Genetic design: rising above the sequence

Jonathan A. Goler[1], Brian W. Bramlett[2] and Jean Peccoud[3]

[1] Synthetic Biology Engineering Research Center, University of California Berkeley, CA 94720, USA
[2] Intel Corporation, Integrated Analytic Solutions, 5200 NE Elam Young (HF 1-61), Hillsboro, OR 97124, USA
[3] Virginia Bioinformatics Institute, Washington Street, MC0477, Blacksburg, VA 24061, USA

Genetic engineering has developed around technologies enabling the targeted *in vitro* recombination of DNA molecules found in living organisms. As a result, the development of new DNA molecules has been primarily focused on cloning strategies that allow their assembly from existing DNA fragments. As chemical gene synthesis matures, the design of synthetic DNA molecules becomes the bottleneck of many biotechnology projects. It becomes urgent to develop representations of synthetic genetic systems more abstract than their DNA sequence. Abstraction makes it possible to reuse simple components to build complex systems or to break down a complex engineering problem into manageable tasks. Specialized computer languages or a general purpose XCell Description Language are promising avenues to build abstraction hierarchies for synthetic biology.

Introduction

Even though the term 'biological engineering' was coined in 1958 by Edward L. Tatum in his Nobel lecture [1], genetic engineering still does not adhere to several methodologies commonly found in other engineering domains. In particular, the idea of considering separately the design and assembly of a recombinant molecule has only been expressed recently [2,3]. To leverage the full power of rapidly growing DNA synthesis capabilities, it is necessary to develop abstract representations capable of capturing the increasing sophistication of synthetic genetic systems.

In 2004, Blue Heron Biotechnology (http://www.blueheronbio.com/), a leading gene synthesis company, organized the Big DNA Contest. They offered to synthesize, free of charge, the most interesting genetic construct longer than 40 kb that people would send them. One of the stated goals of this initiative was to encourage scientists to consider the potential impact of large synthetic DNA fragments on their research strategies. The company did not receive a single submission even though the estimated value of the grand prize exceeded US$250 000. Although some of the terms of the offer might have contributed to the lack of applications, it is likely that none of their customers could design such a large synthetic DNA molecule. Even if there had been many contestants with exciting ideas, there were no tools or engineering processes available to implement those ideas. The end result of this contest perfectly illustrates the enormous challenge of conceptualizing and designing genetic sequences. The possibility of chemically synthesizing DNA fragments has dramatically increased since 2004. The number of companies offering gene synthesis services is rapidly growing, and the prices are dropping so dramatically that it is often more cost-effective to order kilobases of synthetic DNA than to obtain these molecules by cloning. The redesign of 12 kb of the phage T7 [4] genome and of the heterologous expression of polyketide synthase [5] are among the largest synthetic DNA molecules to be described in the literature. Furthermore, several gene synthesis companies have reported the synthesis of large DNA molecules in the 30–40 kb range but, in most cases, the constructs have not been published.

The recent report of the synthesis and assembly of an entire bacterial genome [6] indicates the substantial scientific and technological opportunities that could be seized when tools for designing new DNA sequences of this size (583 kb) become available. In this Opinion article, we argue that such tools will rely on abstract representations of the DNA sequence, enabling genetic engineers to analyze the properties of a design before its implementation in DNA. Recently, several research groups have recognized this problem as important and yet little has been published on this subject to date. Some of the ideas we want to put forward have been implemented in two different prototypic

Glossary

Formal language: a language that is defined by precise mathematical or machine-readable expressions.
Hardware Description Language (HDL): a computer language for formal description of electronic circuits. It can describe the circuit operation, its design and organization, and tests to verify its operation by means of simulation.
Non-terminal symbols or non-terminals: elements of a formal grammar that can be transformed by a production rule. They typically correspond to categories of terminal symbols such as 'verbs' or 'nouns' in natural languages or 'promoters' or 'genes' in genetic constructs.
Production rules: transformations rules that can be used to derive expressions generated by a formal language.
Semantic model: a set of rules that governs the meaning of expressions in a language. An expression semantic value can be used to translate it into another language.
Terminal symbols or terminals: symbols that cannot be transformed by any production rule. They correspond to the letters or words of the language. They are the sequences of the genetic elements used to design genetic constructs.
XCell: refers to an extension of living cells that includes both DNA-based abiotic and biological engineered systems.
XCell Description Language (XDL): would be a computer language for formal description of XCells. It would describe their operation, design and organization, and tests to verify its operation by means of simulation. XDL has not been implemented yet.

Corresponding author: Peccoud, J. (peccoud@vt.edu).

0167-7799/$ – see front matter © 2008 Elsevier Ltd. All rights reserved. doi:10.1016/j.tibtech.2008.06.003 Available online 5 August 2008

software applications, which are briefly described in the next two sections. We also give an indication of how these early-stage experiments could grow into a more mature conceptual framework. Considering that this field of investigation is in its nacency, we propose a forward-looking approach by articulating the need to create new representations of DNA sequences, and also by hypothesizing ways to develop the tools capable of manipulating these new representations.

