BIOINFORMATICS ORIGINAL PAPER Vol. 23 no. 20 2007, pages 2760–2767
doi:10.1093/bioinformatics/btm446

Systems biology

A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts

Yizhi Cai[1], Brian Hartnett[1], Claes Gustafsson[2] and Jean Peccoud[1,*]

[1]Virginia Bioinformatics Institute, Virginia Polytechnic Institute and State University, Washington Street, MC 0477, Blacksburg VA 24061, USA and [2]DNA2.0 Inc. 1430 O'Brien Drive Suite E, Menlo Park CA 94025, USA

Received on July 3, 2007; revised on August 20, 2007; accepted on August 21, 2007

Advance Access publication September 5, 2007

Associate Editor: John Quackenbush

ABSTRACT

Motivation: The sequence of artificial genetic constructs is composed of multiple functional fragments, or genetic parts, involved in different molecular steps of gene expression mechanisms. Biologists have deciphered structural rules that the design of genetic constructs needs to follow in order to ensure a successful completion of the gene expression process, but these rules have not been formalized, making it challenging for non-specialists to benefit from the recent progress in gene synthesis.

Results: We show that context-free grammars (CFG) can formalize these design principles. This approach provides a path to organizing libraries of genetic parts according to their biological functions, which correspond to the syntactic categories of the CFG. It also provides a framework for the systematic design of new genetic constructs consistent with the design principles expressed in the CFG. Using parsing algorithms, this syntactic model enables the verification of existing constructs. We illustrate these possibilities by describing a CFG that generates the most common architectures of genetic constructs in *Escherichia coli*.

Availability: A web site allows readers to experiment with the algorithms presented in this article: www.genocad.org

Contact: peccoud@vt.edu

Supplementary information: Sequences and models are available at Bioinformatics online.

1 INTRODUCTION

Gene synthesis technology now enables molecular biologists to assemble long DNA molecules that may include multiple genes and their regulatory sequences. We will refer to these molecules as 'genetic constructs' or just 'constructs'. As the throughput of construct manufacturing increases, the design of complex genetic constructs becomes the bottleneck of the process. It becomes easier to assemble complex DNA molecules than to design them. A natural way of designing complex constructs involves combining basic building blocks also known as 'biological parts' or 'genetic parts' (Benner and Sismour, 2005; Heinemann and Panke, 2006; Voigt, 2006). These parts are small DNA fragments implementing specific biological functions. The mechanisms of gene expression require that certain structural constraints are met in order for a construct to be functional. Parts of different types need to be placed in a particular order and next to each other in order to ensure that coding sequences are properly transcribed and translated. Certain parts are functional only in a specific context whereas other parts have proved functional in organisms other than the one from which they originate. For instance, promoters are often restricted to specific organisms or even cell types (Cavin Perier *et al*., 1998; Munch *et al*., 2003; Zhu and Zhang, 1999) whereas genes coding for proteins can often be expressed in multiple species (Goeddel *et al*., 1979). The design of complex genetic constructs such as artificial gene networks (Chin, 2006; Elowitz and Leibler, 2000; Gardner *et al*., 2000; Guet *et al*., 2002; Guido *et al*., 2006; Heinemann and Panke, 2006; Kobayashi *et al*., 2004; McDaniel and Weiss, 2005) therefore requires an intimate knowledge of gene expression mechanisms. It is interesting to observe that more than 6 years after the description of the first artificial gene networks (Elowitz and Leibler, 2000; Gardner *et al*., 2000), this technology has yet to find biomedical applications. It is likely that most biologists who could use sophisticated genetic constructs to control the expression of their gene of interest do not have the expertise to design the construct they need. One way to lower the barrier to entry into synthetic biology is to formalize the structural constraints associated with the use of standardized biological parts in a construct. Such formalism can be used to build software wizards to guide users in the design of their constructs. It can also provide a foundation to the development of parsers capable of verifying the structural validity of a synthetic DNA sequence.

Several prominent synthetic biologists have advocated an engineering approach to the design of genetic constructs (Baker *et al*., 2006; Endy, 2005) well illustrated by the Registry of Standard Biological Parts, a service provided by MIT to promote the development and dissemination of well-specified, standardized and interchangeable biological parts. The records in this database are organized in different categories corresponding to different levels of abstraction (Endy, 2005). At the bottom of this hierarchy lay the basic parts. Parts can be combined in functional modules called devices. Devices and parts can ultimately be combined in self-contained systems. The 'Parts' category is itself divided into subcategories (Regulatory, Terminators, RNA, DNA, Protein Coding,

[*]To whom correspondence should be addressed.

© 2007 The Author(s)
This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/2.0/uk/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Ribosome Binding Sites and Conjugation) corresponding to biological functions. The database enables users to create new records by combining existing records corresponding to basic parts, devices or construction intermediates. Standardized graphical representation of complex records makes it easy to visualize their structure. After examining a number of records, it is possible to identify common features shared by many entries. However, the record editing process is unconstrained; no structural rule is imposed on new records nor are the records automatically verified upon submission.

