BIOINFORMATICS ORIGINAL PAPER

Vol. 30 no. 2 2014, pages 251–257
doi:10.1093/bioinformatics/btt646

Databases and ontologies

Advance Access publication November 8, 2013

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

Mandy L. Wilson[1], Sakiko Okumoto[2], Laura Adam[1] and Jean Peccoud[1,3,*]

[1]Virginia Bioinformatics Institute, [2]Department of Plant Pathology, Physiology, and Weed Science and [3]ICTAS Center for Systems Biology of Engineered Tissues, MC 0193, Virginia Tech, Blacksburg, VA 24061, USA

Associate Editor: Martin Bishop

ABSTRACT

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

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

Availability and implementation: The context-free grammar editor is part of the GenoCAD application. A public instance of GenoCAD is available at http://www.genocad.org. GenoCAD source code is available from SourceForge and licensed under the Apache v2.0 open source license.

Contact: peccoud@vt.edu

Supplementary Information: Supplementary data are available at *Bioinformatics* online.

Received on September 11, 2013; revised on November 1, 2013; accepted on November 4, 2013

1 INTRODUCTION

Most bioinformatics software packages include sequence editors that facilitate the design and assembly of new DNA sequences. Automatic recognition of sequence features, identification of restriction sites and tools to add sequence annotations help biologists visualize the different elements of the DNA sequences they manipulate. Software lets users switch between graphical representations of DNA sequences that provide a macroscopic view and textual representations more suitable to examine sequences with a base-level resolution. Irrespective of the software environment used, the cut-and-paste approach to sequence editing increases the chance of introducing errors, such as leaving or deleting a DNA segment, accidently inserting it twice or truncating a functional element. For a large multigene construct (e.g. an expression cassette encoding all components of a biochemical pathway), these risks could become unacceptably high. In many cases, errors are uncovered only after several months of unsuccessful attempts to express a gene.

Many of these errors can be avoided by developing a library of genetic parts before designing DNA sequences. The Registry of Standard Biological Parts (Peccoud *et al*., 2008) was the first database of genetic parts. The notion of genetic parts supports a different approach to sequence design: complex genetic constructs can be designed using drag-and-drop user interfaces that rely on icons to represent different categories of genetic parts (Villalobos *et al*., 2006). This added level of abstraction makes it easier to understand the structure of a new sequence. It also avoids sequence manipulation errors. However, it still makes it possible to design sequences lacking components required for proper gene expression.

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

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

We have now formalized the grammar development process and developed a graphical user interface enabling life scientists to develop context-free grammars. The GenoCAD grammar editor allows users to revise existing grammars or even to develop brand-new grammars. These grammars can be fairly generic to generate a broad range of expression vectors for a new host. Alternatively, they can be made specific to capture project-specific design constraints, such as the ones resulting from intellectual property licensing agreements. These languages describing families of synthetic DNA molecules are comparable with domain-specific languages (DSL) used in computer programming. Specialized DSLs facilitate communication between

*To whom correspondence should be addressed.

programmers and domain experts by directly expressing concepts specific to the domain. The Structured Query Language (SQL) is an example of a DSL used for programming databases.

The design of vectors to express polycistronic genes in the chloroplast of *Chlamydomonas reinhardtii* is an example of a domain that calls for the development of a DSL. Site-specific gene insertion into *Chlamydomonas* chloroplast can be performed through homologous recombination, making it an attractive system to study processes such as photosynthesis. It has been previously shown that recombinant protein can accumulate at much higher levels when expressed in the chloroplast genome of *Chlamydomonas* compared with the nuclear genome (Franklin *et al*., 2002). Also its generally oxidizing environment makes the formation of disulfide bonds possible, hence enabling expression of large complex proteins such as full-length antibodies (Mayfield *et al*., 2003). Owing to these characteristics, *Chlamydomonas* chloroplast recently gained much attention as a venue for recombinant protein production (Rasala and Mayfield, 2011).

