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, but which 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. We then 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. We then 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 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 demonstrate 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.

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

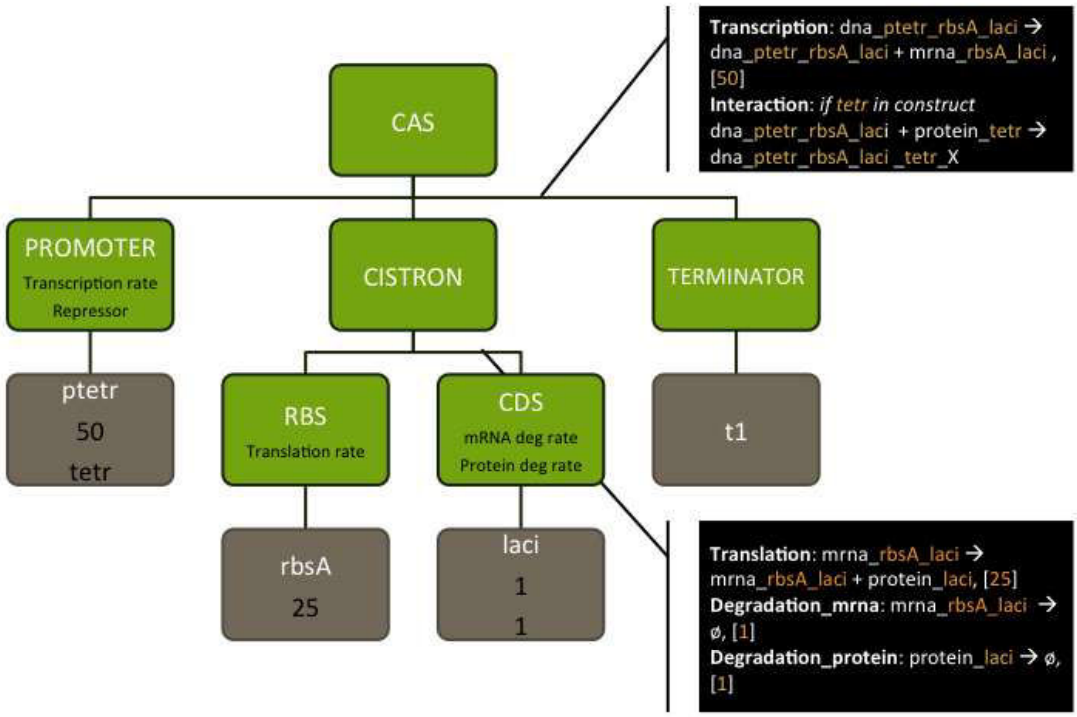


Figure 6.1: Tree based on an Attribute Grammar for Biological Language. This tree is a simple example of a genetic cassette. The start symbols is 'CAS'. It shows how the semantic elements of an attribute grammar can be used to generate the equations that correspond to the example construct. The nonterminal, 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.

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. The 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-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 mechanism 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