The development of Gene Designer (Villalobos *et al*., 2006), a software application to quickly design synthetic DNA molecules from a library of basic parts, has been inspired by a similar vision. The user interface includes a standard library of parts called the Design Toolbox. Its hierarchical organization is multilayered to accommodate sequences specific to multiple biological species and a broader spectrum of biological functions than in the MIT Registry. Gene Designer makes it very easy to drag elements of the toolbox into new DNA sequences. The structure of complex sequences combining multiple parts is represented by an icon view. Gene Designer does not provide a wizard to guide the user in the design of a construct nor does it have a feature to verify the structural validity of constructs.

In mathematics, logic and computer science, a formal language is a language that is defined by precise mathematical or machine processable formulas. Like natural languages, these formal languages generally have two aspects. The syntax of a language is what the language looks like (more formally: the set of possible expressions that are valid utterances in the language). The semantics of a language are what the utterances of the language mean. The syntax or grammar of the language can be formally defined by the specification of a set of non-terminal symbols or variables, a set of terminal symbols and a set of production rules also called transformation rules. The variables represent categories of words such a nouns and verbs; they are often referred to as syntactic categories. The terminals represent actual words such as 'dog' or 'sing'. The production rules map one string of symbols to another, where the first string contains at least one non-terminal symbol. The recursive application of production rules, beginning from the start variable often denoted S, generates the set of strings containing only terminal symbols, which is the language generated by the grammar in which every production rule is of the form $V \rightarrow w$, where V is a single non-terminal symbol and w 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.

We have developed a context-free grammar that formalizes the structure of a library of previously published artificial genetic constructs, which are derived by combining standard genetic parts. We show how this syntactic model provides a rigorous foundation for the organization of a parts library in syntactic categories defined according to the structural constraints affecting the position of parts in genetic constructs. In addition, we show how this model enables a systematic approach to the design of genetic constructs that can be implemented in software. Last, this model can be used to build parsers capable of accepting constructs consistent with the design principles captured by the CFG production rules.

Early applications of linguistic models in the analysis of biological sequences have been reviewed in an article that also provides a short introduction to these types of models (Searls, 2002). Most of these early works were attempting to analyze naturally occurring sequences. Grammars were developed with the goal of understanding genome structures (Brendel and Busse, 1984; Brendel *et al*., 1986) and associating genes with their regulatory sequences (Collado-Vides, 1992). Another body of work focused on predicting the secondary structures of RNA molecules (Knudsen and Hein, 1999, 2003; Matsui *et al*., 2005; Rivas and Eddy, 2000; Sakakibara *et al*., 1994). The discovery of grammatical models from sets of curated biological sequences remains a very active field of research in the machine-learning community (Sakakibara, 2005). Linguistic models have also been used to analyze proteins with different purposes. Most of the work in this field attempts to understand the rules of protein organization in modular domains (Gimona, 2006), but recently grammatical models have been developed with the goal of designing new antimicrobial peptides (Loose *et al*., 2006). This work proceeded in two steps. In order to decipher the design principles of natural antimicrobial peptides, a set of grammars was inferred from natural sequences using a pattern discovery algorithm (Rigoutsos and Floratos, 1998). In a second step, 42 peptides consistent with the discovered grammars but not homologous to natural peptides were synthesized and tested. Approximately half of the new peptides exhibited an antimicrobial activity, which demonstrates the power of this approach. In the context of this article, we have also used formal grammars to support the design of new DNA sequences, which is a very different goal from the analysis of natural genomic sequences. Instead of inferring the production rules from a training data set, our production rules utilize preexisting biological knowledge relative to the structure of functional genetic constructs.

2 METHODS

2.1 Variables

The first step in the construction of the grammar is to recognize syntactic categories in categories used to organize genetic parts. These syntactic categories are represented by the CFG variables listed in Table 1. The specific CFG presented in this article has only 26 syntactic categories each represented by a single capital letter. The orientation of constructs can be left to right or right to left depending on which DNA strand is transcribed. If left to right is the direct orientation and right to left the reverse orientation, each category of genetic parts needs to be broken down into two syntactic categories corresponding to the direct and reverse orientations as different structural rules apply to each.

Variables have been organized in four hierarchical categories. The first category contains only S, the start variable from which all derivations are initiated. The second category corresponds to complex fragments of DNA composed of multiple functional parts. This category includes the variables M and N which correspond to transcripts in the forward and reverse orientation, respectively. In the context of this article, a transcript is a DNA fragment located between a promoter and a transcription terminator. Also in this category are the

Table 1. Variable set

Name	Description	Category
S	Start	I
E	Gene coding region	II
F	Gene coding region reverse	
M	Transcript	
N	Transcript reverse	
G	Terminator	III
H	Terminator reverse	
O	Linker	
R	T7 terminator	
T	T7 terminator reverse	
U	Protein domain	
V	Protein domain reverse	
Y	Stop codon	
Z	Stop codon reverse	
A	Promoter	IV
B	Promoter reverse	
C	Ribosome binding site	
D	Ribosome binding site reverse	
I	Riboregulator	
J	Riboregulator reverse	
K	Hammerhead ribozyme	
L	Hammerhead ribozyme reverse	
P	T7 promoter	
Q	T7 promoter reverse	
W	Start codon	
X	Start codon reverse	

variables *E* and *F* used to represent genes defined as DNA fragments composed of a ‘‘start’’ codon followed by one or more protein domains and terminated by a ‘‘stop’’ codon. The third category of variables includes parts that can be duplicated in a construct. For instance, it is common practice to put two transcription terminators *G* at the end of a transcript to ensure a tight termination of the transcript. The fourth category contains all the variables that represent basic genetic parts that cannot be decomposed into smaller functional blocks and are not used in series in genetic constructs such as *A* (promoter), *C* (ribosome binding site) or *P* (T7 promoter). Variables representing less frequently used parts such as *I* and *J* (riboregulators) are also included in this category. The boundaries between the four categories used in Table 1 are arbitrary and have no consequence on the rest of the development. Listing variables in alphabetical order would have been equally acceptable.