Interestingly, *Chlamydomonas* chloroplast is capable of carrying out anaerobic reactions such as hydrogen production under well-studied specific circumstances, despite its generally oxidizing environment (Esquivel *et al*., 2011; Hemschemeier *et al*., 2009). Also, *Chlamydomonas* chloroplast genome has a prokaryotic arrangement, encoding both mono- and polycistronic genes, which offers the potential for introduction of polycistronic synthetic expression cassettes like in bacteria (Drapier *et al*., 1998).These characteristics make *Chlamydomonas* chloroplast an attractive venue to introduce genes required for nitrogen fixation, as (i) appropriate stoichiometry between the components is essential, which can be better controlled in a system that allows expression from polycistronic operons and homologous recombination; (ii) high level of expression is essential due to the slow kinetics of nitrogenase; and (iii) even though such a state is transient, anaerobicity can be achieved. Yet, the DSLs that are available for bacterial synthetic constructs would not be sufficient because in many cases the exact locations of regulatory components are not known. For example, ribosomal binding sequences (RBSs) are not characterized for most of the genes, and it is predicted that the translational initiation is more complex than the ancestry bacterial system. This necessitates more flexibility than a simple promoter-RBS-coding sequence (CDS) arrangement found in synthetic constructs for bacteria; in some cases, it is not practical to separate the promoter from 5′ untranslated region sequence that potentially includes RBS and other regulatory elements. Undoubtedly, the expression of multiple genes from a complex polycistronic unit would require empirical optimization of genetic parts through biological assays, hence a rapid method to assemble multiple parts without risk of introducing sequence mistakes is highly beneficial.

2 SYSTEM AND METHODS

2.1 Accessing the grammar editor

The context-free grammar editor can be accessed from GenoCAD's Parts module by clicking on the Parts header, then the Grammars tab from the left-hand navigation bar. This displays a list of the grammars available in the system, defined as either public grammars (the ones anyone can use and view) or user grammars (those that belong to the logged-in user). From this list, users can select a grammar from the list on the left and see a summary of the selected grammar on the right, including a description of the grammar, the number of categories and rules that define the grammar and the number of libraries and parts associated with that grammar.

2.2 Category definition

The grammar development process starts by declaring categories of genetic parts, such as promoters, transcription terminators or coding sequences. Each category needs an alphanumeric code. By convention, three or four capital letters such as PRO for promoter or TER for terminator work well. In addition, the category needs to be defined by providing a short text description of the type of genetic element corresponding to the category. An optional field is available to associate the category with a specific GenBank qualifier. This is used to ensure that parts in that category will be properly annotated when exporting expression vectors as GenBank files. Finally, the software allows users to select an icon that will be used to represent parts of this category in the GenoCAD design tool.

All new GenoCAD grammars come with a set of predefined reserved categories. The first is the start category (S), which is the default root to the hierarchical rules tree; it is required in the grammar definition that one category be designated as the root, but it is possible to designate a separate category to fulfill this role. The other reserved categories are used as sequence delimiters. Brackets are used to indicate the orientation of a DNA sequence, parentheses are used to delimit plasmids in constructs that involve more than one plasmid and braces are used to delimit chromosomes in genome design projects.

2.3 Rule declaration

As soon as a few categories have been declared, it is possible to start declaring rewriting rules. The rules can be added by means of a drag-and-drop interface that allows the user to pull categories from a list on the left and organize them on the right. Rules are identified using a short alphanumeric code. Lower case codes are used to differentiate them from category codes.

To facilitate the development of sets of transformation rules, it is convenient to identify three groups of categories. *Rewritable* categories are categories that are composed of one or more different categories. For example, rule sgen: CDS -> ATG GEN STP states that the category CDS is composed of a start codon (ATG), an open reading frame (GEN) and a stop codon (STP). *Terminal* categories are those that may not be transformed via a rule to another category or set of categories; usually DNA segments (parts) will be associated with terminal categories. ATG and STP are examples of terminal categories. *Orphaned* categories are those not used in any rewriting rule.

In a new grammar, all categories are orphaned categories. As new rules are added to the grammar, the software automatically reassigns categories to one of the three groups according to their use. A properly developed grammar should not have any orphaned categories. Predefined categories that are not applicable to a particular grammar should be deleted; for example, the chromosome delimiters are rarely used.