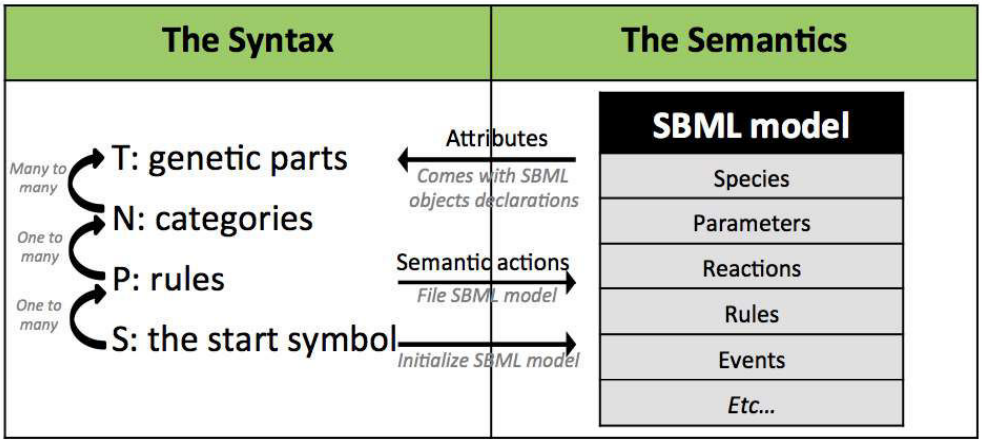


Figure 6.2: Attribute Grammar for Predictive Biological Language. We used the SBML model example to explain the logic of developing an Attribute Grammar. The syntax describes the genetic sequences that can be made by narrowing the possibilities of the spacial structure of genetic elements and the semantics 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.

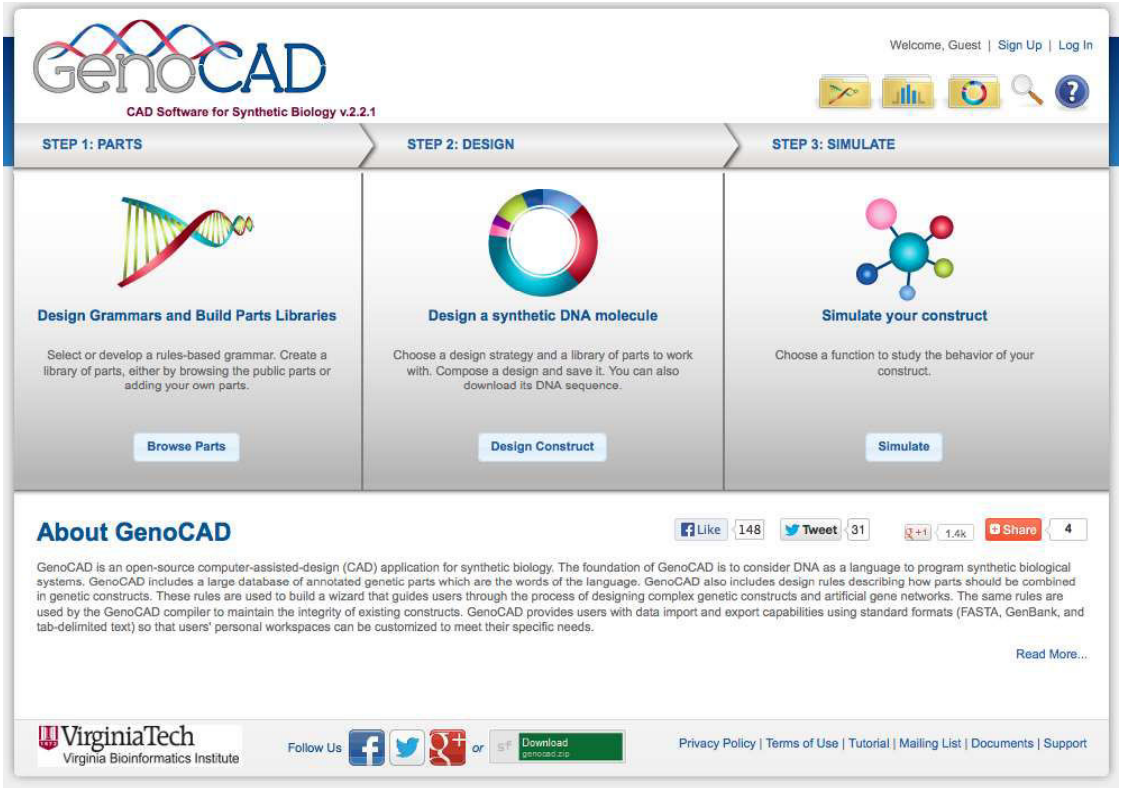


Figure 6.3: GenoCAD Workflow. Using GenoCAD from an end-user perspective is a three-step process. In Step 1, a grammar can be defined to solve a particular problem or organism. Then genetic parts are added and organized as project libraries and grammar rules can be defined. In Step 2, following grammar 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).

