OPEN ACCESS

BIO-complexity

Research Article

A *Stylus*-Generated Artificial Genome with Analogy to Minimal Bacterial Genomes

Douglas D. Axe*, Philip Lu, and Stephanie Flatau

Biologic Institute, Redmond, WA, USA

Abstract

The difficulty of explaining evolutionary innovation on a scale that would account for the functional diversity of life and its components continues to dog evolutionary theory. Experiments are shedding light on this, but the complexity of the subject calls for other approaches as well. In particular, computational models that capture some aspects of simple life may provide useful proving grounds for ideas about how evolution can or cannot work. The challenge is to find a model 'world' simple enough for rapid simulation but not so simple that the real thing of interest has been lost. That challenge is best met with a model world in which *real*-world problems can be solved, as otherwise the connection with real innovation would be in doubt. *Stylus* is a previously described model that meets this criterion by being based on one of the most powerful real-world problem-solving tools: written language. *Stylus* uses a genetic code to translate gene-like sequences into vector sequences that, when processed according to simple geometric rules, form patterns resembling penned strokes. These translation products, called *vector proteins*, are functionless unless they form legible Chinese characters, in which case they serve the real function of writing. This coupling of artificial genetic causation to the real world of language makes evolutionary experimentation possible in a context where innovation can have a richness of variety and a depth of causal complexity that at least *hints* at what is needed to explain the complexity of bacterial proteomes. In order for this possibility to be realized, we here provide a complete *Stylus* genome as an experimental starting point. To construct it we first wrote a concise description of the *Stylus* algorithm in Chinese. Using that as a proteome specification, we then constructed the *Stylus* genes to encode it. In this way the *Stylus* proteome specifies how its encoding genome is decoded, making it analogous to the gene-expression machinery of bacteria. The complete 70,701 base *Stylus* genome encodes 223 vector proteins with 112 distinct vector domain types, making it more compact than the smallest bacterial genome but with comparable proteomic complexity for its size.

Cite as: Axe DD, Lu P, Flatau S (2011) A *Stylus*-generated artificial genome with analogy to minimal bacterial genomes. BIO-Complexity 2011(3): 1-15. **doi:**10.5048/BIO-C.2011.3

Editor: Jed Macosko

Received: May 26, 2011; **Accepted:** September 29, 2011; **Published:** October 24, 2011

Notes: A *Critique* of this paper, when available, will be assigned **doi:**10.5048/BIO-C.2011.3.c.

* Email: daxe@biologicinstitute.org

INTRODUCTION

The study of molecular evolution is complicated in some respects by the complexity of genetically encoded proteins and their functions. In particular, the length of a typical biological protein chain makes it but one of an extraordinarily large number of *possible* chains that differ in their amino acid sequences. The fact that no real process can sample anything but a minuscule fraction of these sequence possibilities means that many topics of fundamental importance, such as the structure of fitness landscapes or the sparseness of function in protein sequence space, must be explored through inferences from a relatively small set of observations. Consequently, any tool that facilitates the making, testing, and refining of such inferences should be seen as a welcome addition to the available modes of inquiry.

Computational models have long been considered important in this respect because they enable quantitative sampling on a scale that may, in some cases, exceed what can be achieved experimentally. For such studies to be relevant to biology, however, the model they implement must represent one or more aspects of life correctly. This intuitive principle can be expressed more rigorously in terms of classes. Specifically, any model system that shares a particular property with living systems becomes a co-member (with the living systems) of the class of all systems having that property. Such models are important from the perspective of theoretical biology because whatever we reason to be true of a class must be true of all its members, and (conversely) whatever is shown not to be true of a member is shown not to be generally true of the class. Consequently, models designed in such a way that they must be classified with life in important respects are necessarily of interest to the theoretician, whose aim is to understand generalities rather than particulars—not what *this particular thing* does per se, but rather why *things like it* must behave like it in certain respects.