2.4 Grammar management

The development of a new grammar proceeds through multiple iterations that include both category and rule definitions. The development of new rules can necessitate the declaration of new categories. A test tool makes it possible to test the new grammar by experimenting with the existing rules. These simulations often uncover limitations of the existing rule sets.

Users may only make changes to their own grammars, but they can make copies of existing public grammars and use these as templates for the development of new grammars. In addition, grammar files may be

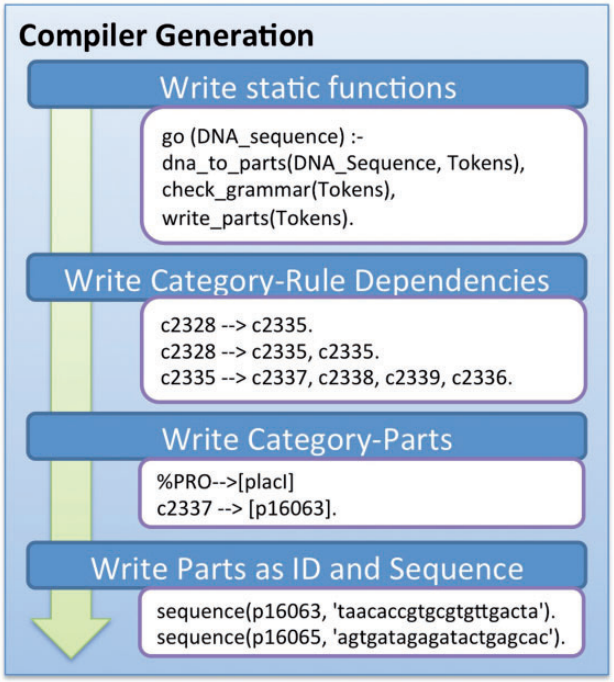


Fig. 1. Compiler generation workflow. When validating a design, GenoCAD generates an on-demand compiler written in Prolog that is tied to the library and grammar of the design

imported into a user's workspace; such files are available from a number of sources. We have previously published several GenoCAD grammars (Cai *et al.*, 2007, 2010; Czar *et al.*, 2009).

When users copy or export a grammar, they have the option of also including any parts libraries, and, by extension, designs and parts, with the copied grammar; this is a quick way to export a complete data set and share it with a colleague who can load it into their own GenoCAD workspace for review and further modification.

2.5 Parser generation and plasmid verification

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

The first step is to generate a compiler specific to the grammar and library that the design is associated with; this process is illustrated in Figure 1. Written in Prolog, this compiler consists of some hard-coded functions to handle processing of the design, but also contains the grammar's rule specifications, a list of the parts supported by the library and the relationship between the categories and the parts. The compiler's algorithm incorporates strategies from Moore's algorithm (Moore, 2000) to remove immediate left recursion from context-free grammars because Prolog does not natively support it; this feature makes it possible for users to define rules like 'CAS -> CAS CAS', where the first category on the right side of the rule is the same as the category on the left side. It should also be noted that this compiler needs to be regenerated every time the design is validated to ensure that all modifications to the grammar, library or parts have been captured.