Instead of using single capital letters that are difficult to interpret, we initially used more descriptive variables such as ‘‘promoters’’, ‘‘RBS’’, ‘‘coding’’, etc. Using descriptive notations hindered the visual display of complex sequences on the GenoCAD web site. By using single letters as variables, it was possible to generate a more compact graphical representation of the sequence and production rules. The lack of information in the single letter variables was compensated for by creating icons associated with each variable and by displaying in a mouse-over tooltip the description of each variable as it appears in the center column of Table 1.

2.2 Terminal set

The terminal set is composed of the genetic parts themselves. A library of more than 100 parts has been organized according to the syntactic categories used in this article. Parts have been indexed by a unique identifier composed of a prefix corresponding to the part syntactic category and a numerical suffix indexing the parts within each category. For instance, the terminals *a01* to *a09* point to the promoters of the library, whereas genes are represented by the terminals *e01* to *e14*, etc. This library is distributed in two computer-readable formats in the Supplementary Material. In addition to the unique identifiers used as terminals in the CFG, the library files include a part name. In addition, the DNA sequence of each part is included in the library as a proof of concept. These sequences have not been validated experimentally and it is sometimes difficult to extract precise sequence information from the sequences of previously published genetic constructs as the delimitation of parts within the construct is often sketchy in the literature. The development of an experimentally validated parts library is beyond the scope of this article.

2.3 Productions and construct design

Table 2 includes a list of production rules grouped according to the successive steps followed when designing a genetic construct. The process starts at *S*, the transcript. P01 can be applied to *S* several times to fix the construct total number of transcripts. Step 2 of the design process will specify each transcript by choosing a type of promoter and an orientation. Applying P02 to *S* will ensure that the transcript uses the endogenous RNA polymerase by selecting promoters and transcription terminators compatible with this enzyme. Alternatively, the transcript could rely on the bacteriophage T7 RNA polymerase in which case P04 will be applied to *S*. Using P02 or P04 will result in transcripts in the direct orientation. Alternatively, P03 or P05 can be used to generate transcripts in the reverse orientation. In Step 3, it is possible to specify if the transcript is polycistronic by applying P06 or P07 in the direct or reverse orientation, respectively. In Step 4, the architecture of transcripts is specified. P08 specifies that *M* is regular mRNA by decomposing it into a Ribosome Binding Site (RBS) *C* and a coding sequence *E* whereas P09 can be used when *M* is composed of a riboregulator *I* placed between two ribozymes *K* (Bayer and Smolke, 2005). The coding sequence *E* can itself be broken down by P12 into a start codon *W*, a protein domain *U* and a stop codon *Y*. Productions P10, P11 and P13 are the counterparts of P08, P09 and P12 for sequences in the reverse orientation. It is not unusual to place more than one part of a particular type in a specific location. Step 5 can be used to specify the number of repetitions for each part of the construct that can be repeated. For instance, multiple linkers corresponding to different restriction sites can be placed between transcripts by applying P16 several times. Similarly, it is common to place two successive transcription terminator sequences (P14, P15) or two stop codons (P17, P18) to ensure a tight termination of transcription and translation, respectively. P19 and P20 can be used to place additional protein domains to the coding sequence of a gene. In Step 6, it is possible to add linkers, DNA elements having a structural role but not involved in the gene expression mechanisms, next to some parts in the constructs. Typical linkers include restriction sites that could be used to extract parts in a construct and replace them by ligation of a DNA fragment extracted from a different construct.

At this stage of the design process, the general architecture of the construct is completely specified as a series of parts belonging to specific functional categories. However, the specific parts used to build the construct are yet to be specified. For instance, the construct could be described by a string such as *ACWUUY* (promoter, RBS, start codon, 2 protein domains, stop codon) but the particular promoter, RBS, start and stop codons, or the protein domains used to assemble a specific construct have not yet been specified. Therefore, this string does not describe a specific construct but a family of constructs expressing a protein. This family includes a wide range of transcription and transcription levels and any protein composed of three domains.