Stylus is a model of that kind that has been fully implemented with open-source software[1] [1]. The motivation for *Stylus* was the recognition that prior models used to study evolutionary innovation did not adequately represent the complex causal connection between genotypes and phenotypes. Although very little is known about this connection as it applies to anatomical features in complex organisms, a great deal more is understood if we narrow the focus to molecular features in simple organisms. Specifically, we know how genes encode protein chains in bacteria, and we have at least a conceptual understanding of how these chains fold to form functional proteins and protein complexes. With many thousands of published bacterial protein structures, we also have a reasonably good picture of both the variety of fundamentally different protein forms in simple life and the relationship between these forms and their biological functions. What we lack is the combination of understanding and computing power that would enable accurate general prediction either of protein structures from their sequences or of protein functions from their structures. *Stylus* in no way fills these gaps, but rather it provides an artificial world in which they are absent, and in which the causal connection between gene sequences and their functions is analogous in many respects to the real connection. That analogy, in combination with the computational tractability of the *Stylus* world, makes it possible to tackle the *analogs* of important questions that cannot easily be tackled in the laboratory.

For the sake of clarity and simplicity, we have borrowed the vocabulary of biology to describe aspects of the artificial *Stylus* world that correspond to aspects of the real world. The advantage of this is that we avoid the communication challenges that a completely new vocabulary would bring, but we acknowledge some disadvantages as well. One of these is the possibility of what might be called 'world confusion.' That is, comments made about the *Stylus* world might, if removed from their intended context, be misconstrued as comments about biology. Another is that identity of terms might be interpreted as a presumption of genuine equivalence between the things to which these terms are applied. We intend to avoid confusion in both respects, but we recognize that some care will be needed to ensure this. To be clear, the *Stylus* world we describe in this paper is entirely non-biological. It has been constructed in a way that captures some of the basic *concepts* of molecular biology, but it makes no use of biological *data* of any kind.

Because *Stylus* is substantially unlike other models, we will reiterate many of its key aspects in this paper. One of these is its use of a genetic code. Briefly, *Stylus* uses gene-like sequences of the four letters associated with DNA (A, C, G, and T) to encode graphical constructs that are built by connecting vectors end-to-end to form complex two-dimensional paths. The encoding scheme used for this is deliberately analogous to the real genetic code. That is, of the 64 possible 'codons' (letter triplets), 61 map to a fixed set of twenty coplanar vectors that vary in length and direction, with the remaining three (TAA, TAG, and TGA) signaling chain termination. By analogy, we refer to a complete vector path encoded in this way as a *vector protein*.

Vector proteins are of course very unlike real proteins in substance, the latter being dynamic molecular constructs while the former are static graphical constructs. Yet *Stylus* endows these graphical constructs with interesting similarities to their molecular counterparts by uncovering and exploiting a pre-existing analogy—the analogy between the set of characters used in Chinese writing and the set of protein structures used in life [1]. Specifically, vector proteins are drawn objects that may function as legible Chinese characters if they are suitably formed. *Stylus* uses a database of ideal character forms, called *archetypes*, as the basis for calculating the geometric likeness of a given vector protein to a specified Chinese character. The calculation is rapid enough (often sub-millisecond) to be performed millions of times on a single processor core within a modest experimental timeframe, making it possible to tackle a wide variety of interesting problems in this model world (see Figure 1). Other models may match or surpass this computational speed, but *Stylus* is unique in its use of real function that maps well to molecular biology. It therefore represents a significant advance in the field of evolutionary modeling.