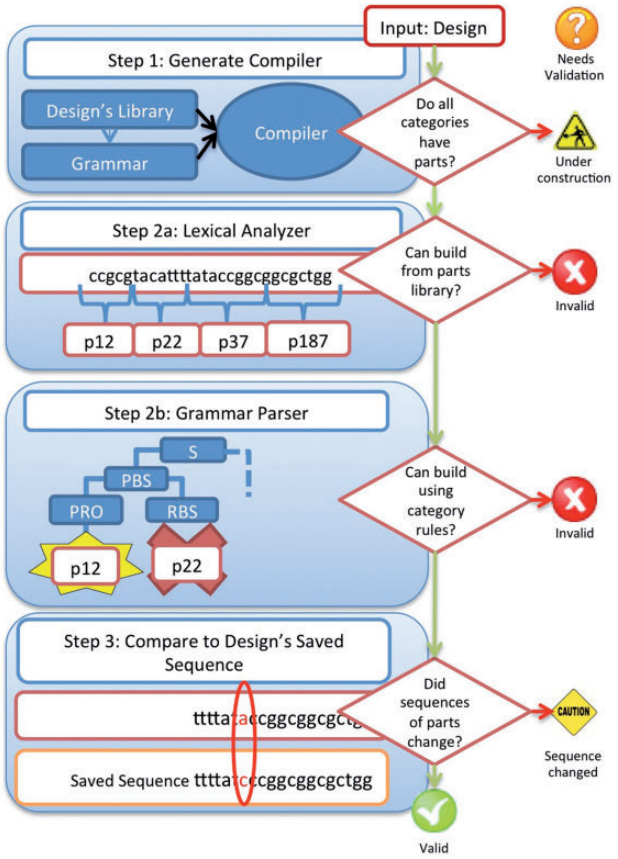


Fig. 2. Validating designs. In Step 1, the compiler is generated from the design's grammar and library. In Step 2, the design sequence is run through the compiler to ensure it is compliant with the parts library and the grammar. In Step 3, the sequence based on the parts is compared with the previously saved sequence in case one of the part sequences has changed

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

If the sequence is deemed valid during the second step, the third step is to compare the sequence generated from the parts of the design to the sequence of the design from the last time it was saved; this is a simple string comparison and does not involve the compiler. This check is performed because a design may still be consistent with its parts library and grammar, but changes to the sequences of underlying parts could result in a different design sequence than the one that was originally developed. In this case, the design is marked with a caution sign, which warns that while the design is still valid based on its parts and grammars, the underlying sequence has changed. The revalidation process is illustrated in Figure 2.

3 RESULTS

We have applied the grammar-design workflow described before to develop a grammar to design vectors that can express one or multiple genes from a DNA sequence as inserted in the chloroplast chromosome.

3.1 Category definition

The list of categories includes those found in many grammars, like open reading frames, tagging sequences or transcription terminators. Here, we have decided to declare start and stop codons as a category. This common convention facilitates the design of fusion proteins by concatenation of multiple protein coding sequences lacking stop codons.

The grammar also includes categories that are specific to this application. For instance, the translation initiation sites are not as well characterized in *C.reinhardtii* as in *Escherichia coli*. The promoter category is defined as sequences that include both transcription and translation initiation sites. The *C.reinhardtii* chloroplastic chromosome has several operons in which adjacent coding sequences are separated by short interval sequences. We decided to define these sequences as a part category that will be used to make polycistronic cassettes. To target a specific region of the chloroplast chromosome by homologous recombination, the constructs need to include flanking sequences in 5′ and 3′ positions of the expression cassette. The grammar also includes a number of rewritable categories corresponding to large structural blocks. Another category labeled 'targeted expression cassette' corresponds to expression cassettes flanked by two genomic sequences for homologous recombination. Figure 3 lists all the grammar categories along with their definition, abbreviation and graphical representation.

3.2 Parts sequences

On completion of the category definition process, we imported sequences into each category. Nif genes were originally from *Azotobacter* genome (accession number M20568) (Jacobson *et al*., 1989). The sequence of nif genes and tags were codon-optimized for *Chlamydomonas* using OPTIMIZER (Puigbo *et al*., 2007).

The sources of promoter and Short Interval Sequences (SISs) are described in the online supplement. Additional information regarding the origin of parts sequences is available in the parts description field.

3.3 Rules declaration

The rules for the *C.reinhardtii* grammar are illustrated in Figure 4. The best way to review the rule set is to start from S. S can be rewritten into a single targeted expression cassette (rule 1tcs), a complete plasmid that includes both a cassette and a vector backbone (rule 1plas) or a construct that includes two plasmids (rule 2plas) that could be convenient in cases where the size of the insert exceeds the cloning capacity of the vector backbone.

Three rules are then used to rewrite the CAS category. Rule 2cas is used to introduce an additional cassette in the design. Rule rcas is used to change the orientation of the cassette. Finally rule prct is used to break the cassette down into promoter, open reading frame and transcription terminator.

The rewriting of CDS is the focus on the next two rules. Rule 2cds is used to create polycistronic constructs. Rule sgen reveals the start and stop codons that delimit the open reading frame (GEN).

