A4660o1), c117413965(Name117413965), c117413982(Name117413982,
    A4737, A4738, A4739, A4740, A4741, A4742, A4743), c117413972(Name117413972,
    A4668, A4669, A4670, A4671, A4672, A4673, A4674), c117413970(Name117413970,
    A4654, A4655, A4656, A4657, A4658, A4659, A4660), c117413967(Name117413967), {
%%name and pass list attributes
atomic_list_concat([A4716,'_', A4661,'_', A4695,'_', A4654o1,'_', A4737,'_',
    A4668,'_', A4654], A4702),

%%used PROT and Chaperone keywords in species name
atomic_list_concat(['PROT_Chaperone_', A4668],Species_Name),
atomic_list_concat(['"Oc_PROT_Chaperone_', A4668,'*p_', A4737,'"'],Rule_math),


append([[species(Species_Name,0)], A4717, A4662, A4696, A4655o1, A4738, A4669,
    A4655], A4703),

append([A4719, A4664, A4698, A4657o1, A4740, A4671, A4657], A4705),
append([A4718, A4663, A4697, A4656o1, A4739, A4670, A4656], A4704),

append([[rule(Species_Name,Rule_math)], A4722, A4667, A4701, A4660o1, A4743,
    A4674, A4660], A4708),

append([A4721, A4666, A4700, A4659o1, A4742, A4673, A4659], A4707)}.

%OutPlas --> ( VEC Poutput RFP TER )
c117413986(A4764, A4765, A4766, A4767, A4768, A4769, A4770) -->
    c117413966(Name117413966), c117413979(Name117413979, A4716, A4717, A4718,
    A4719, A4720, A4721, A4722), c117413981(Name117413981, A4730, A4731, A4732,
    A4733, A4734, A4735, A4736), c117413983(Name117413983, A4744, A4745, A4746,
    A4747, A4748, A4749, A4750), c117413970(Name117413970, A4654, A4655, A4656,
    A4657, A4658, A4659, A4660), c117413967(Name117413967), {
%% name
atomic_list_concat([A4716,'_', A4730,'_', A4744,'_', A4654], A4764),

%%list
```

```
append([A4717, A4731, A4745, A4655], A4766),

atomic_list_concat(['pr_', A4730],NameParam),

append([[parameter(NameParam)], A4718, A4732, A4746, A4656], A4765),
B='\'',
atomic_list_concat(['rule(pr_', A4730, ',TRANS\{[PROT-Activator,
    PROT-Chaperone], kr_', A4730, '*((K1_', A4730, '+K2_', A4730,
    '/Kac_PROT-Chaperone*sqr(PROT-Chaperone)*PROT-Activator) / (1+K1_', A4730,
    '+K2_', A4730,
    '/Kac_PROT-Chaperone*PROT-Activator*sqr(PROT-Chaperone))),\})'], RuleTrans),

append([A4722, A4736, A4750, A4660], A4767),

append([A4719, A4733, A4747, A4657], A4769),

append([A4720, A4734, A4748, A4658], A4768),

append([[RuleTrans], A4721, A4735, A4749, A4659], A4770)}.

%S --> Chap PlasM OutPlas
c117413963(A4647, A4648, A4649, A4650, A4651, A4652, A4653) -->
    c117413977(A4702, A4703, A4704, A4705, A4706, A4707, A4708),
    c117413987(A4771, A4772, A4773, A4774, A4775, A4776), c117413986(A4764,
    A4765, A4766, A4767, A4768, A4769, A4770), {
%name
atomic_list_concat([A4702, A4771, A4764],A4647),

append([A4703, A4772, A4765], A4648),
                                                        append([A4704, A4773,
                                                            A4766], A4649),
                                                        append([A4705, A4774,
                                                            A4767], A4650),
                                                        append([A4706, A4775,
                                                            A4768], A4651),
                                                        append([A4708,
                                                            A4769], A4653),
                                                        append([A4707, A4776,
                                                            A4770], A4652)
}.

%PlasM --> ( VEC [ Pcons InputP TER ] Pchap ChapP TER VEC [ Pacti Acti TER ]
    Pcons InputP TER VEC Pacti Acti TER )
c117413987(A4771, A4772, A4773, A4774, A4775, A4776) -->
    c117413966(Name117413966), c117413979(Name117413979o2, A4716o2, A4717o2,
```

```
    A4718o2, A4719o2, A4720o2, A4721o2, A4722o2), c117413964(Name117413964o1),
    c117413971(Name117413971o1, A4661o1, A4662o1, A4663o1, A4664o1, A4665o1,
    A4666o1, A4667o1), c117413976(Name117413976o1, A4695o1, A4696o1, A4697o1,
    A4698o1, A4699o1, A4700o1, A4701o1), c117413970(Name117413970o4, A4654o4,
    A4655o4, A4656o4, A4657o4, A4658o4, A4659o4, A4660o4),
    c117413965(Name117413965o1), c117413982(Name117413982, A4737, A4738, A4739,
    A4740, A4741, A4742, A4743), c117413972(Name117413972, A4668, A4669, A4670,
    A4671, A4672, A4673, A4674), c117413970(Name117413970o3, A4654o3, A4655o3,
    A4656o3, A4657o3, A4658o3, A4659o3, A4660o3), c117413979(Name117413979o1,
    A4716o1, A4717o1, A4718o1, A4719o1, A4720o1, A4721o1, A4722o1),
    c117413964(Name117413964), c117413980(Name117413980o1, A4723o1, A4724o1,
    A4725o1, A4726o1, A4727o1, A4728o1, A4729o1), c117413984(Name117413984o1,
    A4751o1, A4752o1, A4753o1, A4754o1, A4755o1, A4756o1, A4757o1),
    c117413970(Name117413970o2, A4654o2, A4655o2, A4656o2, A4657o2, A4658o2,
    A4659o2, A4660o2), c117413965(Name117413965), c117413971(Name117413971,
    A4661, A4662, A4663, A4664, A4665, A4666, A4667), c117413976(Name117413976,
    A4695, A4696, A4697, A4698, A4699, A4700, A4701), c117413970(Name117413970o1,
    A4654o1, A4655o1, A4656o1, A4657o1, A4658o1, A4659o1, A4660o1),
    c117413979(Name117413979, A4716, A4717, A4718, A4719, A4720, A4721, A4722),
    c117413980(Name117413980, A4723, A4724, A4725, A4726, A4727, A4728, A4729),
    c117413984(Name117413984, A4751, A4752, A4753, A4754, A4755, A4756, A4757),
    c117413970(Name117413970, A4654, A4655, A4656, A4657, A4658, A4659, A4660),
    c117413967(Name117413967), {
%%name
atomic_list_concat([A4716o2, A4661o1, A4695o1, A4654o4, A4737, A4668, A4654o3,
    A4716o1, A4723o1, A4751o1, A4654o2, A4661, A4695, A4654o1, A4716, A4654o1,
    A4723, A4751, A4654], A4771),


%%Chap1
atomic_list_concat(['PROT_ChapI_', A4668], Species_Name1),
atomic_list_concat(['"Oc_PROT_Chaperone_', A4668,'*p_', A4737,'"'], Rule_math1),

%%Acti1
atomic_list_concat(['PROT_ActiI_', A4751o1],Species_Name2),
atomic_list_concat(['"Oa_PROT_Activator_', A4751o1,'*p_', A4723o1,'"'],
    Rule_math2),

%%Acti
atomic_list_concat(['PROT_Activator_', A4751],Species_Name3),
atomic_list_concat(['"Oa_PROT_Activator_', A4751,'*pr_', A4723,'"'], Rule_math3),

%%transs pr with Chap1 and Acti1
atomic_list_concat(['pr_', A4723],NameParam),
B='\'',
```

```
atomic_list_concat(['PROT-', A4751o1],ProtActi),
atomic_list_concat(['PROT-', A4668],ProtChap),
atomic_list_concat(['rule(pr_', A4723, ',TRANS\{[',ProtActi, ',', ProtChap,
    '],kr_', A4723, '*((K1_', A4723, '+K2_', A4723, '/Kac_', ProtChap, '*sqr(',
    ProtChap, ')*', ProtActi, ')/(1+K1_', A4723, '+K2_', A4723,'/Kac_', ProtChap,
    '*', ProtActi, '*sqr(', ProtChap, '))),\})'], RuleTrans),


%%lists
%species
append([[species(Species_Name1, 0), species(Species_Name2, 0),
    species(Species_Name3, 0)], A4717o2, A4662o1, A4696o1, A4655o4, A4738, A4669,
    A4655o3, A4717o1, A4724o1, A4752o1, A4655o2, A4662, A4696, A4655o1, A4717,
    A4724, A4752, A4655], A4772),

%parameters
append([[parameter(NameParam)], A4718o2, A4663o1, A4697o1, A4656o4, A4739,
    A4670, A4656o3, A4718o1, A4725o1, A4753o1, A4656o2, A4663, A4697, A4656o1,
    A4718, A4725, A4753, A4656], A4773),

%reactions
append([A4719o2, A4664o1, A4698o1, A4657o4, A4740, A4671, A4657o3, A4719o1,
    A4726o1, A4754o1, A4657o2, A4664, A4698, A4657o1, A4719, A4726, A4754,
    A4657], A4774),