BioJADE – a ground-breaking proof-of-concept

Electrical engineering models, terminology and representations have been used to represent natural regulatory networks [7] and more recently to describe and model synthetic systems [8]. Electrical Engineering has a strong tradition of using abstraction and symbolic notation to build incredibly complicated systems while maintaining abstraction layers between various levels of design and exposing the relevant information to the designers. A few years ago, one of the authors explored the possibility of adapting this methodology to the development of synthetic genetic systems (J.A. Goler, Masters Thesis, Massachusetts Institute of Technology, 2004) This effort led to the development of BioJADE, a graphical design tool to engineer genetic systems using graphical representations of circuit components. A comprehensive description of the capabilities of BioJADE, featuring numerous screenshots to illustrate its user interface, is available in an MIT Technical Report that can be downloaded from ftp://publications.ai.mit.edu/ai-publications/2004/AITR-2004-003.pdf.

BioJADE relies on genetic component prototypes. For instance, an inverter prototype encapsulates the function of inverting a signal. The word ‘prototype’ means that the inverter is defined irrespective of the gene coded by the expression cassette and the specific transcription factor used to repress its expression. An inverter prototype is an abstract representation of a large number of expression cassettes sharing similar functional properties. A large number of actual genetic components can implement a prototype inverter. BioJADE marks the first-ever implementation of abstract representations of genetic components and designs. Its development also required the development of a database of genetic parts that eventually morphed into the popular Registry of Standard Biological Parts (http://partsregistry.org) [9]. BioJADE also managed to translate abstract logical representations of genetic designs into DNA sequences that instantiate them into specific physical representations. BioJADE even includes a mechanism to simulate the phenotype of designs by transforming them into mathematical models of gene networks. These two compilation steps are an essential benefit of using abstract representation in real-world engineering projects. To articulate this compelling and comprehensive vision, BioJADE was not able to address important questions with the depth of analysis they deserved. The definition of genetic parts, their functional characterization, and the best ways of combining different parts in a synthetic DNA molecule are some of the problems that require dedicated research efforts, combining experimental and computational approaches. The BioJADE source code is available from http://www.biojade.org. Software developers interested in learning more about this application can recompile it in their own environment. However, BioJADE has never been officially released. In order for this prototype to become a practical solution to the design of new genetic systems, it is first necessary to characterize libraries of genetic parts. Developing a database of parts appears more challenging than initially anticipated [9] and the functional characterization of these parts is a complex problem actively debated in the community.

Linguistic description of synthetic DNA molecules

An alternative to the top-down approach proposed by BioJADE is the progressive building of increasingly more abstract representations of genetic constructs. Formal language theory (see Glossary) provides a framework to support this project. This branch of mathematics used in computer science and linguistics provides tools to develop formal descriptions of natural and computer languages [10]. A formal language is a set of strings over some alphabet. It is generally defined by a grammar that can generate all the sequences of symbols included in the language.

Because DNA molecules are most commonly represented by sequences of the four bases A, C, G, and T (instead of using their molecular structure for instance), it is natural to try to decipher the language of DNA molecules [11–13]; several authors have used this linguistic approach to describe the structure of natural DNA sequences [14–21]. We have recently proposed that this framework can be used to describe the structure of synthetic DNA molecules [22]. A formal grammar is completely defined by specifying a set of terminal symbols, a set of non-terminal symbols, and a set of production rules:

- The terminal symbols are the elements composing the strings of the language. In the case of DNA sequences, the terminal symbols are the basic DNA sequences that can be used to build more complex molecules.
- The non-terminal symbols can be regarded as categories of terminal symbols, such as genes, promoters, enhancers, multiple cloning sites, and other types of genetic elements available to design new DNA molecules.
- The production rules are rewriting rules that can be used to transform the string on the left hand side into the string on the right hand side of the rules.

Complex sequences of terminal symbols are generated by successive applications of production rules, allowing the rewrite of non-terminal symbols until no non-terminal symbols are left in the sequence. Figure 1 shows how a genetic construct comparable to a previously published bistable switch [23] can be designed by successive applications of the production rules of a general purpose grammar, generating the structure of gene expression cassettes in *Escherichia coli*.