Finally, the grammar has three rules to rewrite the GEN category. Rule 2gen creates the possibility to make fusion proteins by combining two open reading frames. Rules tgen and gent correspond to the addition of an epitope tag on the N or C terminus of the open reading frame.

3.4 Sample designs

Four designs of varying complexity were developed to illustrate how one grammar can have the flexibility to create a variety of designs. These designs can be viewed in Figure 5.

Design 1 illustrates the implementation of a targeted expression cassette (TCS). Although usually a TCS would be implemented as part of a plasmid, this example illustrates how partial sequences can be implemented in GenoCAD, then further refined in another application. This design includes a HisTag, along with the NifB and lucCP genes.

Design 2 takes this concept a step further by encapsulating the first design within a complete plasmid, along with a vector. The special case delimiters for the plasmid (indicated with parenthesis) is handled when the sequence is downloaded; this is especially evident in the GenBank download.

Design 3 builds on the second design by adding a second nif gene on a single transcript; it also relies on an SIS.

Design 4, on the other hand, relies on two cassettes instead of an SIS part. An additional feature that makes Design 4 distinct from the others is that the first cassette is implemented in the reverse complement orientation, as indicated by the special case reverse complement delimiter tags (brackets.) Whenever the sequence is downloaded, the sequence encapsulated between those delimiters is reverse complemented before the final sequence is returned for download; that portion will also be indicated in the GenBank export as complement features.

All four designs are included as part of the *C.reinhardtii* chloroplast grammar provided in the Supplementary Material.

3.5 Design verification

GenoCAD provides a validation feature to notify the user if changes to underlying parts and grammars have impacted any of the designs. To demonstrate this capability, we copied the *C.reinhardtii* chloroplast grammar, along with its designs, and made modifications as described later in the text. It should be noted that at the beginning of this exercise, all four designs were in 'valid' status; the statuses of the designs after the changes are displayed in Figure 6.

The first change was to deselect the terminator in Design 1 and save the design. On revalidation, the design reflects that it is in 'under construction' status. It will retain this status until someone selects a terminator so that all of the terminal categories have associated parts.

The second change was to edit the sequence of VEC part pTJ322 (Noor-Mohammadi *et al*., 2012), which is used in designs 2, 3 and 4. Initially, all three designs subsequently showed a

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

Fig. 3. Categories of genetic parts used in the *C.reinhardtii* chloroplast grammar

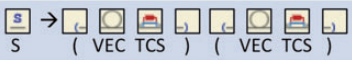
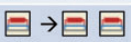
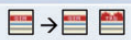
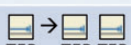
Rule Code	Rule	Description
1tcs	 S TCS	This rule is used to design only one expression cassette
1plas	 S (VEC TCS)	This rule is used to specify the expression cassette along with the vector where it is inserted. The output is the entire plasmid sequence.
2plas	 S (VEC TCS) (VEC TCS)	This rule is for designs that involve two plasmids.
tgs	 TCS 5FLR CAS 3FLR	Specifies the flanking regions for homologous recombination.
2cas	 CAS CAS CAS	This rule makes it possible to have more than one expression cassette on a construct.
rcas	 CAS [CAS]	This rule is used to specify that the cassette is coded on the negative strand.
prct	 CAS PBS CDS TER	A gene expression cassette is composed of a promoter, open reading frame, and a transcription terminator.
2cds	 CDS CDS SIS CDS	This rule makes it possible to design polycistronic constructs.
sgen	 CDS ATG GEN STP	The open reading frame is composed of a single gene flanked by a start and stop codon.
2gen	 GEN GEN GEN	This rule can be used to fuse two coding sequences that are not tags.
tgen	 GEN TAG GEN	This rule is used to add a tag before the gene. It can be used iteratively to add more than one tag.
gent	 GEN GEN TAG	This rule is used to add a tag after the gene. It can be used iteratively to add more than one tag.
2ter	 TER TER TER	This rule makes it possible to include a double terminator.