%rules
append([[rule(Species_Name1, Rule_math1), rule(Species_Name2, Rule_math2),
    rule(Species_Name3, Rule_math3)], A4722o2, A4667o1, A4701o1, A4660o4, A4743,
    A4672, A4660o3, A4722o1, A4729o1, A4757o1, A4660o2, A4667, A4701, A4660o1,
    A4722, A4729, A4757, A4660], A4775),

%trans
append([[RuleTrans], A4721o2, A4666o1, A4700o1, A4659o4, A4742, A4673, A4659o3,
    A4721o1, A4728o1, A4756o1, A4659o2, A4666, A4700, A4659o1, A4721, A4728,
    A4756, A4659], A4776)}.
```

Library of parts from the generated compiler

```
%---------------PARTS---------------

%(-->[Opening_plasmid_delimiter]
c117413966('Opening_plasmid_delimiter') --> [p22511].
%VEC-->[pSC101*]
c117413979('pSC101*','psc101s',[],[],[],[],[],[]) --> [p22523].
```

```
%VEC-->[colE1]
c117413979('colE1','cole1',[],[],[],[],[],[]) --> [p22524].
%VEC-->[p15A]
c117413979('p15A','p15a',[],[],[],[],[],[]) --> [p22525].
%[-->[Opening_reverse_complement_delimiter]
c117413964('Opening_reverse_complement_delimiter') --> [p22509].
%Pcons-->[BBa_J23100]
c117413971('BBa_J23100','pcons',[],[],[],[],[],[]) --> [p22527].
%InputP-->[araC]
c117413976('araC','araC',[],[],[],[],[],[]) --> [p22519].
%InputP-->[tetR]
c117413976('tetR','tetr',[],[],[],[],[],[]) --> [p22520].
%InputP-->[lacI]
c117413976('lacI','laci',[],[],[],[],[],[]) --> [p22531].
%TER-->[ter]
c117413970('ter','ter',[],[],[],[],[],[]) --> [p22526].
%]-->[Closing_reverse_complement_delimiter]
c117413965('Closing_reverse_complement_delimiter') --> [p22510].
%Pacti-->[pTet]
c117413980('pTet','pTet',[],[parameter('aTc',10), parameter('Kd_pTet', 3.6),
    parameter('n_pTet', 2), parameter('K1_pTet', 350), parameter('K2_pTet', 51),
    parameter('Fmax_pTet', 0.26), parameter('p_pTet'),
    parameter('ft_pTet')],[],[],[],[atomic_list_concat(['"Fmax_pTet*(K1_pTet/(1 +
    K1_pTet + 2*K2_pTet*ft_pTet+sqr(K2_pTet)*sqr(ft_pTet)))"'], Rule2_math),
    rule('p_pTet', Rule2_math),
    atomic_list_concat(['"1-power(aTc,n_pTet)/(power(Kd_pTet,n_pTet) +
    power(aTc,n_pTet))"'],Rule1_math), rule('ft_pTet',Rule1_math)]) --> [p22522].
%Pacti-->[pBAD]
c117413980('pBAD','pBAD',[], [parameter('Ara',10), parameter('Kd_pBAD', 0.1),
    parameter('n_pBAD', 2.2), parameter('K1_pBAD', 0.014), parameter('K2_pBAD',
    12), parameter('K3_pBAD', 4.4), parameter('Fmax_pBAD', 0.2),
    parameter('p_pBAD'), parameter('ftl_pBAD')], [], [], [],[
    atomic_list_concat(['"Fmax_pBAD * ((K1_pBAD+K2_pBAD*ftl_pBAD) / (1 + K1_pBAD+
    K2_pBAD*ftl_pBAD + K3_pBAD*(1-ftl_pBAD)))"'], Rule2_math), rule('p_pBAD',
    Rule2_math), atomic_list_concat([ '"power(Ara,n_pBAD) /
    (power(Kd_pBAD,n_pBAD) + power(Ara,n_pBAD))"' ], Rule1_math),
    rule('ftl_pBAD', Rule1_math)]) --> [p22521].
%Pacti-->[pipaHs]
c117413980('pipaHs','pipaHs',[],[parameter('K1_pipaHs', 0.00000026),
    parameter('K2_pipaHs', 450), parameter('kr_pipaHs', 20000)],[],[],[],[]) -->
    [p22530].
%Acti-->[SicA*]
c117413984('SicA*','SicAs',[],[parameter('Oa_PROT_Activator_SicAs', 0.0000095),
    parameter('Oa_PROT_ActiI_SicAs', 0.0000095)],[],[],[],[]) --> [p22516].
%Acti-->[ipgC]
```

```
c117413984('ipgC','ipgC',[],[parameter('Oa_PROT_Activator_ipgC', 0.000026),
    parameter('Oa_PROT_ActiI_ipgC', 0.000026)],[],[],[],[]) --> [p22529].
%)-->[Closing_plasmid_delimiter]
c117413967('Closing_plasmid_delimiter') --> [p22512].
%Pchap-->[pBAD]
c117413982('pBAD','pBAD',[],[parameter('Ara',10), parameter('Kd_pBAD', 0.1),
    parameter('n_pBAD', 2.2), parameter('K1_pBAD', 0.014), parameter('K2_pBAD',
    12), parameter('K3_pBAD', 4.4), parameter('Fmax_pBAD', 0.2),
    parameter('p_pBAD'), parameter('ftl_pBAD')], [], [], [], [
    atomic_list_concat(['"Fmax_pBAD*((K1_pBAD+K2_pBAD*ftl_pBAD) /
    (1+K1_pBAD+K2_pBAD*ftl_pBAD+K3_pBAD*(1-ftl_pBAD)))"'], Rule2_math),
    rule('p_pBAD', Rule2_math), atomic_list_concat(['"power(Ara,n_pBAD) /
    (power(Kd_pBAD,n_pBAD) + power(Ara,n_pBAD))"'], Rule1_math),
    rule('ftl_pBAD',Rule1_math)]) --> [p22521].
%Pchap-->[pTet]
c117413982('pTet','pTet',[],[parameter('aTc',10), parameter('Kd_pTet', 3.6),
    parameter('n_pTet', 2), parameter('K1_pTet', 350), parameter('K2_pTet', 51),
    parameter('Fmax_pTet', 0.26), parameter('p_pTet'), parameter('ft_pTet')], [],
    [], [], [ atomic_list_concat(['"Fmax_pTet*(K1_pTet /
    (1+K1_pTet+2*K2_pTet*ft_pTet + sqr(K2_pTet)*sqr(ft_pTet)))"'], Rule2_math),
    rule('p_pTet', Rule2_math), atomic_list_concat(['"1-power(aTc,n_pTet) /
    (power(Kd_pTet,n_pTet) + power(aTc,n_pTet))"'], Rule1_math),
    rule('ft_pTet',Rule1_math)]) --> [p22522].
%Pchap-->[pTac]
c117413982('pTac','pTac',[],[parameter('iPTG',10), parameter('Kd_pTac', 0.003),
    parameter('n_pTac', 1.6), parameter('K1_pTac', 53), parameter('K2_pTac',
    1950), parameter('Fmax_pTac',0.2), parameter('p_pTac'),
    parameter('ft_pTac')], [], [], [], [atomic_list_concat(['"Fmax_pTac*(K1_pTac
    / (1+K1_pTac+K2_pTac*ft_pTac))"'],Rule2_math), rule('p_pTac', Rule2_math),
    atomic_list_concat(['"1-(power(iPTG,n_pTac) / (power(Kd_pTac,n_pTac) +
    power(iPTG,n_pTac)))"'], Rule1_math), rule('ft_pTac',Rule1_math)]) -->
    [p22532].
%ChapP-->[invF]
c117413972('invF','invF',[],[parameter('Oc_PROT_Chaperone_invF', 0.0011),
    parameter('Kac_PROT_Chaperone_invF', 0.0000000054),
    parameter('Oc_PROT_ChapI_invF', 0.0011), parameter('Kac_PROT_ChapI_invF',
    0.0000000054)],[],[],[],[]) --> [p22515].
%ChapP-->[mxiE]
c117413972('mxiE','mxiE', [], [parameter('Oc_PROT_Chaperone_mxiE', 0.000041),
    parameter('Kac_PROT_Chaperone_mxiE', 0.000000049),
    parameter('Oc_PROT_ChapI_mxiE', 0.000041), parameter('Kac_PROT_ChapI_mxiE',
    0.000000049)],[],[],[],[]) --> [p22528].
%Poutput-->[psicA]
c117413981('psicA','psicA',[],[parameter('K1_psicA', 0.00000014),
    parameter('K2_psicA', 3.6), parameter('kr_psicA', 13000)],[],[],[],[]) -->
```