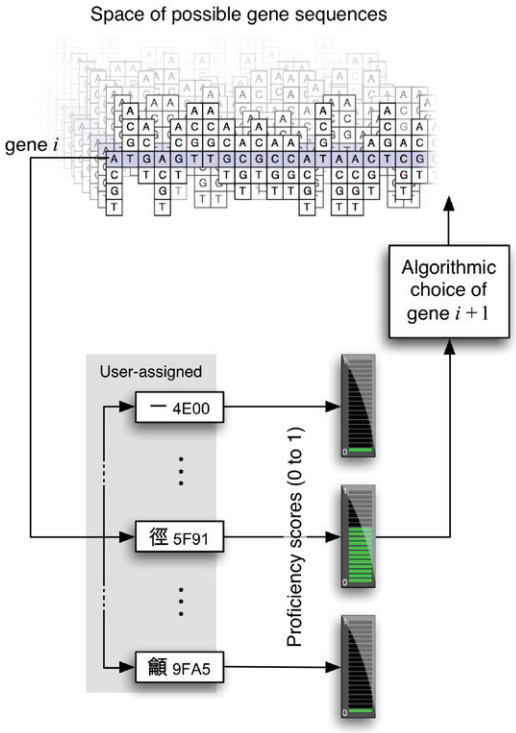

Figure 1: *Stylus* as a general-purpose engine for gene scoring. At the core of *Stylus* software is an algorithm that quantifies the likeness of a given vector protein to a specified Chinese character. This numerical result, called the proficiency score, is calculated as a double-precision floating-point number ranging in value from zero to one. Any number of scoring cycles may be executed in one experiment, starting from an initial gene and an initial assignment of drawn lines to character strokes. A variety of algorithms can be used to produce each new gene sequence after the prior one has been scored. Some of these simulate natural selection, but other algorithms can be used as well to suit the purpose of the experiment. As depicted, a single gene sequence can, in principle, be evaluated with respect to any of the 20,000+ Chinese characters (provided each of its strokes can be assigned). In practice, proficient vector proteins achieve much higher scores with respect to one character (the one they represent well) than others. Functional specificity therefore has a structural basis in the *Stylus* world, just as it does in the real world.
doi:10.5048/BIO-C.2011.3.f1

[1] See https://github.com/biologic/stylus.

One thing *Stylus* does not offer, though, is minimalist simplicity. Indeed, its main advantage—that it captures well some of the messy complexity of molecular biology—might also be a distinct disadvantage, depending on the intended application. To help potential users of *Stylus* understand where it fits within the existing assortment of evolutionary artificial-world models, the remainder of this section compares the most prominent alternatives.

Our discussion of the various models will be framed around the familiar causal series that informs our understanding of proteins:

$$\textit{Sequence} \rightarrow \textit{Structure} \rightarrow \textit{Function}. \qquad (1)$$

The central importance of this series for molecular evolution stems from the fact that it is closely coupled to another causal series, equally familiar:

$$\textit{Genotype} \rightarrow \textit{Phenotype}. \qquad (2)$$

Both are simplifications (even for bacteria) but because they represent key aspects of the real picture, they are of particular importance for evolutionary modeling. Interestingly, despite the apparent simplicity of Series 1 above, very few models have the causal structure it describes (Figure 2). Many evolutionary models include two of the three key property categories (sequence, structure, or function), but models representing the full series in proper causal relation are conspicuously rare.