Fig. 4. The rules for the *C.reinhardtii* chloroplast grammar

Fig. 5. The four sample designs developed using the *C.reinhardtii* chloroplast grammar

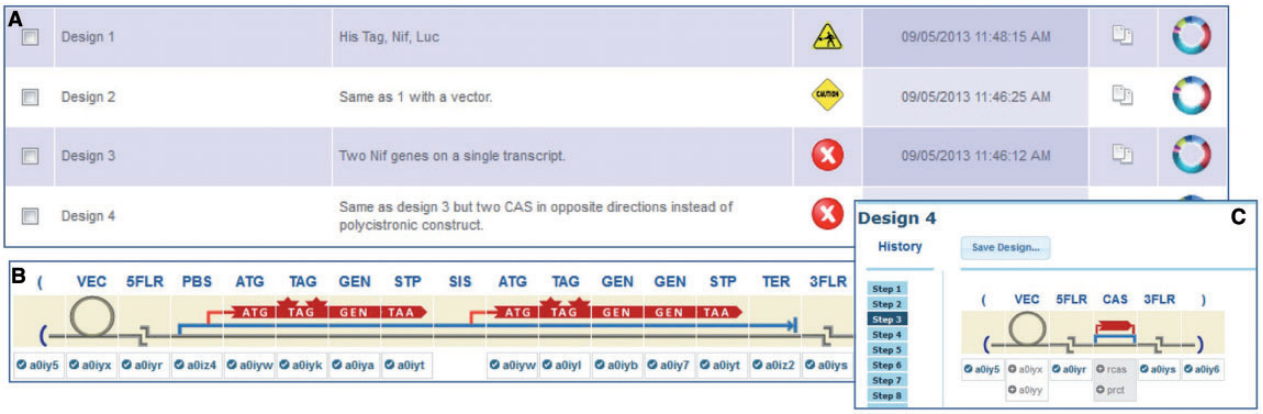


Fig. 6. Examples of design verification. (**A**) Changes to designs, parts, parts libraries and grammars can invalidate previously valid designs. (**B**) Design 3 after the selected SIS part was removed from the design's library. (**C**) Design 4 needs to be rebuilt from Step 3 after the deletion of the 2cas rule

status of 'needs validation', but on revalidation, they all showed a status of 'caution'. Although still valid designs, the design sequences for each, as implemented from the part segments, differ from their original design sequences. If the designs had been saved again, the original design sequences would be overwritten with the new sequences, as based on the parts, and the statuses would return to 'valid'.

After removing the SIS part rrnL_Cr_igs from the *C.reinhardtii* chloroplast parts library, Design 3 became invalid because it depended on that part; when displayed in the design screen, that part is not even displayed for the SIS element, and the user would have to go select another SIS part to make the design valid again.

Finally, using the grammar editor, the rule 2cas was deleted; this rule allowed the user to have two cassettes in their plasmid instead of just one. Design 4 contains two cassettes, so this design was also rendered 'invalid'. To restore this design to 'valid' status, the user would have to return to Step 3 and rebuild the design from that point; that said, the design could never be restored to its original form without the reimplementation of the 2cas rule.

4 DISCUSSION

A grammar should be flexible enough to support a variety of designs applicable to a certain application domain. Although there are often multiple ways to express a set of design principles, some grammar development strategies are better than others.

Rewriting rules that resolve to a long list of categories are usually an indication of bad grammar design. It is often possible

to break down this transformation into a set of simpler rewriting rules by introducing intermediate categories corresponding to shorter aggregations of categories. Defining these new categories is an opportunity to formalize an abstraction hierarchy.

When defining categories and rules, it is important to identify reusable patterns and implement rules that group these categories together. This limits the number of categories and rewriting rules.

A good grammar should be expressive enough to allow the generation of a broad range of constructs but should also be constrained enough to limit the possibilities of designing faulty constructs. How constraining it needs to be is something that should be discussed with a domain expert. For example, the grammar presented here allows users to design constructs distributed over two different plasmids. When two plasmids are used together, it may be helpful to ensure that they rely on different selection markers. It was decided that it was not necessary to express this constraint in rule 2plas, but the grammar could easily be modified should it become necessary to enforce this constraint.

The need to provide biologists with tools to express custom design strategies is somewhat antithetic of the need for standardization. For instance, we could not identify any GenBank qualifier corresponding to the domain-specific categories defined in the grammar, such as SIS, start codon (ATG), stop codon (STP) or vector (VEC). This limitation would lead to an incomplete rendering of designs built with this grammar if exported in GenBank format for import into another application.