Table 2. Production rules

P01	$S \rightarrow SOS$	Start symbol (S), linker (O), start symbol (S)	Step 1
P02	$S \rightarrow AMG$	Promoter (A), transcript (M), terminator (G)	Step 2
P03	$S \rightarrow HNB$	Terminator rev (H), transcript rev (N), promoter rev (B)	
P04	$S \rightarrow PMR$	T7 promoter (P), transcript (M), T7 terminator (R)	
P05	$S \rightarrow TNQ$	T7 terminator rev (T), transcript rev (N), T7 promoter rev (Q)	
P06	$M \rightarrow MM$	Transcript (M), transcript (M)	Step 3
P07	$N \rightarrow NN$	Transcript rev (N), transcript rev (N)	
P08	$M \rightarrow CE$	Ribosome binding site(C), gene (E)	Step 4
P09	$M \rightarrow KIK$	Hammerhead (K), riboregulator (I), hammerhead (K)	
P10	$N \rightarrow FD$	Gene rev (F), ribosome binding site rev (D)	
P11	$N \rightarrow LJL$	Hammerhead rev (L), riboregulator rev (J), hammerhead rev (L)	
P12	$E \rightarrow WUY$	Start codon (W), protein domain (U), stop codon (Y)	
P13	$F \rightarrow ZVX$	Stop codon rev (Z), protein domain rev (V), start codon rev (X)	
P14	$G \rightarrow GG$	Terminator (G), terminator (G)	Step 5
P15	$H \rightarrow HH$	Terminator rev (H), terminator rev (H),	
P16	$O \rightarrow OO$	Linker (O), linker (O),	
P17	$Y \rightarrow YY$	Stop codon (Y), stop codon (Y)	
P18	$Z \rightarrow ZZ$	Stop codon rev (Z), stop codon rev (Z)	
P19	$U \rightarrow UU$	Protein domain (U), protein domain (U)	
P20	$V \rightarrow VV$	Protein domain rev (V), protein domain rev (V)	
P21	$A \rightarrow OA$	Linkers can be added next to some parts	Step 6
P22	$B \rightarrow OB$		
...	...		
P0100, P0101...	$A \rightarrow a1 \mid a2 \mid \ldots$	All variables can betransformed into terminals	Step 7
P0200, P0201...	$B \rightarrow b1 \mid b2 \mid \ldots$		
...	...		
P2400, P2401...	$Z \rightarrow z1 \mid z2 \mid \ldots$		

The last phase of the design process (Step 7) involves transforming variables into terminal symbols pointing toward specific DNA sequences. Productions corresponding to this step are the most numerous because there is one production for every part available to the designer. Table 2 provides only the general architecture of this last group of productions. Productions starting from the same variable have been grouped on a single line using the standard notation Variable → Terminal 1 | Terminal 2 |...indicating that a variable can be transformed into any of the terminals separated by |. All the grammar variables can potentially be transformed into a terminal or this type of transformation can be restricted to a category of variables corresponding to the most basic genetic parts. For instance, a variable like E (gene) can be transformed into terminals corresponding to a self-contained coding sequence or it can be transformed into a coding sequence composed of multiple domains between a start and stop codon. The most extreme case would be to include productions allowing the transformation of the start symbol S into a terminal. Allowing this type of production in the grammar maximizes flexibility since any DNA fragment can be made valid. However, this option makes it possible to completely bypass the design process enforced by the grammar, which may not be desirable. The design process is completed when all non-terminal variables have been transformed into terminals. At this stage the construct is represented by a series of terminal part identifiers. This high-level description of the construct can be converted into a DNA sequence suitable for gene synthesis using the sequence data of each of the parts in the part library.

3 RESULTS

3.1 Parsing for construct verification

The construct design process applies a series of productions starting from S to generate a construct with a structure consistent with the grammar rules. The design process therefore "derives" the construct from S. A computationally more complex problem is evaluating whether or not a specific construct can be generated by a given grammar. In order to answer this question it is necessary to construct the DNA sequence into S through the application of the grammar productions. This operation is called parsing. By parsing a construct, it is possible to verify its design, which is useful if the construct was not generated by the systematic process outlined in the previous section.

Prior to parsing the construct, it is necessary to perform a lexical analysis of the construct DNA sequence to transform it into a series of parts (Appel and Palsberg, 2002). As a proof of concept, we have developed a basic lexical analyzer that scans the parts list and compares the sequence of each part with the start (leftmost) sequence of the construct. If the part does not match the start of the construct sequence, the next part in the library is evaluated. At the end of the scan, it is possible that no part matches the beginning of the construct sequence, in which

Table 3. Parsing results of selected parts from the Registry

ID	Source	Symbolic representation	Result	Comment
BBa_J04450	Registry	a03c01e01g03	Pass	
BBa_I13520	Registry	a01c01e01g01g02	Pass	
pMKN7a	(Gardner *et al*., 2000)	a08c08e14g01g02	Pass	
BBa_J13004	Registry	a02c01e03c01e04g01g02	Pass	
BBa_I13513	Registry	a01c01e01o02c01e09g01g02	Pass	
pTAK102	(Gardner *et al*., 2000)	h02h01f01d04b02a08c07e14g01g02	Pass	
pTAK117	(Gardner *et al*., 2000)	h02h01f01d04b02a08c08e15c05e14g01g02	Pass	
BBa_J23022	Registry	I01g01g02	Fail	No promoter
BBa_J36335	Registry	a03c01e05a03c01e06	Fail	Lack of terminator
BBa_J44003	Registry	o01a04o01c02e07	Fail	Lack of terminator
BBa_J45119	Registry	c03e02g01g02	Fail	No promoter
BBa_J52038	Registry	a05e08	Fail	No RBS, no terminator
BBa_E0241	Registry	c03e09g04	Fail	No promoter
BBa_J5516	Registry	a01c01e12g01g02a06	Fail	Orphan promoter in 3′

case, the construct is rejected. If only one match is found, then the part matching the construct sequence is recorded and the rest of the construct DNA sequence is analyzed in the same way. It is also possible that several matches will be found if the parts library includes complex parts composed of more basic parts. In this case, all the matches are recorded possibly leading to multiple lexical interpretations of the construct sequence. The presence of multiple interpretations of a construct DNA sequence is an indication that the parts list is redundant in the sense that it includes complex parts that can be obtained by concatenation of more basic parts. It would be preferable to ensure that the CFG defined in the parts library includes rules allowing the derivation of complex parts from the basic parts.