The grammar used in Figure 1 is so simple that it does not express more than what is considered common knowledge of gene expression in prokaryotes; however, more sophisticated grammars capturing a more specialized biological expertise could be developed. By combining three bacterial repressors (the lactose repressor LacI, the phage

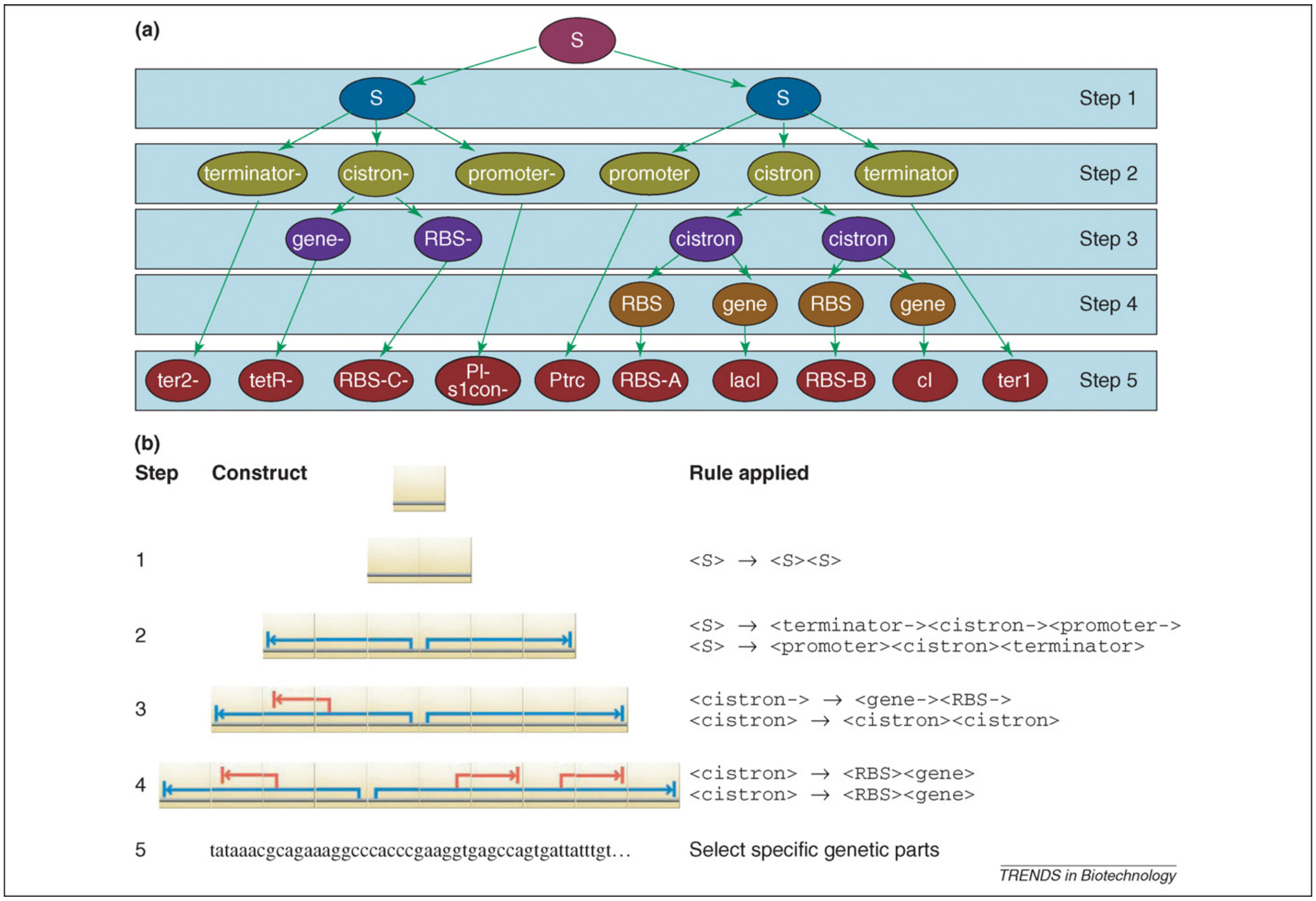
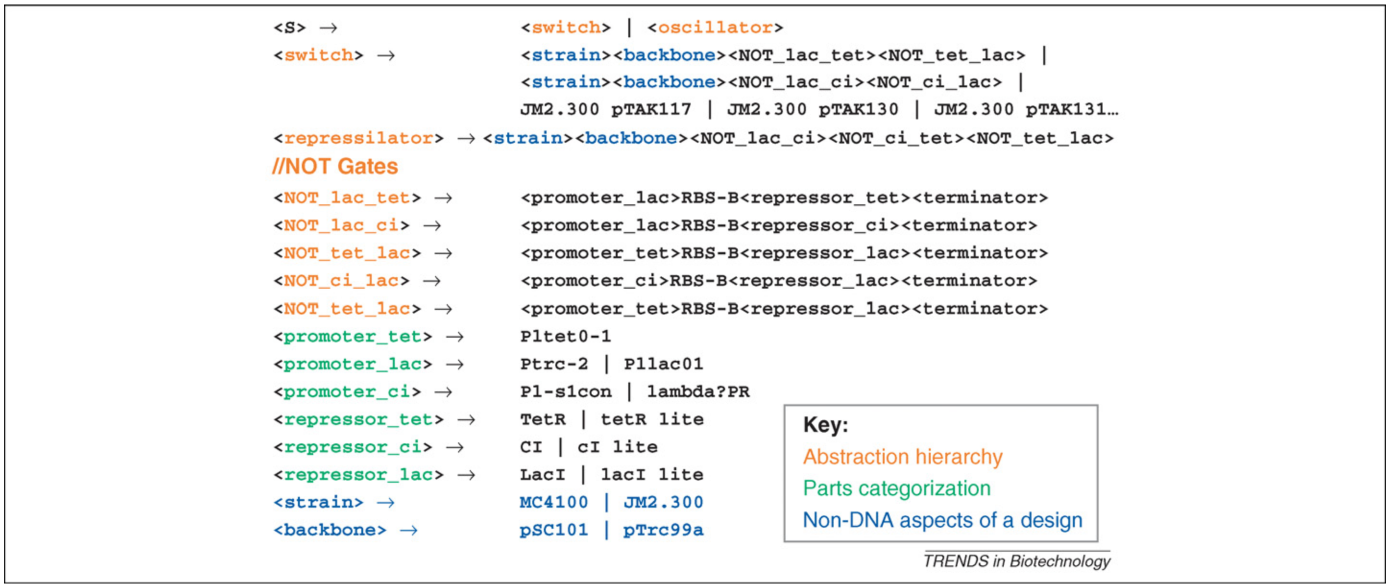