```
    [p22517].
%Poutput-->[pipaHs]
c117413981('pipaHs','pipaHs',[],[parameter('K1_pipaHs', 0.00000026),
    parameter('K2_pipaHs', 450), parameter('kr_pipaHs', 20000)],[],[],[],[]) -->
    [p22530].
%RFP-->[rfp]
c117413983('rfp','rfp',[],[],[],[],[],[]) --> [p22518].
```

C.0.1 Systems Biology: Biological Languages to Design Natural Genomes with the Cell Cycle Example

We explored the possibility of scaling-up our approach by designing a G2P language for a natural genome from a system biology perspective. Indeed, looking at the system biology model addressed most syntactic issues because we focus only on a set of molecules, often corresponding to discrete genetic areas of interest that are not overlapping.
In the previous sections, the GRN examples have illustrated the use of our semantic DNA compilation system in making biological constructs. We used attribute grammars that appear to be suitable for genotype to phenotype mapping. In order to show the possibility to design not only genetic constructs but genomes, we built an attribute grammar for the budding yeast genome and modeled the yeast cell cycle [Chen et al. 2004]: in a logical and structured fashion, information from genomic databases and mathematical models will be combined for the exploration of mutants.

For instance, the budding yeast cell cycle regulation system [Chen et al. 2004] draws heavily on experiments (131 mutants), but the model itself does not encompass genetic data. We develop a linguistic model of the yeast genome expressing how it encodes a complex regulatory network. First, the syntax of the cell cycle grammar must be defined. We used a systems biology approach to focus on essential elements. Therefore, by looking at the mathematical model, we can identify the protein of interest and use a genomic database, Saccharomyces Genome Database (SGD) [Cherry et al. 1998], to find the corresponding coding sequences and their locations in the genomes of the cell. A map can then be drawn, and the syntax of a grammar proposed C.2. The genome is broken down into its sixteen constituent chromosomes, and each chromosome can be further broken down into the functional elements of the cell cycle it carries.
Then using biological knowledge and the list of mutants and their impact on the mathematical model (`http://mpf.biol.vt.edu/research/budding_yeast_model/pp/mutant_list.php`), the declaration of parameters is made for each wild-type part. The semantic action of each chromosome may carry declarations for the equation of the proteins encoded and their declaration as species as well as rule to compute parameters values or for the formation of complexes.
Some elements of the mathematical model cannot be linked to a DNA sequence, and are either in the category Model or declared in the semantic action of the Start rule. For example, it is the case for the intermediary enzyme (IE) or for the four events. There is an additional category, Init, which contains all the initial conditions for the protein of the system. Init and Model parts offers to choose different sets of parameters (wild type in glucose, or wild type in galactose, etc.) but the parts in these categories are only here for the semantics: they do not have any DNA sequence.

By reusing the SBML library, we implemented in GenoCAD a straightforward version of the yeast genome that outputs the mathematical model for the wild type version in Tables C.0.1 of Appendix C and the GenoCAD and compiler generated files are at `http:`

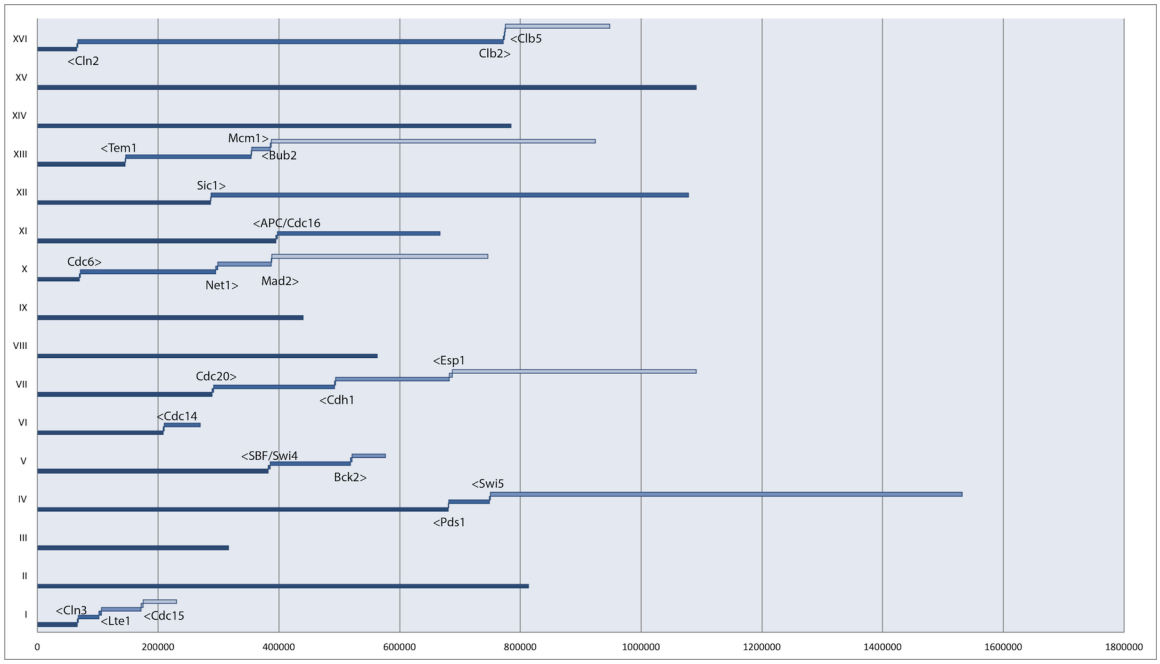


Figure C.2: Syntax proposed for the attribute grammar for the cell cycle of the Budding Yeast Genome. We listed the key proteins for the regulation of the cell cycle used in Chen's model and search for their coding sequences and locations on the yeast genome in the SGD database `www.yeastgenome.org/?`. We were then able to draw this map and derive the syntactic rules for the yeast genome. The 'Genome' is transformed into the 16 chromosomes. Each chromosome may have a rule that let the users choose the coding sequences of the genes located on this chromosome that are related to the regulation of the cell cycle.

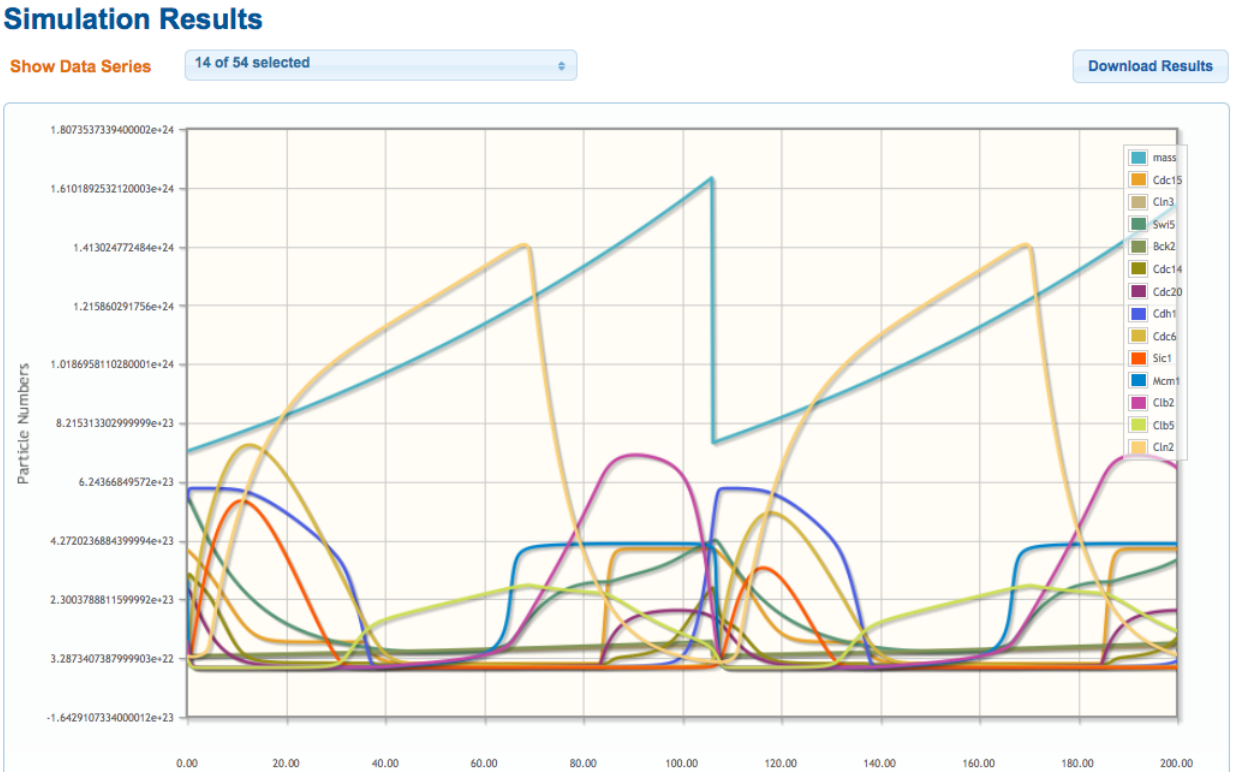


Figure C.3: Simulation results for the cell cycle of the Budding Yeast Genome Wild-Type design in GenoCAD. The time course simulation is ran for 200min. We can observe in blue the mass. The cell divides after about 110 minutes

`//web.figshare.com/download/file/1118850`. The semantics have been implemented according to the BioModel database [Le Novère et al. 2006] at `http://www.ebi.ac.uk/biomodels-main/BIOMD0000000056`. After designing the Wild-Type (WT) genome in GenoCAD, we ran the simulation and obtained the WT phenotype as expected, see Figure C.3.

Additionally,, we can introduce rules to design mutants. For instance, the deletion of cln2 may be syntactically represented by the rule 'Cln2 $\rightarrow$ []'. The semantics of this rule states that the parameter for the synthesis of cln2, kppsn2, is null. When a design uses this rule, the null value of ksppn2 is now used for the SBML model. The simulation of a *cln2*Δ mutant shows that the cell grows bigger and divides after about 160 minutes, as cln2 is responsible for bud emergence and its absence delays it.

References

[1] KC Katherine C Chen et al. "Integrative analysis of cell cycle control in budding yeast." In: *Molecular biology of the cell* 15.8 (Aug. 2004), pp. 3841–3862. ISSN: 1059-1524. DOI: 10.1091/mbc.E03.

[2] JM Cherry et al. "SGD: Saccharomyces genome database". In: *Nucleic acids* 26.1 (1998).

[3] Nicolas Le Novère et al. "BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems." In: *Nucleic acids research* 34.Database issue (Jan. 2006), pp. D689–91. ISSN: 1362-4962. DOI: 10.1093/nar/gkj092.

Attribute Grammar for the Cell Cycle

Parameters

Category	Name	Description	Value
Cln3	**C0**	Max. activity	0.4
	Dn3	Dosage of gene	1
	Jn3	MM constant	6
Cdc15	**ki15**	inactivation	0.5
	kpa15	background activation of Cdc15 by inactive form of	0.002
	ka15p		0.001
	kppa15	Tem1 activation of Cdc15	1
Pds1	**kpspds**	background synthesis	0
	kppd2pds	Cdc20 degradation of Pds1	0.2
	kpd1pds	background degradation	0.01
	kppd3pds	Cdh1 degradation of Pds1	0.04
	kpps1pds	SBF synthesis of Pds1	0.03
	kpps2pds	Mcm1 synthesis of Pds1	0.055
Swi5	**kpsswi**	background synthesis	0.005
	kdswi	degradation	0.08
	kaswi	Cdc14 activation of Swi5	2
	kppsswi	Mcm1 synthesis of Swi5	0.08
	kiswi	Clb2 inactivation of Swi5	0.05
Bck2	**B0**	conc.	0.054
Cdc14	**ks14**	synthesis	0.2
Cdc14 / RENT	**kd14**	degradation	0.1
	kasrent	Cdc14 and Net1 association to	200
	kasrentp	Cdc14 and Net1P association	1
	kdirent	Cdc14 and Net1 dissociation	1
	kdirentp	Cdc14 and Net1P dissociation	2
Cdc20	**kps20**	background synthesis	0.006
	kd20	degradation	0.3
	kpa20	background activation	0.05
	kpps20	Mcm1 synthesis of Cdc20	0.6
	kmad2	Mad2 inactivation of Cdc20	8
	kppa20	IEP: activation of Cdc20	0.2
Cdh1	**kscdh**	synthesis	0.01
	kdcdh	degradation	0.01
	kppacdh		0.8
	kpacdh	background activation	0.01
	kpicdh	background inactivation	0.001
	Jacdh	MM constant a	0.03
	Jicdh	MM constant I	0.03
	Ecdhn2	Cln2 efficiency on Cdh1	0.4
	kppicdh	CDKs: inactivation of Cdh1	0.08
	Ecdhn3	Cln3 efficiency on Cdh1	0.25
	kppacdh	Cdc14 activation of Cdh1	0.8
	Ecdhb2	Clb2 efficiency on Cdh1	1.2
	Ecdhb5	Clb5 efficiency on Cdh1	8

		Pds1 and Esp1 (PE) association	
Esp1	**kasesp**	to form an inactive trimer	50
Esp1 / PE	**kdiesp**	Pds1 and Esp1 (PE) dissociation	0.5
Cdc6	**kpsf6**	background synthesis	0.024
	kd1f6	background phosphorylation	0.01
	kd3f6	degradation of phosphorylated	1
	Ef6b2	Clb2 efficiency on Cdc6	0.55
	kd2f6	CDKs: phosphorylation of Cdc6	1
	kpppsf6	SBF synthesis of Cdc6	0.004
	kppsf6	Swi5 synthesis of Cdc6	0.12
	Ef6k2	Bck2 efficiency on Cdc6	0.03
	Ef6n3	Cln3 efficiency on Cdc6	0.3
	kppf6	Cdc14 dephosphorylation of	4
	Jd2f6		0.05
	Ef6b5	Clb5 efficiency on Cdc6	0.1
	Ef6n2	Cln2 efficiency on Cdc6	0.06
Net1	**ksnet**	synthesis	0.084
	kdnet	degradation	0.03
	kpppnet	background	0.05
	kpkpnet	background phosphorylation	0.01
	kppkpnet	Cdc15 phosphorylation of Net1	0.6
	kppppnet	PPX dephosphorylation of	3
SBF(Swi4)	**kasbf**	activation	0.38
	kpisbf	background inactivation	0.6
	Jasbf	MM constant a	0.01
	Jisbf	MM constant I	0.01
	Esbfn2	Cln2 efficiency on SBF	2
	Esbfb5	Clb5 efficiency on SBF	2
	kppisbf	Clb2 inactivation of SBF	8
	Esbfn3	Cln3 efficiency on Sbf	10
Sic1	**kpsc1**	background synthesis	0.012
	kd1c1	background phosphorylation	0.01
	kd3c1	degradation of phosphorylated	1
	Ec1b5	Clb5 efficiency on Sic1	0.1
	kd2c1	CDKs: phosphorylation of Sic1	1
	Ec1b2	Clb2 efficiency on Sic1	0.45
	Ec1k2	Bck2 efficiency on Sic1	0.03
	Ec1n3	Cln3 efficiency on Sic1	0.3
	kppsc1	Swi5 synthesis of Sic1	0.12
	kppc1	Cdc14 dephosphorylation of	4
	Jd2c1		0.05
	Ec1n2	Cln2 efficiency on Sic1	0.06
Tem1	**Jatem**	MM constant a	0.1
	Jitem	MM constant I	0.1
Mcm1	**kimcm**	inactivation	0.15
	Jamcm	MM constant a	0.1
	Jimcm	MM contant I	0.1
	kamcm	Clb2 activation of Mcm1	1
Cln2	**kpsn2**	background synthesis	0
	kdn2	degradation	0.12
	kppsn2	SBF synthesis of Cln2	0.15
Clb2	**kpsb2**	background synthesis	0.001
Clb2 / C2,F2	**kpdb2**	background degradation	0.003
	kppdb2	Cdh1 degradation of Clb2	0.4
	kdb2p	Cdc20 degradation of Clb2	0.15
	kppsb2	Mcm1 synthesis of Clb2	0.04
		Sic1 and Clb2 (C2) association	
	kasb2	to form an inactive trimer	50
	kdib2	Sic1 and Clb2 (C2) dissociation	0.05
		Cdc6 and Clb2 (F2) association	
	kasf2	to form an inactive trimer	15
	kdif2	Cdc6 and Clb2 (F2) dissociation	0.5

Clb5	**kpsb5**	background synthesis	0.0008
Clb5 / C5,F5	**kpdb5**	background degradation	0.01
	kasb5	Sic1 and Clb5 (C5) association to form an inactive trimer	50
	kdib5	Sic1 and Clb5 (C5) dissociation	0.06
	kasf5	Cdc6 and Clb5 (F5) association to form an inactive trimer	0.01
	kdif5	Cdc6 and Clb5 (F5) dissociation	0.01
	kppdb5	Cdc20 degradation of Clb5	0.16
	kppsb5	MBF(=SBF) dependant	0.005
Mad2	**mad2h**		8
	mad2l		0.01
Bub2	**bub2h**		1
	bub2l		0.2
Lte1	**lte1h**		1
	lte1l		0.1
APC (CDC16)	**kaiep**	Clb2 Activation of IE	0.1
Cdc28 / CDKs,APC-Phc	**kiiep**	IE: inactivation	0.15
	Jaiep	IE: MM constant a	0.1
	Jiiep	IE: MM constant I	0.1
Model	**k**		1
	ksori	ORI: synthesis	2
	kdori	ORI: degradation	0.06
	Eorib2	Clb2 efficiency on ORI	0.45
	Eorib5	Clb5 efficiency on ORI	0.9
	ksbud	BUD: synthesis	0.2
	kdbud	BUD: degradation	0.06
	Ebudn3	Cln3 efficiency on Bud	0.05
	Ebudn2	Cln2 efficiency on Bud	0.25
	Ebudb5	Clb5 efficiency on Bud	1
	kspn	SPN: synthesis	0.1
	kdspn	SPN: degradation	0.06
	Jspn	SPN: MM constant	0.14
	Kez	THRESHOLD: exit from mitosis when Clb2 drops below this	0.3
	Kez2	THRESHOLD: ORI is reset to 0 when Clb2+Clb5 drops below	0.2
	MDT	mass doubling time	90
	ksppx	synthesis	0.1
	kpdppx	background degradation	0.17
	Jpds	MM constant	0.04
	kppdppx	Cdc20 degradation of PPX	2
	J20ppx	Pds1 inhibition of PPX by	0.15

InititalCondition	**BUD_init**	BUD: initial condition	0.008473
	C2_init	C2: initial condition	0.238404
	C2P_init	C2P: initial condition	0.024034
	C5_init	C5: initial condition	0.070081
	C5P_init	C5P: initial condition	0.006878
	Cdc14_init	initial condition	0.468344
	Cdc14T_init	total	2
	Cdc15_init	initial condition	0.656533
	Cdc15T	total	1
	Cdc20_init	initial condition active form	0.444296
	Cdc20i_init	total	1.472044
	Cdc6_init	initial condition	0.10758
	Cdc6P_init	initial condition	0.015486
	Cdh1_init	initial condition	0.930499
	Cdh1i_init	total	0.0695
	Clb2_init	initial condition	0.1469227
	Clb2T_init		0.17
	Clb5_init	initial condition	0.0518014
	Clb5T_init		0.12
	Cln2_init	initial condition	0.0652511
	Esp1_init	initial condition	0.301313
	Esp1T	total	1
	F2_init	F2: initial condition	0.236058
	F2P_init	F2P: initial condition	0.0273938
	F5_init	F5: initial condition	7.24E-005
	F5P_init	F5P: initial condition	7.91E-006
	IET		1.00E+000
	IEP_init	IEP: initial condition	0.1015
	mass_init	mass: initial condition	1.206019
	Net1_init	initial condition	0.018645
	Net1P_init		0.970271
	Net1T_init	total	2.8
	ORI_init	ORI: initial condition	0.000909