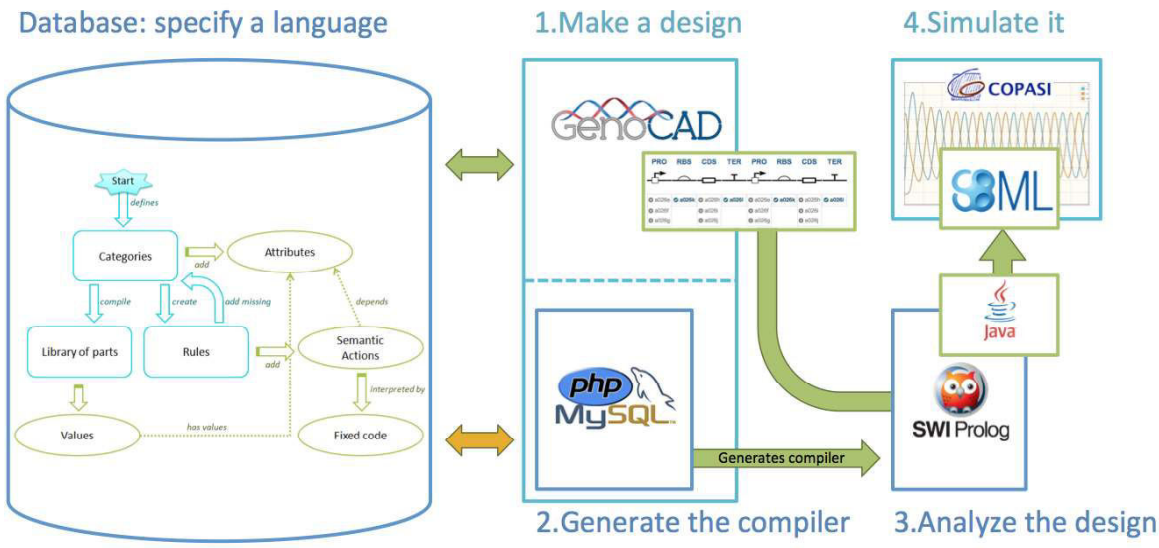


Figure 6.4: Behind the GenoCAD Workflow. Compilation of a design occurs after the on-the-fly generation of a semantic DNA compiler based on the user's attribute grammar as stored in the database. The compiler generated is used to compile the user's design and the SBML target file computed is returned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

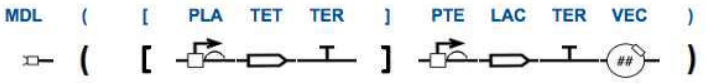
The semantic layer of the example Attribute Grammar specifies how the gene expression is regulated. In a biochemical network, the transcriptional regulation happens through RNA and protein products that have been expressed. These products then regulate the transcription of DNA sequences influencing the 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.

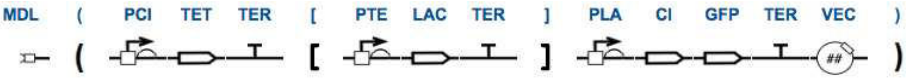
In order to test the example grammar and the generation of the compiler, we examined a genetic toggle switch and an oscillator [Elowitz and Leibler 2000; Gardner, Cantor, and Collins 2000]. We searched for parameters that satisfied the dynamic constraints of the two examples in reusing parts and the 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`.