Figure 1. Using a grammar to design a genetic construct. **(a)** A tree is a convenient way of representing the successive application of production rules until a genetic construct is specified at the sequence level. The different steps of the derivation represented in (a) are expanded in **(b)**. DNA is represented by a gray line, RNA by a blue line, and proteins by a red line. The production rules applied at each step are indicated on the right-hand side of the figure. Categories of genetic elements are indicated by <>. Actual genetic parts that will make the final genetic constructs are not represented. The arrow, as in <A> → <B><C>, can be read as <A> is composed of<B> and <C>, for example, a cistron is composed of a RBS and a gene (<cistron> → <RBS><gene>).

lambda repressor cI, and the tetracycline repressor tetR) and their associated promoters it is possible to develop artificial gene networks with interesting properties. A bistable genetic switch [23] can be generated by combining two repressors in a configuration where they repress each other. An oscillator [24] can be developed by combining three repressors in a daisy chain arrangement in which the first repressor represses the second repressor, which represses the third repressor, which in turn represses the first repressor. Because it is generic, the grammar of Figure 1 could be used to generate such oscillators and switches but it can also generate constructs having a single gene, two genes that do not repress each other, or three genes wired in multiple ways leading to a large diversity of phenotypes [25]. Given that this generic grammar can also generate all these other constructs that are neither oscillators nor switches, it does not specifically guide a user in the actual implementation of such specialized constructs.

To facilitate the design of switches and oscillators, it is possible to develop a specific grammar for this purpose. A simplified version of this grammar is represented in Figure 2. Adding production rules to describe the structure of switches and oscillators is a way to start building an abstraction hierarchy in the grammar [26–28]. These rules add new levels of abstraction in the design process on top of the basic gene expression cassette. Switches and oscillators correspond to the highest level of abstraction. A second level is represented by a special kind of expression cassettes called NOT gates in which the expression of a repressor is under the control of another repressor. Defining ever increasing levels of abstraction is essential to enable the engineering of more complex systems by allowing engineers to focus their efforts at a very high-level without having to worry too much about the details of the actual implementation.

Using formal languages to represent genetic constructs is a major leap forward compared with working at the sequence level. This method has been implemented in GenoCAD [22], an experimental web-based application described in Box 1. Implementing this methodology can provide immediate benefits as described in Box 2. Nevertheless, this approach has a limited power of abstraction because it remains very problem-specific. The first generation of computer programming languages consists of machine codes somewhat comparable to the DNA sequence of a genetic construct. Assembly languages represent a second generation of programming languages [29].



```
<S> →                   <switch> | <oscillator>
<switch> →              <strain><backbone><NOT_lac_tet><NOT_tet_lac> |
                        <strain><backbone><NOT_lac_ci><NOT_ci_lac> |
                        JM2.300 pTAK117 | JM2.300 pTAK130 | JM2.300 pTAK131…
<repressilator> → <strain><backbone><NOT_lac_ci><NOT_ci_tet><NOT_tet_lac>
//NOT Gates
<NOT_lac_tet> →         <promoter_lac>RBS-B<repressor_tet><terminator>
<NOT_lac_ci> →          <promoter_lac>RBS-B<repressor_ci><terminator>
<NOT_tet_lac> →         <promoter_tet>RBS-B<repressor_lac><terminator>
<NOT_ci_lac> →          <promoter_ci>RBS-B<repressor_lac><terminator>
<NOT_tet_lac> →         <promoter_tet>RBS-B<repressor_lac><terminator>
<promoter_tet> →        Pltet0-1
<promoter_lac> →        Ptrc-2 | Pllac01
<promoter_ci> →         Pl-s1con | lambda?PR
<repressor_tet> →       TetR | tetR lite
<repressor_ci> →        CI | cI lite
<repressor_lac> →       LacI | lacI lite
<strain> →              MC4100 | JM2.300
<backbone> →            pSC101 | pTrc99a
```