	Pds1_init	initial condition	0.025612
	PPX_init	initial condition	0.123179
	RENT_init	RENT: initial condition	1.04954
	RENTP_init		0.6
	Sic1_init	initial condition	0.0228776
	Sic1P_init	initial condition	0.00641
	SPN_init	SPN: initial condition	0.03
	Swi5_init	initial condition	0.95
	Swi5P_init	total	0.02
	Tem1GTP_in	initial condition	0.9
	Tem1T	total	1

Species

	Species	
Rules	Name	InitConc
	mass	mass_init
	IEP	IEP_init
	IE	
	CKIT	
	PPX	PPX_init
Genome --> InitialCondition Model (ChrI) (ChrII) (ChrIII) (ChrIV) (ChrV) (ChrVI) (ChrVII) (ChrVIII) (ChrIX) (ChrX) (ChrXI) (ChrXII) (ChrXIII) (ChrXIV) (ChrXV) (ChrXVI)	**BUD**	BUD_init
	ORI	ORI_init
	SPN	SPN_init
	Cdc15	Cdc15_init
	Cln3	
	Lte1	lte1l
ChrI --> ChrI_L [**Cln3**] ChrI_M1 [**Lte1**] ChrI_M2 [**Cdc15**] ChrI_R	**Cdc15i**	
	Swi5	Swi5_init
	Swi5P	Swi5P_init
ChrIV --> ChrIV_L [**Pds1**] ChrIV_M [**Swi5**] ChrIV_R	**Pds1**	Pds1_init
	SBF	
ChrV --> ChrV_L [SBF] CHrV_M **Bck2** ChrV_R	**Bck2**	
	Cdc14	Cdc14_init
	Cdc14T	Cdc14T_init
	RENTP	RENTP_init
ChrVI --> ChrVI_L [**Cdc14**] ChrVI_R	**RENT**	RENT_init
	Cdc20	Cdc20_init
	Cdc20i	Cdc20i_init
	PE	
	Esp1	Esp1_init
ChrVII --> ChrVII_L **Cdc20** ChrVII_M1 [**Cdh1**] ChrVII_M2 [**Esp1**] ChrVII_R	**Cdh1**	Cdh1_init
	Cdh1i	Cdh1i_init
	Net1	Net1_init
	Net1P	Net1P_init
	Net1T	Net1T_init
	Mad2	mad2l
	Cdc6	Cdc6_init
	Cdc6T	
ChrX --> ChrX_L **Cdc6** ChrX_M1 **Net1** ChrX_M2 **Mad2** ChrX_R	**Cdc6P**	Cdc6P_init
ChrXI --> ChrXI_L [APC] ChrXI_R		
	Sic1	Sic1_init
	Sic1T	
ChrXII --> ChrXII_L **Sic1** ChrXII_R	**Sic1P**	Sic1P_init
	Tem1GDP	
	Tem1GTP	Tem1GTP_in
ChrXIII --> ChrXIII_L [**Tem1**] ChrXIII_M1 **Mcm1** ChrXIII_M2 [**Bub2**] ChrXIII_R	**Mcm1**	
	Bub2	bub2l
	Clb2	Clb2_init
	Clb2T	Clb2T_init
	Clb5	Clb5_init
	Clb5T	Clb5T_init
	Cln2	Cln2_init
	C2	C2_init
	C2P	C2P_init
	C5	C5_init
	C5P	C5P_init
	F2	F2_init
	F2P	F2P_init
ChrXVI --> ChrXVI_L [**Cln2**] ChrXVI_M1 **Clb2** ChrXVI_M2 [**Clb5**] ChrXVI_R	**F5**	F5_init
	F5P	F5P_init

Reactions

			Reaction		
Rules	**Name**	**Modifiers**	**Reactants**	**Products**	**Math**
	Growth			'mass'	mu*mass
	Activation_IEP		'IE'	'IEP'	Vaiep*IE/(Jaiep+IE)
	Inactivation_IEP		'IEP'	'IE'	kiiep*IEP/(Jiiep+IEP)
	Synthesis_PPX			'PPX'	ksppx
	Degradation_PPX		'PPX'		Vdppx*PPX
	DNA_synthesis	'Clb5','Clb2'		'ORI'	ksori*(Eorib5*Clb5+Eorib2*Clb2)
	Neg_DNA_Synthesis		'ORI'		kdori*ORI
Genome --> InitialCondition Model (ChrI) (ChrII) (ChrIII) (ChrIV) (ChrV) (ChrVI) (ChrVII) (ChrVIII) (ChrIX) (ChrX) (ChrXI) (ChrXII) (ChrXIII) (ChrXIV) (ChrXV) (ChrXVI)	Budding	'Cln2','Cln3','Clb5'		'BUD'	ksbud*(Ebudn2*Cln2+Ebudn3*Cln3+Ebudb5*Clb5)
	Neg_budding		'BUD'		kdbud*BUD
	Spindle_formation	'Clb2'		'SPN'	kspn*Clb2/(Jspn+Clb2)
	Spindle_dissa		'SPN'		kdspn*SPN
ChrI --> ChrI_L [**Cln3**] ChrI_M1 [**Lte1**] ChrI_M2 [**Cdc15**] ChrI_R	Activation_Cdc15	'Tem1GTP','Tem1GDP','Cdc14'	'Cdc15i'	'Cdc15'	(kpa15*Tem1GDP+kppa15*Tem1GTP+ka15p*Cdc14)*Cdc15i
	Inactivation_Cdc15		'Cdc15'	'Cdc15i'	ki15*Cdc15
	Synthesis_Swi5	'Mcm1'		'Swi5'	kpsswi+kppsswi*Mcm1
	Degradation_Swi5		'Swi5'		kdswi*Swi5
	Activation_Swi5	'Cdc14'	'Swi5P'	'Swi5'	kaswi*Cdc14*Swi5P
	Inactivation_Swi5	'Clb2'	'Swi5'	'Swi5P'	kiswi*Clb2*Swi5
	Synthesis_Pds1	'SBF','Mcm1'		'Pds1'	kpspds+kpps1pds*SBF+kpps2pds*Mcm1
ChrIV --> ChrIV_L [**Pds1**] ChrIV_M [**Swi5**] ChrIV_R	Degradation_Pds1		'Pds1'		Vdpds*Pds1
	Degradation_Swi5P		'Swi5P'		kdswi*Swi5P
ChrV --> ChrV_L [SBF] CHrV_M **Bck2** ChrV_R					
	Synthesis_Cdc14			'Cdc14'	ks14
	Degradation_Cdc14		'Cdc14'		kd14*Cdc14
	Formation_RENT		'Cdc14','Net1'	'RENT'	kasrent*Cdc14*Net1
	Dissociation_RENT		'RENT'	'Cdc14','Net1'	kdirent*RENT
	Formation_RENTP		'Cdc14','Net1P'	'RENTP'	kasrentp*Cdc14*Net1P
	Dissociation_RENTP		'RENTP'	'Cdc14','Net1P'	kdirentp*RENTP
	Phosphorylation_RENT		'RENT'	'RENTP'	Vkpnet*RENT
	Dephosphorylation_RENTP		'RENTP'	'RENT'	Vppnet*RENTP
	Degradation_RENT_Cdc14		'RENT'	'Cdc14'	kdnet*RENT
	Degradation_RENTP_Cdc14		'RENTP'	'Cdc14'	kdnet*RENTP
	Degradation_RENT_Net1		'RENT'	'Net1'	kd14*RENT
ChrVI --> ChrVI_L [**Cdc14**] ChrVI_R	Degradation_RENTP_Net1P		'RENTP'	'Net1P'	kd14*RENTP
	Synthesis_Cdc20i	'Mcm1'		'Cdc20i'	kps20+kpps20*Mcm1
	Degradation_Cdc20i		'Cdc20i'		kd20*Cdc20i
	Degradation_Cdc20		'Cdc20'		kd20*Cdc20
	Activation_Cdc20	'IEP'	'Cdc20i'	'Cdc20'	(kpa20+kppa20*IEP)*Cdc20i
	Inactivation_Cdc20	'Mad2'	'Cdc20'	'Cdc20i'	k*Mad2*Cdc20
	Synthesis_Cdh1			'Cdh1'	kscdh
	Degradation_Cdh1		'Cdh1'		kdcdh*Cdh1
	Degradation_Cdh1i		'Cdh1i'		kdcdh*Cdh1i
	Activation_Cdh1		'Cdh1i'	'Cdh1'	Vacdh*Cdh1i/(Jacdh+Cdh1i)
	Inactivation_Cdh1		'Cdh1'	'Cdh1i'	Vicdh*Cdh1/(Jicdh+Cdh1)
	Degradation_PE_Esp1		'PE'	'Esp1'	Vdpds*PE
ChrVII --> ChrVII_L **Cdc20** ChrVII_M1 [**Cdh1**] ChrVII_M2 [**Esp1**] ChrVII_R	Formation_PE		'Pds1','Esp1'	'PE'	kasesp*Pds1*Esp1
	Degradation_PE		'PE'	'Pds1','Esp1'	kdiesp*PE
	Synthesis_Cdc6	'Swi5','SBF'		'Cdc6'	kpsf6+kppsf6*Swi5+kpppsf6*SBF
	Phosphorylation_Cdc6		'Cdc6'	'Cdc6P'	Vkpf6*Cdc6
	Dephosphorylation_Cdc6		'Cdc6P'	'Cdc6'	Vppf6*Cdc6P
	Synthesis_Net1			'Net1'	ksnet
	Degradation_Net1P		'Net1P'		kdnet*Net1P
	Phosphorylation_net1		'Net1'	'Net1P'	Vkpnet*Net1
	Dephosphorylation_Net1		'Net1P'	'Net1'	Vppnet*Net1P
ChrX --> ChrX_L **Cdc6** ChrX_M1 **Net1** ChrX_M2 **Mad2** ChrX_R	Degradation_Net1		'Net1'		kdnet*Net1
	Degradation_Cdc6P		'Cdc6P'		kd3f6*Cdc6P
ChrXI --> ChrXI_L [APC] ChrXI_R					
	Synthesis_Sic1	'Swi5'		'Sic1'	kpsc1+kppsc1*Swi5
	Phosphorylation_Sic1		'Sic1'	'Sic1P'	Vkpc1*Sic1
	Dephosphorylation_Sic1P		'Sic1P'	'Sic1'	Vppc1*Sic1P
ChrXII --> ChrXII_L **Sic1** ChrXII_R	Degradation_Sic1P		'Sic1P'		kd3c1*Sic1P
ChrXIII --> ChrXIII_L [**Tem1**] ChrXIII_M1 **Mcm1** ChrXIII_M2 [**Bub2**] ChrXIII_R	Activation_Tem1	'Lte1'	'Tem1GDP'	'Tem1GTP'	Lte1*Tem1GDP/(Jatem+Tem1GDP)
	Inactivation_Tem1	'Bub2'	'Tem1GTP'	'Tem1GDP'	Bub2*Tem1GTP/(Jitem+Tem1GTP)

	Synthesis_Cln2	'SBF','mass'		'Cln2'	(kpsn2+kppsn2*SBF)*mass
	Degradation_Cln2		'Cln2'		kdn2*Cln2
	Synthesis_Clb2	'Mcm1','mass'		'Clb2'	(kpsb2+kppsb2*Mcm1)*mass
	Degradation_Clb2		'Clb2'		Vdb2*Clb2
	Synthesis_Clb5	'SBF','mass'		'Clb5'	(kpsb5+kppsb5*SBF)*mass
	Degradation_Clb5		'Clb5'		Vdb5*Clb5
	Association_Clb2_Sic1		'Clb2','Sic1'	'C2'	kasb2*Clb2*Sic1
	Dissociation_Clb2_Sic1		'C2'	'Clb2','Sic1'	kdib2*C2
	Association_Clb5_Sic1		'Clb5','Sic1'	'C5'	kasb5*Clb5*Sic1
	Dissociation_Clb5_Sic1		'C5'	'Clb5','Sic1'	kdib5*C5
	Phosphorylation_C2		'C2'	'C2P'	Vkpc1*C2
	Dephosphorylation_C2P		'C2P'	'C2'	Vppc1*C2P
	Phosphorylation_C5		'C5'	'C5P'	Vkpc1*C5
	Dephosphorylation_C5P		'C5P'	'C5'	Vppc1*C5P
	Degradation_C2		'C2'	'Sic1'	Vdb2*C2
	Degradation_C5		'C5'	'Sic1'	Vdb5*C5
	Degradation_C2P		'C2P'	'Clb2'	kd3c1*C2P
	Degradation_C5P		'C5P'	'Clb5'	kd3c1*C5P
	Degradation_C2P_sic1		'C2P'	'Sic1P'	Vdb2*C2P
	Degradation_C5P_sic1		'C5P'	'Sic1P'	Vdb5*C5P
	Formation_F2		'Clb2','Cdc6'	'F2'	kasf2*Clb2*Cdc6
	Dissociation_F2		'F2'	'Clb2','Cdc6'	kdif2*F2
	Formation_F5		'Clb5','Cdc6'	'F2'	kasf5*Clb5*Cdc6
	Dissociation_F5		'F5'	'Clb5','Cdc6'	kdif5*F5
	Phosphorylation_F2		'F2'	'F2P'	Vkpf6*F2
	Dephosphorylation_F2		'F2P'	'F2'	Vppf6*F2P
	Phosphorylation_F5		'F5'	'F5P'	Vkpf6*F5
	Dephosphorylation_F5		'F5P'	'F5'	Vppf6*F5P
	Degradation_F2_Cdc6		'F2'	'Cdc6'	Vdb2*F2
	Degradation_F5_Cdc6		'F5'	'Cdc6'	Vdb5*F5
	Degradation_F2P_Clb2		'F2P'	'Clb2'	kd3f6*F2P
	Degradation_F5P_Clb5		'F5P'	'Clb5'	kd3f6*F5P
ChrXVI --> ChrXVI_L [**Cln2**] ChrXVI_M1 **Clb2**	Degradation_F2P_Cdc6P		'F2P'	'Cdc6P'	Vdb2*F2P
ChrXVI_M2 [**Clb5**] ChrXVI_R	Degradation_F5P_Cdc6P		'F5P'	'Cdc6P'	Vdb5*F5P

Rules

Rules

Rules	Var	Math
Genome --> InitialCondition Model (ChrI) (ChrII) (ChrIII) (ChrIV) (ChrV) (ChrVI) (ChrVII) (ChrVIII) (ChrIX) (ChrX) (ChrXI) (ChrXII) (ChrXIII) (ChrXIV) (ChrXV) (ChrXVI)	**mu**	ln(2)/MDT
	D	1.026/mu-32
	F	exp((-mu)*D)
	IE	IET-IEP
	Vdppx	kpdppx+kppdppx*(J20ppx+Cdc20)*Jpds/(Jpds+Pds1)
ChrI --> ChrI_L [**Cln3**] ChrI_M1 [**Lte1**] ChrI_M2 [**Cdc15**] ChrI_R	**Cdc15i**	Cdc15T-Cdc15
	Cln3	C0*Dn3*mass/(Jn3+Dn3*mass)
ChrIV --> ChrIV_L [**Pds1**] ChrIV_M [**Swi5**] ChrIV_R	**Vdpds**	kpd1pds+kppd2pds*Cdc20+kppd3pds*Cdh1
ChrV --> ChrV_L [SBF] CHrV_M **Bck2** ChrV_R	**Bck2**	B0*mass
	Vasbf	kasbf*(Esbfn2*Cln2+Esbfn3*(Cln3+Bck2)+Esbfb5*Clb5)
	SBF	2*Jisbf*Vasbf/((kpisbf+kppisbf*Clb2)-Vasbf+Jasbf*(kpisbf+kppisbf*Clb2)+Jisbf*Vasbf+sqrt(sqr((kpisbf+kppisbf*Clb2)-Vasbf+Jasbf*(kpisbf+kppisbf*Clb2)+Jisbf*Vasbf)-4*((kpisbf+kppisbf*Clb2)-Vasbf)*Jisbf*Vasbf))
ChrVI --> ChrVI_L [**Cdc14**] ChrVI_R	**Cdc14T**	Cdc14+RENT+RENTP
ChrVII --> ChrVII_L **Cdc20** ChrVII_M1 [**Cdh1**] ChrVII_M2 [**Esp1**] ChrVII_R	**Vacdh**	kpacdh+kppacdh*Cdc14
	Vicdh	kpicdh+kppicdh*(Ecdhn3*Cln3+Ecdhn2*Cln2+Ecdhb2*Clb2+Ecdhb5*Clb5)
	PE	Esp1T-Esp1
ChrX --> ChrX_L **Cdc6** ChrX_M1 **Net1** ChrX_M2 **Mad2** ChrX_R	**Vkpf6**	kd1f6+Vd2f6/(Jd2f6+Cdc6+F2+F5+Cdc6P+F2P+F5P)
	Vppf6	kppf6*Cdc14
	Vd2f6	kd2f6*(Ef6n3*Cln3+Ef6k2*Bck2+Ef6n2*Cln2+Ef6b5*Clb5+Ef6b2*Clb2)
	Cdc6T	Cdc6+Cdc6P+F2+F5+F2P+F5P
	Net1T	Net1+Net1P+RENT+RENTP
	Vppnet	kpppnet+kpppnet*PPX
	Vkpnet	(kpkpnet+kppkpnet*Cdc15)*mass
ChrXI --> ChrXI_L [APC] ChrXI_R	**CKIT**	Sic1T+Cdc6T
ChrXII --> ChrXII_L **Sic1** ChrXII_R	**Vd2c1**	kd2c1*(Ec1n3*Cln3+Ec1k2*Bck2+Ec1n2*Cln2+Ec1b5*Clb5+Ec1b2*Clb2)
	Vkpc1	kd1c1+Vd2c1/(Jd2c1+Sic1+C2+C5+Sic1P+C2P+C5P)
	Vppc1	kppc1*Cdc14
	Sic1T	Sic1+Sic1P+C2+C5+C2P+C5P
ChrXIII --> ChrXIII_L [**Tem1**] ChrXIII_M1 **Mcm1** ChrXIII_M2 [**Bub2**] ChrXIII_R	**Mcm1**	2*Jimcm*kamcm*Clb2/(kimcm-kamcm*Clb2+Jamcm*kimcm+Jimcm*kamcm*Clb2+sqrt(sqr(kimcm-kamcm*Clb2+Jamcm*kimcm+Jimcm*kamcm*Clb2)-4*(kimcm-kamcm*Clb2)*kimcm*kamcm*Clb2))
	Tem1GDP	Tem1T-Tem1GTP
ChrXVI --> ChrXVI_L [**Cln2**] ChrXVI_M1 **Clb2** ChrXVI_M2 [**Clb5**] ChrXVI_R	**Vdb5**	kpdb5+kppdb5*Cdc20
	Vdb2	kpdb2+kppdb2*Cdh1+kdb2p*Cdc20
	Clb5T	Clb5+C5+C5P+F5+F5P
	Clb2T	Clb2+C2+C2P+F2+F2P
	Visbf	kpisbf+kppisbf*Clb2
	Vaiep	kaiep*Clb2

Events

Events

Rules	Name	Event_ass	Trigger_math
Genome --> InitialCondition Model (ChrI) (ChrII) (ChrIII) (ChrIV) (ChrV) (ChrVI) (ChrVII) (ChrVIII) (ChrIX) (ChrX) (ChrXI) (ChrXII) (ChrXIII) (ChrXIV) (ChrXV) (ChrXVI)	reset_ORI	['ORI','"0"']	lt(Clb2+Clb5-Kez2,0)
	start_dna_synthesis	['Mad2','"mad2h"'],['Bub2','"bub2h"']	gt(ORI-1,0)
	spindle_checkpoint	['Mad2','"mad2l"'],['Lte1','"lte1h"'],['Bub2','"bub2l"']	gt(SPN-1,0)
	cell_division	['mass','"exp(-(ln(2)/MDT)*((MDT*1.026/ln(2))-32))*mass"'],['Lte1','"lte1l"'],['BUD','"0"'],['SPN','"0"']	lt(Clb2-Kez,0)

API to Design Genotype-to-Phenotype Languages

Prolog keynotes

Prolog is a logical programming language. It was created in 1972 by Alain Colmerauer (ENSIMAG alumni) with Philippe Roussel.
Prolog's simplest data type is the term which might either be atom, number, variable or compound term. The variables are denoted by a string beginning with an upper-case letter or underscore; they cannot be re-assigned. A compound term is composed of an atom called a "functor" and a number of "arguments", which are terms, separated by commas which are contained in parentheses after the "functor".
A List is an ordered collection of terms. It is denoted by square brackets with the terms separated by commas or in the case of the empty list, [].
Prolog programs describe relations, defined by means of clauses. There are two types of clauses: facts and rules. A rule is of the form "Head: - Body." and is read as "Head is true if Body is true".
A rule's body consists of calls to predicates. The built-in predicate ,/2 denotes conjunction of goals, and ;/2 denotes disjunction in the body of a rule only. Clauses with empty bodies are called facts; they are true by definition.
The execution of a Prolog program is initiated by the user's posting of a single goal, called the query. Logically, the Prolog engine tries to find a resolution refutation of the negated query.
I used the SWI-Prolog distribution for my project and particularly interesting for this project, SWI-Prolog offers the Definite Clause Grammar (DCG) syntax for rules definition. Grammar rules look like ordinary clauses using $--> /2$ for separating the head and body rather than $:-/2$.
Code between {...} is interpreted as ordinary Prolog code and finally, a list is interpreted as a sequence of literals.
Grammar rule-sets are called using the built-in predicates *phrase*/2 and *phrase*/3:

```
''phrase(+RuleSet, +InputList, -Rest)''
or
simply ''phrase(+RuleSet, +InputList)''
```

Prolog handles backtracking and finds all the possible interpretations which is convenient if a grammar is ambiguous by leaving the decision up to the final user.

Generals:

- Variable name, in Prolog, must start with a capital letter.
- Functions start with a un-capitalized letter

- A comma separates each function.
- It 'sections of code', series of functions, should terminate by a dot.
- A comment for an entire line of code is preceded by %%
- To make an SBML model, with Trans interactions, all categories have at least these attributes: a string, Construct and lists: Species, Parameters, Reactions, Events, Trans. These must be added through the interface.
- Code init is a piece of prolog code executed before parsing the design. Here it is: init_sbml, which will declare the java class and declare the units and the compartment, the level of SBML targeted (currently level2version4) etc. Those elements are grammar specific (but can be reused between grammars), but they are not specific of the design. I recommend keeping the compartment to cell. The reason is that the SBML API has the compartment cell hard-coded within Species declarations for example.
- Code end is going to look at the list that have been composed and make the declaration calls to write the corresponding java code. It reduces list_to_set to avoid duplicate (be careful though, you may want to know if a reaction is declared twice because you may want to change the init val of the reactant accordingly), and then just exec the call. Trans is pretreated by looking at Species and compute the call according to the Species found in the design. Finally, the order matters (A specie declaration may use a parameters value for example, which therefore must be declared first), hence it's probably better to start with treating Parameters. There is no comma at the beginning and no dot at the end.

The lists are the categories of the Start. In this case, they are denoted as S1, S2, S3,etc. depending on the order you added them through the interface. Use that designation! Look at API to choose a function to analyze the TRANS, use process_list for others.

Example:

list_to_set(S2,Ss),list_to_set(S3,Ps),list_to_set(S5,Es),list_to_set(S6,Ts),
nl,nl,process_list(Ps),nl,process_list(Ss),nl,list_to_set(S4,Rs),
process_list(Rs),nl,process_list(Es),nl,analyze_trans(Ts,Ss),nl,end_sbml

We get (as we automatically call check_grammar(< *list_of_parts* >), where c117413310 is the start category with attributes: Construct, Species, Parameters, Reactions, Events, Trans):

check_grammar(Tokens) :- init_sbml,c117413310(S1, S2, S3, S4, S5, S6, Tokens, []) ,
list_to_set(S2,Ss),list_to_set(S3,Ps),list_to_set(S5,Es),list_to_set(S6,Ts),
nl,nl,process_list(Ps),nl,process_list(Ss),nl,list_to_set(S4,Rs),
process_list(Rs),nl,process_list(Es),nl,analyze_trans(Ts,Ss),nl,end_sbml.

Note for Coders

- In the compiler generated, Categories are designated by $'c' + category\_id$ (category are in fact functors in prolog), Attribute by $'A' + attribute\_id$ and parts by $'p' + part\_id$
- The rules notation is DCG and semantic action goes between {}

Example:

```
%PCI-->[pcI]
c117413329('pcI', 'pcI', [], [parameter('s_pcI',8), parameter('w0_pcI',0.5)],
    [], [], [])
--> [p21927].
c117413318(A2671, A2672, A2673, A2674, A2675, A2761) -->
c117413329(Name117413329, A2741, A2742, A2743, A2744, A2745, A2775), {
%% < will write the semantic action
%%fetched for this rule >: write some prolog code using API
}.
```

First pre-populate write the semantic actions This step is used to pass the information automatically up in the tree. Hence, do for each parent rules:

- Concat the names on the right to form the attribute name on the left

  atomic_list_concat([name_right_1,",name_right_2, _etc.],name_left)

- Append for each type of list attribute, the ones from right to the corresponding attribute of the ones of the left.

  append([the_ones_on_right],the_one_on_the_left)

Examples:
Here the rule is $CAS \rightarrow PBS, CDS, TER$,
say all have categories have an attribute Construct and
an attribute-list Events; hence, in the box semantic action write:

atomic_list_concat([∧A-Construct(PBS)∧,
'_',∧A-Construct(CDS)∧,'_',∧A-Construct(TER)∧],
∧A-Construct(CAS)∧),
append([∧A-Events(TER)∧,
∧A-Events(CDS)∧,∧A-Events(PBS)∧],
∧A-Events(CAS)∧)

Add your model semantic actions. These are declarations using the swi-prolog library that writes a java-libsbml declaration for your SBML objects. The API is given in the next section.

In order to write the variable or species name, you may have to concat strings and variable Hence, use atomic_list_concat.

Example:

Here Name_PROT will be 'PROT_$< name_CDS >$' and InitCond will be
'init_$< name_CDS >$.getValue()', which in this case, 'init_$< name_CDS >$'
correspond to a parameter previously declared (for ex. parameter('init_lacI',1E-21)).
atomic_list_concat(['PROT_',∧A-Construct(CDS)∧],Name_PROT), atomic_list_concat(['init_',∧A-Construct(CDS)∧],Init_PROT), atomic_list_concat([Init_PROT,'.getValue()'],InitCond)

Then add your function(s) –separated by commas– between brackets, from the API to the concatenation of lists: Example: Here, the categories all have attribute-list Species. In the semantic action of the
rule $CAS \rightarrow PBS, CDS, TER$ we want to declare the existence of the
species protein that can be synthesized from the CDS since is precede by a PBS.
append([[species(Name_PROT,InitCond)], ∧A-Species(TER)∧,
∧A-Species(CDS)∧, ∧A-Species(PBS)∧],
∧A-Species(CAS)∧)

Add Parts attribute values Use the function declarations, but here they are synthesized, hence they do not depend on other variables so everything must be hard coded.
Example: In the category MDL (for model) the part init_switch has:
Construct: 'init_switch'
Rules: []
Species: []
Parameters: [parameter('init_LAC_lacI', 1E-21),
parameter('init_TET_TetR', 5E-21),
parameter('init_CI_cits', 0)]
Reactions: []
Events: [event('switching',[['PROT_LAC_lacI', 5E-21],
PROT_TET_TetR', 1E-21
], "'eq(time,20)"')]
Trans: []

API Prolog to SBML via java-libsbml

Events **event(Name, Event_ass, Trigger_math)** Where:

- Name is the name of the event, unique for the AG, between ' and ', and will be it's id

  no space in the name

- Event_ass is a list of event assignments (hence between brackets) where each of them is of the form [Variable, Value]. Variable is a name hence between ' and '.

  Params as var should not be constant

- Trigger_math is the mathematical expression for the trigger as a string (with " and "), and then between quotes ' and '. I used libsbml.parseL3Formula.

Example:

With values: event('switching', [['PROT_LAC_lacI', 5E-21], ['PROT_TET_TetR', 1E-21]], "'eq(time,20)"')

gives:

```
<event id=''switching'' useValuesFromTriggerTime=''true''>
       <trigger>
         <math xmlns=''http://www.w3.org/1998/Math/MathML''>
           <apply>
             <eq/>
             <csymbol encoding=''text'' definitionURL=
             ''http://www.sbml.org/sbml/symbols/time''> time </csymbol>
             <cn type=''integer''> 20 </cn>
           </apply>
         </math>
       </trigger>
       <listOfEventAssignments>
         <eventAssignment variable=''PROT\_LAC\_lacI''>
           <math xmlns=''http://www.w3.org/1998/Math/MathML''>
             <cn type=''e-notation''> 5 <sep/> -21 </cn>
           </math>
         </eventAssignment>
         <eventAssignment variable=''PROT\_TET\_TetR''>
           <math xmlns=''http://www.w3.org/1998/Math/MathML''>
             <cn type=''e-notation''> 1 <sep/> -21 </cn>
           </math>
         </eventAssignment>
       </listOfEventAssignments>
   </event>
```

or with expressions and parameters in assignments
event_exp('cell_division', [['mass',' " exp(-(ln(2)/MDT)*((ln(2)/MDT)-32))*mass " '], ['Lte1',' " let1l " '], ['BUD',' " 0 " '], ['SPN',' " 0 " ']],' " lt(Clb2-Kez,0) " ')

Species species(Name, InitConc)

- Name will also be id must be between ' and '
- Initial concentration is the value passed as argument
- We have Compartment("cell"), Constant(false), setBoundaryCondition(false), HasOnlySubstanceUnits(true)
- The SpeciesReference is also declared, same name precede by ref_ and it is not constant

species_amount(Name, InitAmount)

- Name will also be id must be between ' and '
- Initial amount (in the substance unit, by default mole) is the value passed as argument
- We have Compartment("cell"), Constant(false), setBoundaryCondition(false), HasOnlySubstanceUnits(true)
- The SpeciesReference is also declared, same name precede by ref_ and it is not constant

species(Name)

- Name will also be id must be between ' and '
- We have Compartment("cell"), Constant(false), setBoundaryCondition(true), HasOnlySubstanceUnits(true)
- The SpeciesReference is also declared, same name precede by ref_ and it is not constant

Example:

species('PROT_LAC_lacI',1E-21)