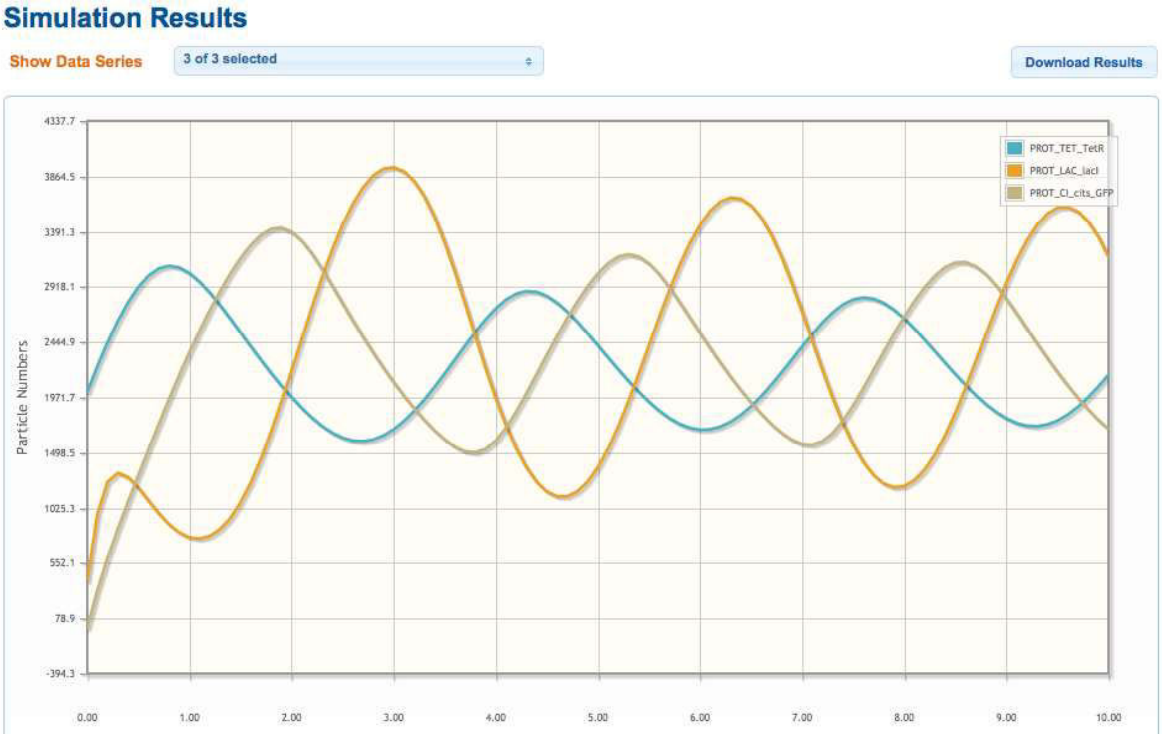
(a) Switch design

(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 control of pBAD, the plasmid with the chaperone under the control of pTet, and the output plasmid in which the core promoter to which the complex binds control 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.