Key:
Abstraction hierarchy
Parts categorization
Non-DNA aspects of a design

TRENDS in Biotechnology

Figure 2. A specialized grammar for switches and oscillators. Switches and oscillators can be recognized as specific categories of constructs by the definition of corresponding non-terminals. Because both switches and oscillators rely on cassettes expressing a repressor under the control of another repressor, different non-terminal symbols can be defined to describe these special expression cassettes. For instance <NOT_lac_tet> corresponds to a cassette in which the expression of tetR is repressed by lacI. This grammar also requires introducing new non-terminals more specific than the generic <gene> and <promoter> categories used in the generic descriptions of expression cassettes. Once these cassettes are properly defined, a simple pattern such as <NOT_lac_tet><NOT_tet_lac> represents a switch. However, for these constructs to be functional it is important to ensure that the bacterial host does not express any of the three repressors. Similarly, the plasmid backbone used to insert the construct might impact the phenotype. It is therefore desirable to introduce in the grammar non-terminal symbols referring to the strain and backbone in which the constructs can be inserted.

Although they are easier to read and edit than machine code, which they can be translated into, they still remain machine-specific – just like the grammars described so far remain specific to a biological organism or to a biotechnology application. A high-level and portable language somewhat comparable to the third generation of programming languages could overcome these limitations.

Toward an XCell description language

BioJADE and GenoCAD both rely on graphical user interfaces easily accessible to potential users with limited training. These visual languages can hide or expose the underlying complexity of the constructs as required for a given design context. Complementing and correlating with these representations, engineers are also likely to design their systems using structured sets of text-files developed via methodologies similar to those used for developing complex computer programs. These textual representations should also be well suited for processing by automated tools.

Because synthetic DNA molecules will lead to products having little in common with natural living organisms [8,30], the term XCell is used to refer to DNA-based abiotic and engineered biological systems. XCells should be described using a formal language called XCell Description Language (XDL), which is comparable to Hardware Description Languages (HDL) used in electronics [31]. XDL will be different from a traditional programming language because it will not describe a sequence of operations in a microprocessor but the molecular interactions coded by the DNA molecules. Like BioJADE, XDL should be capable of providing a behavioral description of the system independent of its physical implementation. However, it might also be desirable to introduce some high-level structural information, capable of expressing that different parts of a system should be implemented in different plasmids, different types of living cells, or inserted in different locations in the genome of the host cell. It will also be particularly important that XDL is capable of expressing physical and chemical properties not only of the DNA but of the abiotic or cellular matrix in which it operates.

It will be necessary to develop compilers capable of transforming the XDL code into mathematical models that can be simulated or verified. This is important to establish whether a particular XDL program meets user specifications, or to prove that two XDL programs are equivalent. The engineering of artificial gene networks still requires using quantitative dynamical models. However, it is possible that artificial genetic systems relying on binary

Box 1. Software: GenoCAD

GenoCAD (http://www.genocad.org) is an experimental web-based application allowing users to design complex genetic constructs generated by languages built into the system [22]. GenoCAD relies on an iconic representation of the genetic constructs similar to the one used in Figure 1. The construct is developed by clicking on links under each icon that correspond to various transformation options consistent with the grammar. GenoCAD also provides a sequence verification feature that determines if a user-provided sequence is consistent with a specific grammar.

Ongoing research efforts aim to complement the syntactic models of a DNA sequence with semantic models that could be used to transform the sequence into mathematical models, predicting the construct phenotype. Specifically, a compiler will translate the structural description of genetic constructs into systems biology markup language (SBML) [39] to use existing simulation engines [40,41] to analyze the expected behavior and phenotype of the construct.

Box 2. Benefits of linguistic representations of DNA sequences

Software wizard

A formalized language makes it possible to develop software wizards guiding users through the process of developing new DNA molecules. Compared with sequence editors allowing an unstructured design of new DNA sequences [32], a grammar constrains users and could prevent some time-consuming mistakes. The effort of defining parts libraries and rules describing the valid combinations of these parts is offset by the time saved avoiding mistakes such as forgetting to insert an essential genetic element or making an error in cutting and pasting a sequence.

Automatic validation

Another benefit is the automatic validation of genetic constructs not developed with a grammar-constrained sequence builder. In this case, the construct can be inconsistent with the grammar describing a specific class of constructs. Several algorithms exist in computer science to prove if a particular sequence can be generated by a given grammar [42]. A gene synthesis company or a vector construction group could use this approach to verify automatically the structure of the sequences designed by their customers. For example, such an organization could describe the protein expression vectors it supports using a custom language. Orders could then be automatically verified and nonconforming orders would be flagged for review by an operator before inclusion in the production pipeline.