The development of efficient parsing algorithms is an important problem in computer science since its solutions directly affect the performance of interpreters and compilers of programming languages. An introduction to parsing methods can be found in computer science textbooks (Linz, 2006). JFLAP is a very nice tool allowing the non-specialist to experiment with formal languages with a strong emphasis on automata theory (Rodger and Finley, 2006). JFLAP, however, is not suitable for the development of complex grammars or the analysis of large strings. YACC and Bison are production grade tools that can be used to develop compilers that rely on the LALR parsing algorithm. It is possible to code the grammar defined in Table 2 into Bison to build a custom parser capable of analyzing genetic constructs (data not shown). However, this requires proficiency in the C programming language and each time the grammar or the parts list is edited, the parser needs to be recompiled.

We have therefore developed a custom parser relying on the LR(0) algorithm and a specific precedence of the productions. After the lexical analysis step, the input string is converted into a series of non-terminal variables through the productions listed in Step 7 in Table 2. In a second step, the parser eliminates possible shift-reduce conflicts by eliminating series of identical variables that recursive productions can create in the construct (Steps 3 and 5 in Table 2). Finally, the resulting string is processed using the precedence set by the order of the productions. The Supplementary Material includes an animation that may help readers unfamiliar with parsing algorithms to visualize the process.

3.2 Validation

In order to validate the grammar in Table 2 and the parsing algorithm, a series of complex constructs described in the MIT Registry and in various publications (Elowitz and Leibler, 2000; Gardner *et al*., 2000; Guet *et al*., 2002) is reported in Table 3. Each construct is recognized by the identifier used in the source reference. Constructs are described by a series of lexical tokens corresponding to basic genetic parts. Most constructs were selected to illustrate different types of construct architectures generated by the grammar. However, some constructs outside of the language generated by the grammar have also been introduced in this validation set to illustrate structures outside the language generated by the grammar. The outcomes of the construct parsing are reported along with some comments explaining why some constructs failed the verification. A larger set of test cases than can be reported in Table 3 is available in the Supplementary Material.

In order to validate the lexical analyzer, we analyzed the sequences of several bistable genetic switches (US Patent 6 841 376). Tables 3–5 of this publication indicate the location of the promoters, RBS and genes used to implement the switches. In addition, the sequence list provides the sequence data for the promoter and RBS in addition to the complete sequences of the plasmids. The sequence of the transcription terminators labeled T1T2 in Figure 3 of Gardner *et al*. (2000) does not appear to be documented. The sequence of part g01 (also BioBrick BBa_B0010) was identified in the location of T1, but the sequence of the second terminator T2 could not be identified. We also noticed that the sequence of the *GFP-mut3* and *LacI* genes found in the published plasmid sequences is not exactly the same as the sequence published in the MIT Registry. We therefore created new parts (e17, f03, f04) corresponding to variants of these genes. We also observed that RBS sequences

A

Step	Production	String
1	P01	**SS**
2	P03	HNB**S**
	P02	HNBA**M**G
3	P06	H**N**BAMMG
4	P10	HFDBA**M**MG
	P08	HFDBACE**M**G
	P08	**H**FDBACECEG
5	P15	HHFDBACECEG
	P14	HHFDB**A**CECEGG
6	P21	HHFDB0ACECEGG
7		h02h01f01d04b02o01a08c08e15c05e14g01g02
		tataa..gcgttata

B

```
[tataaacgcagaaaggcccacccgaaggtgagccagtgtga][gagagcgttcaccgacaaacaacaga
taaaacgaaaggcccagtctttcgactgagcctttcgttttatttgatgcctgg][ttaagc…cacca][
catcgaaccggtttcct][tcctttgcataccctgctgatgtgctcattataaccgccagtggtatttat
gtcaacaccgccagagataatttatcaccgcagatggttatctgtgcatgc][ttatcaaaaaccatggt
ttttgataa][ccatcgaatggctgaaatgagctgttgacaattaatcatccggctcgtataatgtgtgg
aattgtgagcggataacaatttcacacagga][aggaaaccggttatg][atgagca…ttacaa][agga
atttaaatg][atgcgt…aaataa][ccaggcatcaaataaaacgaaaggctcagtcgaaagactgggcc
tttcgttttatctgttgtttgtcggtgaacgctctc][tcacactggctcaccttcgggtgggcctttct
gcgtttata]
```