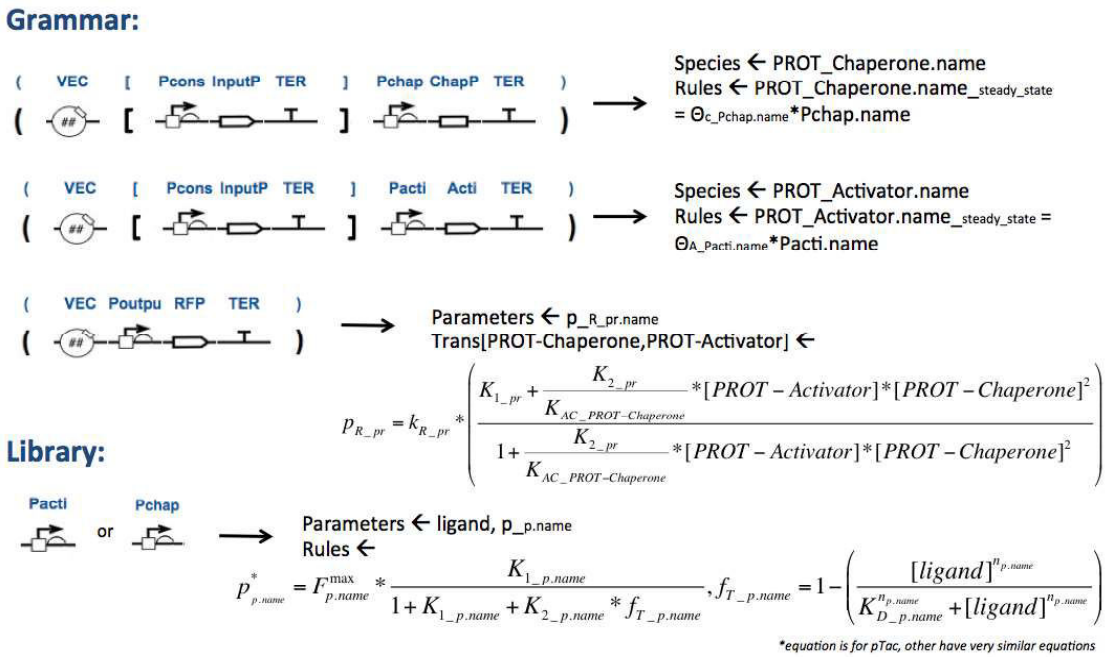


Figure 6.7: AND gate structure and its semantics. The construct layout of an AND gate is made out of three plasmids. The first one carries the Chaperone protein. The semantic action of this plasmid rule declares the protein species using the 'PROT' and 'Chaperone' keywords in the name. The level of expression 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.

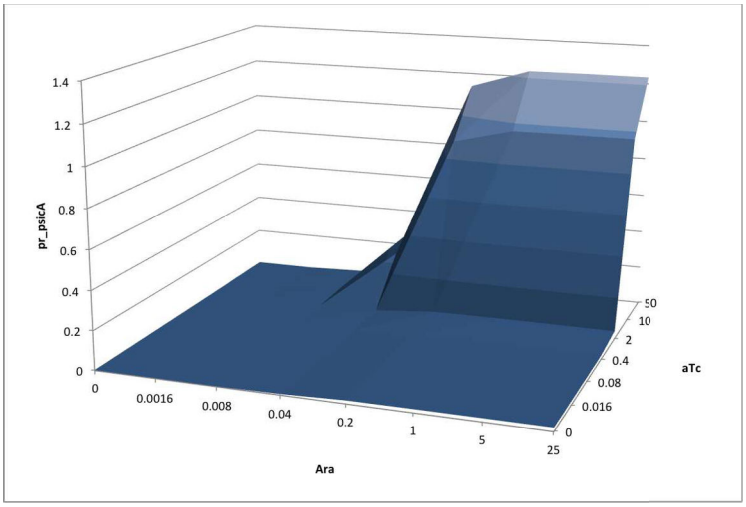


Figure 6.8: Computed transfer function of psicA in the design of the AND gate with the *Salmonella* parts library. We designed an AND gate with the AND gate language described previously and the *Salmonella* parts. We obtained the SBML file from GenoCAD thanks to the generated compiler. We imported it 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.

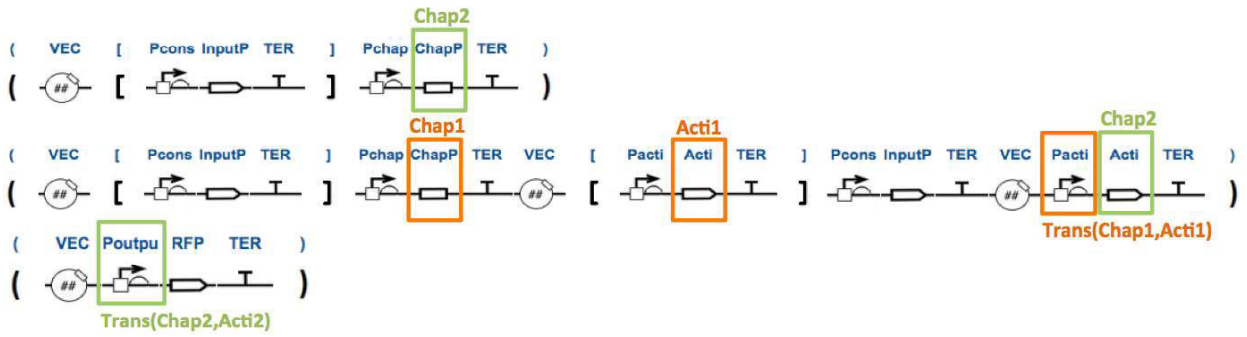


Figure 6.9: Concept to layer AND gates. The 3 input AND gate is design on three different plasmids. The semantic actions that compute the values of the promoters repressed by the transcription factors and chaperones uses different keywords to impose the layered AND gate layout.