Division of labor

The different aspects of the language definition could involve scientists and technicians with different scientific expertise. The definition of the production rules is the most difficult aspect of such a project. It requires a solid vector design expertise and an ability to formalize this expertise. The parts library curation can be assigned to curators. Once the grammar and the parts libraries have been set up, others can quickly design valid constructs because the framework allows them to benefit from the expertise of the team who designed the grammar used by the organization.

Construct optimization

Metrics can be associated with DNA sequences. An obvious one is, for example, the sequence length. A less trivial metric is the number of patents associated with a sequence. It is possible to generate automatically all the constructs that have a similar structure to find an optimal sequence maximizing one or more figures of merit.

switches could eventually be represented by Boolean systems, allowing their designer to benefit from the wealth of methods available to analyze digital systems.

Once a design or XDL program has been validated, another compiler will translate this abstract representation into a physical implementation. The output of this process could be a single DNA sequence but could also be broken down into the sequence of multiple constructs and could include information describing the cellular environment in which they should be transfected, or the biochemical environment allowing their operation *in vitro*. Mapping the high-level description into a physical implementation is an important step because it enables the experimental validation of XDL designs. Generally, an infinite number of DNA sequences could implement a specific design, whereby the compiler would have to generate the optimal sequence or at least an acceptable sequence. Different compilers would be available to translate designs for different target organisms. The back-translation of amino acid sequences into DNA for expression of proteins in different hosts [32–34] hints at the complexity of this problem. Assuming that each amino acid can be encoded by an average of three different codons, there are $\sim 3^{100}$ or 5.10^{47} different DNA sequences coding for the same 100 amino-acid protein. Analysis of natural sequences has uncovered some of the parameters that can be considered when designing a gene for heterologous expression. They include codon usage, codon pair frequency, GC content, the presence of direct repeats, mRNA secondary structure and cryptic splice sites, and the presence or absence of certain restriction sites [33]. If many tools are available to perform a basic back-translation of an amino-acid sequence, the optimization of the resulting DNA sequences still remains an educated guess. DNA2.0, a leading gene synthesis company, has recently received funding from the National Science Foundation to identify the crucial variables affecting protein expression and to build machine-learning algorithms capable of predicting their interactions (http://www.dna20.com/index.php?pageID=139). This type of work requires the systematic characterization of large numbers of physical implementations of a design.

Conclusion

To take full advantage of large-scale DNA synthesis for engineering complex living or abiotic systems, it is necessary to develop abstract representations of these systems. Abstraction is key to the development of a hierarchy of descriptions that can either be used to build complex systems from the ground up by reusing existing components, or to decompose a complex engineering problem into smaller more manageable components. BioJADE and GenoCAD provide illustrative examples of the patterns that characterize Electronics Design Automation, particularly synthesis, validation, extraction, and design rule checking (Box 3). Furthermore, these two tool systems and supporting design languages span different levels of abstraction, within and across potential design and engineering domains. Among these are core concepts such as sequence, gene, and genome.

In the case of biologically derived engineering, the current phase of early discovery and development of simple device-like mechanisms has also begun to expose patterns in evolved architectures and protocols of substantially complex systems. Many of these systems seem coupled across levels of abstraction and physical scales in ways that engineers tend to avoid. Deliberately designing for hierarchy and decomposition is essential for comprehension by human minds, both individually and as collective, coordinated teams.

Now that it is possible to synthesize the entire genome of a bacterium, it is time to develop a means to represent this object in a more compact and meaningful form than simply the 582 970 bases of its DNA sequence. We are coming to a point where the lack of language runs the risk of inhibiting further progress. By stepping back from the sequence, we can gain a better perspective of the technical and scientific challenges facing the research community today. In particular, we need to identify the mechanisms of gene expression well enough to map abstract representations into acceptable physical implementation. Although formal methods are needed to better represent the structure and

Box 3. Comparisons to Electronic Design Automation

Electronic Design Automation (EDA) is an engineering discipline and a category of tools for designing and producing electronic systems ranging from printed circuit boards to integrated circuits. EDA often includes computer-aided design (CAD), engineering, and manufacturing but, beyond these, the more general emphasis is on Design Automation. EDA has rapidly increased in importance in conjunction with Moore's Law describing the scaling of semiconductor technology. Not only are modern processor and application-specific integrated circuit designs too cumbersome and complex to be designed manually, but EDA forms the backbone of a large, diverse, and multidisciplinary ecosystem that permits necessary specialization while maintaining coherent end-to-end integration.