Similarly, it proved challenging to use SBOLv, the set of standard icons developed by the Synthetic Biology Open Language (SBOL) project (Galdzicki *et al*., 2012). This difficulty forced us to develop a custom set of icons.

GenoCAD's grammar editor has been used to develop other DSLs. It has been used to model the structure of a family of vaccine vectors derived from the vesicular stomatitis virus (Overend *et al*., 2012). A grammar has also been developed for the design of synthetic transcription factors based on zinc fingers, transcription activator-like effectors and the recently developed clustered regularly interspaced short palindromic repeats CRISPR/Cas-based system (Purcell O., Peccoud J., Lu T.K., Grammar for the Design of Eukaryotic Synthetic Transcription Factors, submitted).

To make it easier for potential users to develop DSLs for their own applications using the GenoCAD grammar editor, this feature is introduced in the GenoCAD tutorial available from Figshare (doi:10.6084/m9.figshare.153827). Potential users are encouraged to familiarize with this tutorial and complete the exercises before starting the development of their own grammars.

Funding: National Science Foundation (Grant EF-0850100 to J.P.).

Conflict of Interest: JP has a financial interest in GenoFAB, LLC.

REFERENCES

Cai,Y. *et al.* (2007) A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts. *Bioinformatics*, **23**, 2760–2767.

Cai,Y. *et al.* (2010) GenoCAD for iGEM: a grammatical approach to the design of standard-compliant constructs. *Nucleic Acids Res.*, **38**, 2637–2644.

Czar,M.J. *et al.* (2009) Writing DNA with GenoCAD. *Nucleic Acids Res.*, **37**, W40–W47.

Drapier,D. *et al.* (1998) The chloroplast atpA gene cluster in *Chlamydomonas reinhardtii*. Functional analysis of a polycistronic transcription unit. *Plant Physiol.*, **117**, 629–641.

Esquivel,M.G. *et al.* (2011) Efficient H2 production via *Chlamydomonas reinhardtii*. *Trends Biotechnol.*, **29**, 595–600.

Franklin,S. *et al.* (2002) Development of a GFP reporter gene for *Chlamydomonas reinhardtii* chloroplast. *Plant J.*, **30**, 733–744.

Galdzicki,M. *et al.* (2012) Synthetic Biology Open Language (SBOL) Version 1.1. 0. *BioBricks Foundation RFC*, **87**.

Hemschemeier,A. *et al.* (2009) Analytical approaches to photobiological hydrogen production in unicellular green algae. *Photosynth. Res.*, **102**, 523–540.

Jacobson,M.R. *et al.* (1989) Physical and genetic map of the major nif gene cluster from *Azotobacter vinelandii*. *J. Bacteriol.*, **171**, 1017–1027.

Mayfield,S.P. *et al.* (2003) Expression and assembly of a fully active antibody in algae. *Proc. Natl Acad. Sci. USA*, **100**, 438–442.

Moore,R.C. (2000) Removing left recursion from context-free grammars. In: *6th Applied Natural Language Processing Conference/1st Meeting of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Conference and Proceedings of the Anlp-Naacl 2000 Student Research Workshop*. Morgan Kaufmann, pp. A249–A255.

Noor-Mohammadi,S. *et al.* (2012) Method to assemble and integrate biochemical pathways into the chloroplast genome of *Chlamydomonas reinhardtii*. *Biotechnol. Bioeng.*, **109**, 2896–2903.

Overend,C. *et al.* (2012) The synthetic futures of vesicular stomatitis virus. *Trends Biotechnol.*, **30**, 497–498.

Peccoud,J. *et al.* (2008) Targeted development of registries of biological parts. *PLoS One*, **3**, e2671.

Puigbo,P. *et al.* (2007) OPTIMIZER: a web server for optimizing the codon usage of DNA sequences. *Nucleic Acids Res.*, **35**, W126–W131.

Rasala,B.A. and Mayfield,S.P. (2011) The microalga *Chlamydomonas reinhardtii* as a platform for the production of human protein therapeutics. *Bioeng. Bugs*, **2**, 50–54.

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