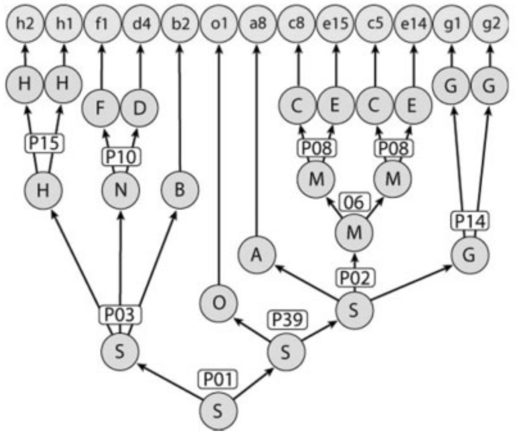

Fig. 1. (A) The successive applications of productions starting from S provide a framework to guide the design of genetic constructs. (B) The verification of an existing DNA sequence requires the use of a lexical analyzer to identify the parts composing the sequence. The symbolic description of the sequence provided by the lexical analyzer can be parsed using an LR algorithm.

overlapped the sequence of the genes placed in 3′ since all the RBS sequences included the start codon ATG. We have edited the RBS sequence to remove the ATG codon responsible for the overlap between the RBS and gene sequence. The sequences of the promoters used to build the switches included the AGGA motif of the Shine-Delgarno sequence which was also part of the RBS sequences. By removing AGGA from the promoter sequences, we resolved the overlap between the promoter and RBS sequences. We also observed a discrepancy between the sequence of the *PL-s1con* promoter (Sequence 2 in the patent and part a09 and b02 for the reverse orientation) and the sequence used in the plasmids that led to the introduction of part b04. Finally, we observed sequence variations in the *LacI* gene in the different plasmids, which we addressed by creating parts f03 and f04. The regions of the plasmids outside of the switches were treated as linkers in the context of this verification. The annotated sequence of the six switches is provided in the Supplementary Material. We introduced variants of the different parts found in the plasmid sequences into the parts list. The DNA sequences of the six switches could be analyzed by the lexical analyzer and their structure verified by the parser. All the plasmids could be parsed.

4 DISCUSSION

4.1 Grammar form and limitations

Even though a single grammar has been presented in this article, it is important to stress that this grammar is a somewhat arbitrary set of design of principles. It certainly does not encompass all natural DNA sequences. Even designers of some synthetic constructs have used unusual architectures that are not included in the language generated by our grammar. For instance, multiple promoters have been used to control the expression of a gene (Finn *et al.*, 2004). By adding the production $A \rightarrow AA$ to the grammar in Table 2, it would be possible to authorize the use of multiple promoters in constructs. This example shows that the grammar is nothing more than a set of accepted rules selected by the user of GenoCAD to design new constructs or analyze pre-existing ones. These rules come from the current understanding of the molecular mechanisms controlled by the different genetic parts used in genetic constructs.

The set of production rules listed in Table 2 was structured with the goal of minimizing the number of variables to ensure a biological interpretation of these variables. In addition, the rules have been organized in a way that mimics the way biologists design genetic constructs. The priority was to illustrate the approach by a grammar that could be interpreted in biological terms. As a result, most of the rules recurse by simply duplicating non-terminals, which leads to needless non-determinism and greater complexity in parsing. The grammar could be transformed into an equivalent grammar in a normal form such as the Greibach Normal Form. This transformation would make parsing more natural but would be difficult to read for biologists. Ultimately, it is possible that equivalent grammars in different forms will be used for design and parsing purposes.

Even though it is presented in CFG form, the grammar in Table 2 could probably be represented by a simpler regular language. In addition, the grammar does not take advantage of more advanced features of CFG that could be used, for instance, to express long distance interactions between an enhancer and a promoter, a transcription factor and its operator sequences, etc. Natural sequences include complex features such as overlapping genes (Miller *et al.*, 2003; Pavesi, 2006), introns and splicing sites (Landthaler and Shub, 1999; Miller *et al.*, 2003) or alternative splicing (Landthaler and Shub, 1999) that are not readily expressed by a CFG but could be expressed by richer Definite Clause Grammars (DCG) or String Variable Grammars (Dong and Searls, 1994; Searls, 1992, 1993). The readily available Prolog programming language is able to parse DCG and this environment has been extensively used to build complex gene grammar and parsers (Searls, 1997; Searls and Noordewier, 1991).

Future efforts will attempt to use this approach for the verification of the DNA sequences of genetic constructs.

4.2 Data model for libraries of genetic parts

Software applications can use syntactic models of genetic constructs to increase the productivity of individual users. Syntactic models could also be used to improve infrastructures serving the entire community. Syntactic categories provide a rigorous foundation to the organization of genetic parts in different categories. The "Transcriptional regulator" category of the Registry contains a large collection of prokaryotic promoters. However, some complex constructs composed of multiple parts (BBa_I13005 or BBa_J24669) are also found in this category even though they would probably fit in a category corresponding to a higher level of abstraction. Similarly, a number of eukaryotic promoters are listed in the Transcriptional regulator category. It might be preferable to have Eukaryotic transcription activators listed in their own category as they are not compatible with other prokaryotic genetic parts. By using syntactic models to develop community infrastructures, it would be possible to verify user submissions and existing content. As artificial gene networks become more complex by combining parts coming from distant organisms (Chen and Weiss, 2005; Finn *et al.*, 2004), a broader syntactic model than the one presented in this paper will help articulate rules of compatibility between parts. Of particular importance is the inclusion of existing knowledge relating to the use of prokaryotic transcription factors in eukaryotes (Gonzalez-Nicolini and Fussenegger, 2005; Meyer-Ficca *et al.*, 2004; Padidam, 2003).