Early integrated circuits were designed by hand, and the lithographic masks needed to produce them were laid out manually. Some of the first automation tools generated magnetic tapes of geometric data for a photoplotter – much like drafting. By the mid-1970s, automation reached beyond drafting into design, marking the start of the eventual divergence of EDA from more traditional CAD tools used in civil, architectural, and mechanical engineering. By 1980, concepts of frameworks and design languages were established in academia, appearing in commercial offerings by mid-1980s from companies that remain dominant players today.

EDA tools are often classified by the primary engineering domains involved along the path from high-level architecture to manufacturing: architecture, logic, circuit and layout. Each domain focuses on structural and behavioral concerns – static and dynamic perspectives – within the same abstract representation (Figure I). Furthermore, certain classes of tools connect representations from one or more domains. The logic of ones and zeros flowing through connected gates are related to circuits of electrical voltages and currents flowing through devices and wires. Regardless of which domains are involved, common functions form the basis of most design methodologies: for example, synthesis, simulation, validation, extraction, equivalence checking, and design rule checking. Categorizing via these abstract functions to represent the design process as a systems engineering problem exposes potential analogs with emerging tools in biology-derived domains.

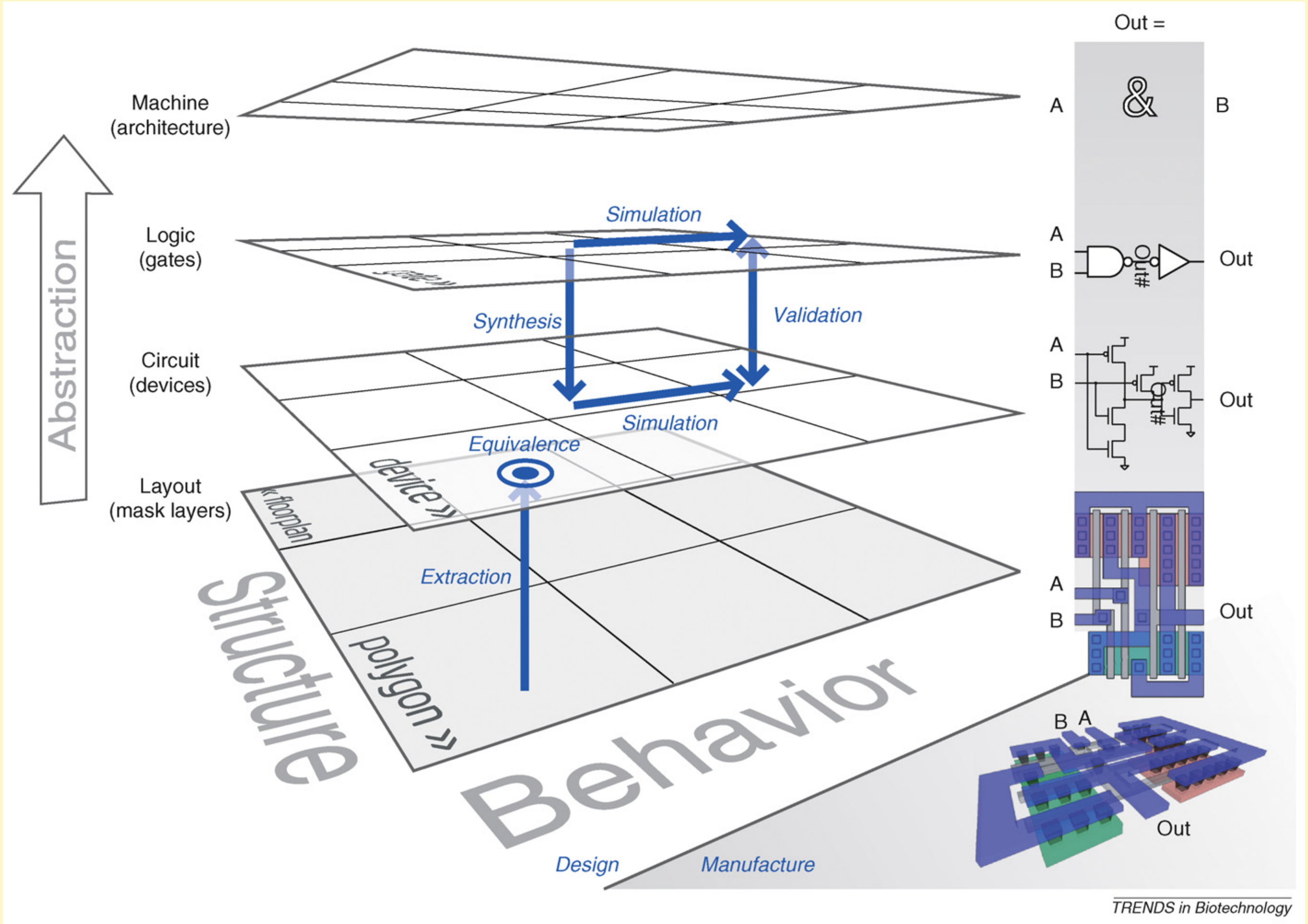


Figure I. Schematic representation of the EDA abstraction hierarchy.