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 download at `http://web.figshare.com/download/`

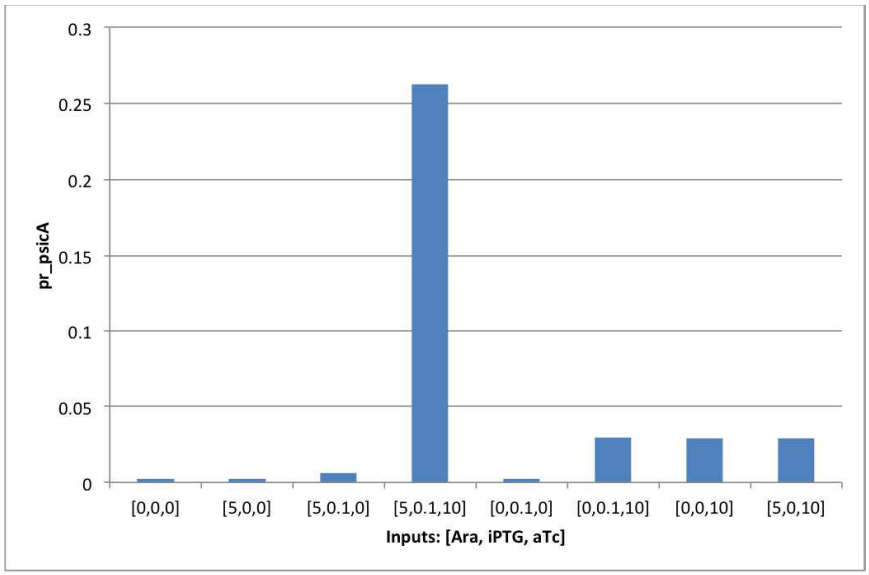


Figure 6.11: Verification of the 3-input AND gate. We used the SBML model generated for the 3-input AND gate design in GenoCAD by our compiler to compute pr_psicA for all possible input values and verified the 3-input AND gate behavior. We see that the value of pr_psciA is high only in the case that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

`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.

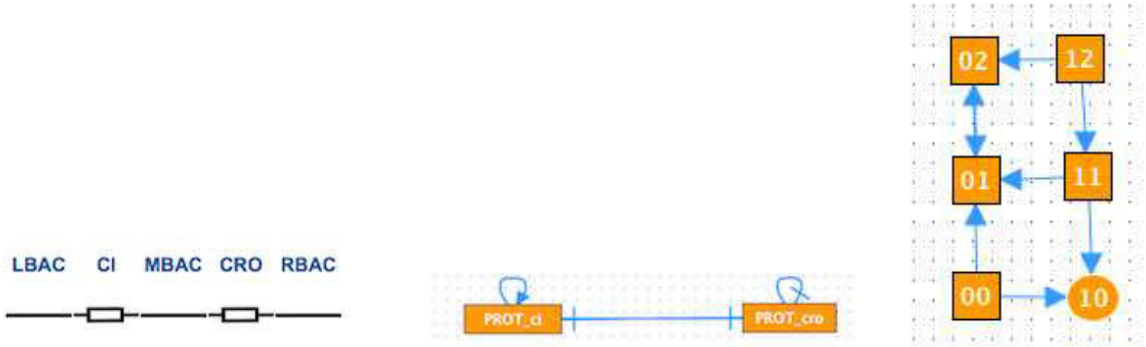
In a discrete gene regulatory network (dGRN), nodes, the proteins of the system designed, are related by arrows, which represent interactions (direct or indirect between the proteins). The sign of the arrow, negative or positive, respectively indicate repression or activation. The concentration of proteins 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: Design of the lambda bacteriophage genome 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.