As syntactic models of genetic constructs become broader, it might become necessary to specify the context in which the construct will be used. The tetracycline repressor has been shown to work in multiple organisms including mammalian cells and some plants (Berens and Hillen, 2003; Gossen and Bujard, 1992; Padidam, 2003; Wang *et al.*, 2003) but not all plants. In this context, the distinction between prokaryotes and eukaryotes may be not sufficient. The distinction between mammalian cells and plant cells may not be sufficient either as it may be desirable to specify the species in which this transcription factor can be used. Similarly, a number of eukaryotic promoters are tissue-specific whereas the activity of other promoters is not affected by the type of cells in which they are used. Each context will require the development of separate sets of production rules, but some parts should be useable in multiple contexts.

4.3 Limitations of syntactic models

The models and tools presented in this article rely on a higher level of abstraction than the DNA sequence. When using a syntactic model to guide the design of a new construct, it is straightforward to translate the description of the construct into a sequence since each genetic part corresponds to a unique sequence. However, verifying the sequence of genetic constructs is more complicated. The very basic lexical analyzer used in the context of this work is too rigid for practical use. The development of a more flexible analyzer capable of handling constructs with legacy sequences interspersed between functional parts will require dedicated efforts.

Another limitation of this syntactic model is its purely structural nature. There is no reference to the function of the parts used in the construct. To capture structure-function relationships, it will be necessary to develop a semantic model of genetic constructs that would complement the syntactic model presented here.

4.4 Beyond the proof-of-concept

It will take some time after the publication of this proof-of-concept paper to gain a better perspective on the advantages and limitations of this approach. Previously published artificial gene networks appear to have been designed by a labor-intensive and error-prone process. To the best of our knowledge, no other framework has been proposed at this time to streamline and formalize this process. We are investing significant efforts in the development of a user-friendly web site allowing biologists to design new constructs from previously defined grammars and parts libraries. Users will also be provided with tools to customize their grammars and parts libraries. Will users having no previous experience with formal languages be comfortable using this approach? Will they use it preferably to design new sequences or to verify sequences designed using other approaches? The analysis of patterns of activity on the Genocad.org web site will, over time, provide the best evaluation of the approach.

ACKNOWLEDGEMENTS

We thank two anonymous reviewers for insightful comments on an earlier version of this work and suggestions of future research directions. We are indebted to Otto Folkerts and Rebecca Shelton for their thorough reading of the manuscript. This work was supported with a startup fund from the Virginia Bioinformatics Institute and a fellowship from the Genetics, Bioinformatics, and Computational Biology Program at Virginia Tech.

Conflict of Interest: CG holds an interest in DNA2.0, a gene synthesis company.

REFERENCES

Appel,A.W. and Palsberg,J. (2002) *Modern Compiler Implementation in Java.* Cambridge University Press, Cambridge, UK; New York, USA.

Baker,D. *et al.* (2006) Engineering life: building a fab for biology. *Sci. Am.*, **294**, 44–51.

Bayer,T.S. and Smolke,C.D. (2005) Programmable ligand-controlled riboregulators of eukaryotic gene expression. *Nat. Biotechnol.*, **23**, 337–343.

Benner,S.A. and Sismour,A.M. (2005) Synthetic biology. *Nat. Rev. Genet.*, **6**, 533–543.

Berens,C. and Hillen,W. (2003) Gene regulation by tetracyclines. *Eur. J. Biochem.*, **270**, 3109–3121.

Brendel,V. and Busse,H.G. (1984) Genome structure described by formal languages. *Nucleic Acids Res.*, **12**, 2561–2568.

Brendel,V. *et al.* (1986) Linguistics of nucleotide sequences: morphology and comparison of vocabularies. *J. Biomol. Struct. Dyn.*, **4**, 11–21.

Cavin Perier,R. *et al.* (1998) The Eukaryotic Promoter Database EPD. *Nucleic Acids Res.*, **26**, 353–357.

Chen,M.T. and Weiss,R. (2005) Artificial cell-cell communication in yeast Saccharomyces cerevisiae using signaling elements from Arabidopsis thaliana. *Nat. Biotechnol.*, **23**, 1551–1555.

Chin,J.W. (2006) Programming and engineering biological networks. *Curr. Opin. Struct. Biol.*, **16**, 551–556.

Collado-Vides,J. (1992) Grammatical model of the regulation of gene expression. *Proc. Natl Acad. Sci. USA*, **89**, 9405–9409.

Dong,S. and Searls,D.B. (1994) Gene structure prediction by linguistic methods. *Genomics*, **23**, 540–551.

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

Endy,D. (2005) Foundations for engineering biology. *Nature*, **438**, 449–453.

Finn,J. *et al.* (2004) An enhanced autogene-based dual-promoter cytoplasmic expression system yields increased gene expression. *Gene Ther.*, **11**, 276–283.

Gardner,T.S. *et al.* (2000) Construction of a genetic toggle switch in Escherichia coli. *Nature*, **403**, 339–342.

Gimona,M. (2006) Protein linguistics – a grammar for modular protein assembly? *Nat. Rev. Mol. Cell Biol.*, **7**, 68–73.