behavior of genetic systems, they are far from sufficient to ensure the predictable engineering of DNA-based products. Understanding the physico-chemical processes that these languages represent is equally important. By designing artificial gene networks, it has become possible to gain a much deeper understanding of the molecular noise affecting gene expression mechanisms [35,36]. By developing abstract representations of the complex relationships between the sequence of a DNA molecule and its predicted performance, engineers will provide the life science community with more refined models of the relationships between genotype and phenotypes [37,38] and their underlying molecular mechanisms. These models will lead to the development of tools used to redesign simple genomes or to create new ones. It is unclear when this technology will lead to a new generation of biotechnology products, but the lessons learned along the way will help us revisit the way we understand the genetics of natural organisms.

Acknowledgements

By introducing abstract representation of DNA sequences, Tom Knight played an essential role in the emergence of synthetic biology as a discipline. We are indebted to Isabelle Hakala for helping us find the origins of the term ‘genetic engineering’. The opinion expressed in this article is the product of countless hours of stimulating discussions with Yizhi Cai, Kent Vander Velden, Michael Czar, William Baumann, and Claes Gustafsson. Emily Berisford edited this opinion to make it more accessible to non-specialists. This work was partly funded by the Virginia Commonwealth Research Initiative.

References

1 Tatum, E.L. (1959) A case history in biological research. *Science* 129, 1711–1715

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

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

4 Chan, L.Y. *et al.* (2005) Refactoring bacteriophage T7. Mol. Syst. Biol. 1, 2005.0018.

5 Kodumal, S.J. *et al.* (2004) Total synthesis of long DNA sequences: synthesis of a contiguous 32-kb polyketide synthase gene cluster. *Proc. Natl. Acad. Sci. U. S. A.* 101, 15573–15578

6 Gibson, D.G. *et al.* (2008) Complete chemical synthesis, assembly, and cloning of a *Mycoplasma genitalium* genome. *Science* 319, 1215–1220

7 McAdams, H.H. and Shapiro, L. (1995) Circuit simulation of genetic networks. *Science* 269, 650–656

8 Seelig, G. *et al.* (2006) Enzyme-free nucleic acid logic circuits. *Science* 314, 1585–1588

9 Peccoud, J. *et al.* Targeted development of registries of biological parts. *PLoS ONE* 3, e2671

10 Slonneger, K. and Kurtz, B.L. (1995) *Formal Syntax and Semantics of Programming Languages: a Laboratory Based Approach,* Addison-Wesley Pub. Co

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

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

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

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

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

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

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

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

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

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

21 Chiang, D. *et al.* (2006) Grammatical representations of macromolecular structure. *J. Comput. Biol.* 13, 1077–1100

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

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

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

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

26 Andrianantoandro, E. *et al.* (2006) Synthetic biology: new engineering rules for an emerging discipline. Mol. Syst. Biol. 2, 2006.0028

27 Arkin, A.P. and Fletcher, D.A. (2006) Fast, cheap and somewhat in control. *Genome Biol.* 7, 114

28 Heinemann, M. and Panke, S. (2006) *Synthetic Biology – Putting Engineering into Biology,* Bioinformatics

29 Louden, K.C. (2002) *Programming Languages, Principles and Practice,* Brooks/Cole

30 Forster, A.C. and Church, G.M. (2007) Synthetic biology projects *in vitro*. *Genome Res.* 17, 1–6

31 Sangiovanni-Vincentelli, A. (2003) The tides of EDA. *IEEE Des. Test Comput.* 20, 59–75

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

33 Gustafsson, C. *et al.* (2004) Codon bias and heterologous protein expression. *Trends Biotechnol.* 22, 346–353

34 Richardson, S.M. *et al.* (2006) GeneDesign: rapid, automated design of multikilobase synthetic genes. *Genome Res.* 16, 550–556

35 Maheshri, N. and O’Shea, E.K. (2007) Living with noisy genes: how cells function reliably with inherent variability in gene expression. *Annu. Rev. Biophys. Biomol. Struct.* 36, 413–434

36 Raser, J.M. and O’Shea, E.K. (2005) Noise in gene expression: origins, consequences, and control. *Science* 309, 2010–2013

37 Peccoud, J. *et al.* (2004) The selective values of alleles in a molecular network model are context dependent. *Genetics* 166, 1715–1725

38 Cooper, M. *et al.* (2005) Gene-to-phenotype models and complex trait genetics. *Aust. J. Agric. Res.* 56, 895–918

39 Hucka, M. *et al.* (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19, 524–531

40 Hoops, S. *et al.* (2006) COPASI—a COmplex PAthway SImulator. *Bioinformatics* 22, 3067–3074

41 Griffith, M. *et al.* (2006) Dynamic partitioning for hybrid simulation of the bistable HIV-1 transactivation network. *Bioinformatics* 22, 2782–2789

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