```
<species id=''PROT_LAC_lacI'' name=''PROT_LAC_lacI'' compartment=''cell''
initialConcentration=''1e-21'' hasOnlySubstanceUnits=''true''
boundaryCondition=''false'' constant=''false''/>
```

or

species('PROT_LAC_lacI','init_ PROT_LAC_lacI')

Parameters parameter(Name, Value)

- Name will also be id must be between ' and '
- It is constant and the value is the one given by Value

parameter_notconstant(Name, Value)

- Name will also be id must be between ' and '
- It is not constant and it's initial value is given by Value

parameter(Name)

- Name will also be id must be between ' and '
- It is not constant and there is no initial value.

Example: parameter('init_LAC_lacI',1E-21)

gives :

```
<parameter id=''init\_LAC\_lacI'' value=''1e-21'' constant=''true''/>
```

Reactions reaction(Name, Modifiers, Reactants, Products, Math) Where:

- Name is between ' ' unique for the AG, and will also be its id
- Modifiers is a list (can be empty i.e. [])
- Reactants is a list (can be empty i.e. [])
- Products is a list (can be empty i.e. [])
- Math is the mathematical expression for the trigger as a string (with " and "), and then between quotes ' and '. I used libsbml.parseFormula.

reaction_rev(Name, Modifiers, Reactants, Products, Math) this is the same as reaction but it is reversible

Example:

reaction('Growth',[],[],['mass'],' "kg*mass" ')

```
<reaction id= '' Growth '' name= '' Growth '' reversible= '' false ''
fast= '' false '' >
       <listOfProducts>
         <speciesReference species= '' mass '' />
       </listOfProducts>
       <kineticLaw>
         <math xmlns= '' http://www.w3.org/1998/Math/MathML '' >
           <apply>
             <times/>
             <ci> kg </ci>
             <ci> mass </ci>
           </apply>
         </math>
       </kineticLaw>
     </reaction>
```

Rules rule(Var, Math)

- Name is between ' ' unique for the AG, and will also be its id
- it is an assignment rule, for the variable Var
- Math is the mathematical expression for the trigger as a string (with " and "), and then between quotes ' and '. I used libsbml.parseFormula.

Example:

rule('Bck2',' " B0*mass " ')

gives:

```
<assignmentRule variable= '' Bck2 '' >
       <math xmlns= '' http://www.w3.org/1998/Math/MathML '' >
         <apply>
           <times/>
           <ci> B0 </ci>
           <ci> mass </ci>
         </apply>
       </math>
     </assignmentRule>
```

Trans library

In the part name of sequence that has a trans function (such as protein for lacI coding sequence is 'LAC'), insert the keyword desired in the name (i.e. 'PROT_LAC_lacI'). Then when needed in semantic actions, place in the attribute trans declaration from the swi-prolog library described. But the element may refer to trans. Use the specific TRANS library. It will be matching corresponding species declarations and write the corresponding prolog code that will generate the object declaration desired depending on the species present in the system analyzed. Hence trans is always analyzed first.

TRANS

```
<something-before>TRANS{[Type-Key], trans_math_exp, basal_math_exp}
    <something-before>
```

The program is going to cut off the TRANS{} section which has been used within a standard reaction declaration for the mathematical expression and replace it with the mathematical expression. The mathematical expression is computed by: searching all species in the species list that contains the Type and the Key.
Then in the trans_math_exp, for each species found, replacing the keyword "Type-Key" in the expression by the species and concatenate all the resulting strings. hence, the string should start with the + or -. Finally, the basal expression is added (that way if there is not trans species found it is still valid).

TRANSspecies_and_declare

```
<something-before>TRANSspecies_and_declare{[Type-Key], name, InitConc}
    <something-before>
```

TRANSspecies

```
TRANSspecies{Type-Key}
```

list all species as a list that matches (to be used in list of products in a reaction for example)

TRANSfor_each_species

```
TRANSfor_each_species{Type-Key, expression}
```

the type-key will be replace in expression. each match correspond to a new expression, at the end they are separated by a comma.

API Prolog to GinML

code_init initginml

```
<?xml version= '' 1.0 '' encoding= '' UTF-8 '' ?>
<!DOCTYPE gxl SYSTEM '' http://gin.univ-mrs.fr/GINsim/GINML_2_1.dtd '' >
<gxl xmlns:xlink= '' http://www.w3.org/1999/xlink '' >
```

THEN PROCESS write_graph(nodes)

```
<graph id= '' default_name '' class= '' regulatory '' nodeorder=
```

and write the nodes in the order

```
>
```

then use LIBRARY: node(Name, MinValue, MaxValue)

```
<node id= '' A '' basevalue= '' 0 '' maxvalue= '' 2 '' >
</node>
```

edge(From,To,Sign,Val)

```
<edge id= '' A_B_0 '' from= '' A '' to= '' B ''
minvalue= '' 2 '' sign= '' negative '' >
</edge>
```

code_end

```
</graph>
</gxl
```