Goeddel,D.V. *et al.* (1979) Direct expression in Escherichia coli of a DNA sequence coding for human growth hormone. *Nature*, **281**, 544–548.

Gonzalez-Nicolini,V. and Fussenegger,M. (2005) A novel binary adenovirus-based dual-regulated expression system for independent transcription control of two different transgenes. *J. Gene Med.*, **7**, 1573–1585.

Gossen,M. and Bujard,H. (1992) Tight control of gene-expression in mammalian–cells by tetracycline-responsive promoters. *Proc. Natl Acad. Sci. USA*, **89**, 5547–5551.

Guet,C.C. *et al.* (2002) Combinatorial synthesis of genetic networks. *Science*, **296**, 1466–1470.

Guido,N.J. *et al.* (2006) A bottom-up approach to gene regulation. *Nature*, **439**, 856–860.

Heinemann,M. and Panke,S. (2006) Synthetic biology – putting engineering into biology. *Bioinformatics*, **22**, 2790–2799.

Knudsen,B. and Hein,J. (1999) RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, **15**, 446–454.

Knudsen,B. and Hein,J. (2003) Pfold: RNA secondary structure prediction using stochastic context-free grammars. *Nucleic Acids Res.*, **31**, 3423–3428.

Kobayashi,H. *et al.* (2004) Programmable cells: interfacing natural and engineered gene networks. *Proc. Natl Acad. Sci. USA*, **101**, 8414–8419.

Landthaler,M. and Shub,D.A. (1999) Unexpected abundance of self-splicing introns in the genome of bacteriophage Twort: introns in multiple genes, a single gene with three introns, and exon skipping by group I ribozymes. *Proc. Natl Acad. Sci. USA*, **96**, 7005–7010.

Linz,P. (2006) *An Introduction to Formal Languages and Automata*. Jones and Bartlett Publishers, Sudbury, MA.

Loose,C. *et al.* (2006) A linguistic model for the rational design of antimicrobial peptides. *Nature*, **443**, 867–869.

Matsui,H. *et al.* (2005) Pair stochastic tree adjoining grammars for aligning and predicting pseudoknot RNA structures. *Bioinformatics*, **21**, 2611–2617.

McDaniel,R. and Weiss,R. (2005) Advances in synthetic biology: on the path from prototypes to applications. *Curr. Opin. Biotechnol.*, **16**, 476–483.

Meyer-Ficca,M.L. *et al.* (2004) Comparative analysis of inducible expression systems in transient transfection studies. *Anal. Biochem.*, **334**, 9–19.

Miller,E.S. *et al.* (2003) Bacteriophage T4 genome. *Microbiol. Mol. Biol. Rev.*, **67**, 86–156, table of contents.

Munch,R. *et al.* (2003) PRODORIC: prokaryotic database of gene regulation. *Nucleic Acids Res.*, **31**, 266–269.

Padidam,M. (2003) Chemically regulated gene expression in plants. *Curr. Opin. Plant Biol.*, **6**, 169–177.

Pavesi,A. (2006) Origin and evolution of overlapping genes in the family Microviridae. *J. Gen. Virol.*, **87**, 1013–1017.

Rigoutsos,I. and Floratos,A. (1998) Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics*, **14**, 55–67.

Rivas,E. and Eddy,S.R. (2000) The language of RNA: a formal grammar that includes pseudoknots. *Bioinformatics*, **16**, 334–340.

Rodger,S.H. and Finley,T.W. (2006) *JFLAP-an Interactive Formal Languages and Automata Package*. Jones and Bartlett, Sudbury, MA.

Sakakibara,Y. (2005) Grammatical inference in bioinformatics. *IEEE Trans. Pattern Anal. Mach. Intell.*, **27**, 1051–1062.

Sakakibara,Y. *et al.* (1994) Stochastic context-free grammars for transfer-Rna modeling. *Nucleic Acids Res.*, **22**, 5112–5120.

Searls,D.B. (1992) The Linguistics of DNA. *Am. Sci.*, **80**, 579–591.

Searls,D.B. (1993) The computational linguistics of biological sequences. In *Artificial Intelligence and Molecular Biology*. American Association for Artificial Intelligence, Menlo Park, CA, USA, pp. 47–120.

Searls,D.B. (1997) Linguistic approaches to biological sequences. *Comput. Appl. Biosci.*, **13**, 333–344.

Searls,D.B. (2002) The language of genes. *Nature*, **420**, 211–217.

Searls,D.B. and Noordewier,M.O. (1991) Pattern-matching search of DNA sequences using logic grammars. In *Seventh IEEE Conference on Artificial Intelligence Applications*. Vol. 1, pp. 3–9.

Villalobos,A. *et al.* (2006) Gene Designer: a synthetic biology tool for constructing artificial DNA segments. *BMC Bioinformatics [electronic resource]*, **7**, 285.

Voigt,C.A. (2006) Genetic parts to program bacteria. *Curr. Opin. Biotechnol.*, **17**, 548–557.

Wang,R. *et al.* (2003) Chemically regulated expression systems and their applications in transgenic plants. *Transgenic Res.*, **12**, 529–540.

Zhu,J. and Zhang,M.Q. (1999) SCPD: a promoter database of the yeast Saccharomyces cerevisiae. *Bioinformatics*, **15**, 607–611.