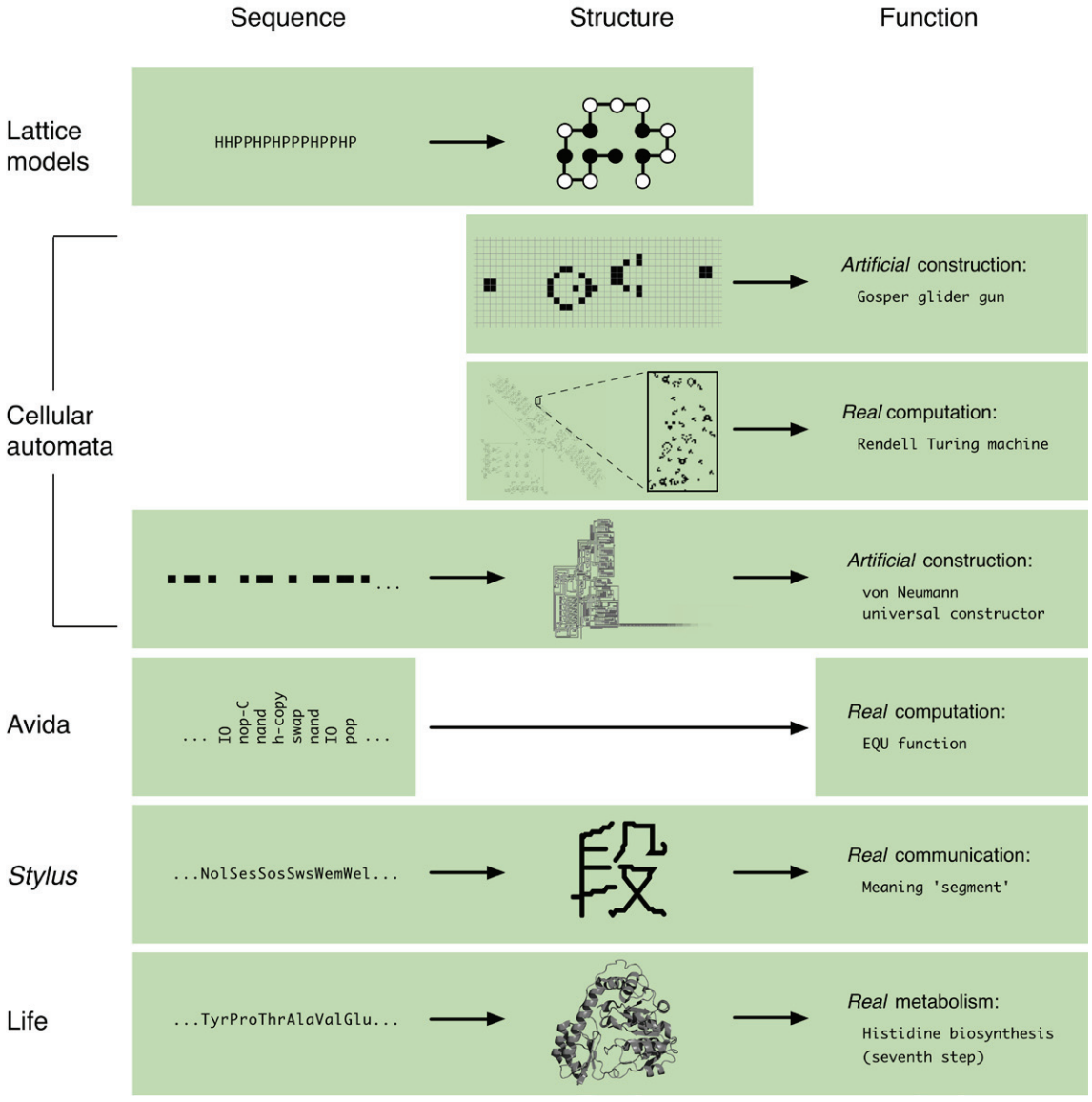

Figure 2: Presence or absence of *Sequence* → *Structure* → *Function* causation in model worlds. See the main text for discussion. The lattice example depicted (from Hirst [2]) uses two bead types, *H* signifying 'hydrophobic' (black) and *P* signifying 'polar' (white). The top two examples of cellular automata are similarly binary, since they use the transition rules of Conway's *Game of Life* [3], where pixels are either *on* (filled) or *off* (open). Animated illustrations of these two examples are available as supplementary CDF files (*Gosper glider gun*, Supplement S1 [4]; *Rendell Turing machine*, Supplement S2 [5]). To view these animations, install the free Wolfram CDF player (http://www.wolfram.com/cdf-player/). **doi:**10.5048/BIO-C.2011.3.f2

One important class of prior models is known as *lattice models*, referring to the kind of structure they describe. In a lattice world, 'beads' of a few different types (typically two) are joined in sequence to form chains. The beads may occupy any of the discrete points on a small two- or three-dimensional lattice (Figure 2). A chain is a sequence of beads with neighbors in the sequence constrained to occupy neighboring points on the lattice, without overlaps. Of the many possible conformations for a given chain sequence, the 'native' conformation would be (if it exists) the one that is uniquely stable according to the simple physical model used. This explicit treatment of conformational alternatives makes lattice models particularly suitable for studying protein folding, but they have also been used to study protein evolution [2, 6, 7]. However, a considerable limitation of these models for evolutionary applications is that, unlike real proteins, these chains have no function. That is, there are no actual problems that lattice structures are capable of solving. To get around this, the notion of function is simply replaced by structural features that *correlate* with functionality in real proteins, like structural stability [7] or the presence of open clefts or pockets ([2, 6]; see Figure 2). This may be useful for studying structural constraints as such, but it is less clear how useful it is for studying the actual functional constraints within which evolution must work.

Cellular automata, where fixed transition rules are applied repeatedly to transform a grid of active pixels ('cells'), form another class of models that have been used to study evolution [8]. Although these models can incorporate functions, the most easily attained functions are themselves artificial, meaning that they are confined to the pixel worlds in which they occur. For example, a pixel construct known as the *Gosper glider gun* (Figure 2) produces a steady stream of small pixel objects called 'gliders' when the pixel states are updated according to the transition rules of Conway's *Game of Life* [3]. Viewing those states in succession as a movie [4] gives the impression of manufacturing, in that the glider gun produces gliders with the regularity we associate with assembly lines. But of course, there are only two elementary events in that artificial world—a pixel going from *off* to *on*, or the reverse—and a glider is merely a particular group of five pixels in the *on* state. The gliding function is therefore a modest extension of the most elementary phenomena that exist in that world, making the production of gliders qualitatively unlike the kind of manufacturing that interests us in the real world.

This points to a key principle for evolutionary modeling. If the ultimate objective is to explain how life acquired such remarkable solutions to real-world problems, then we need to look for models that can be used to study the solution of real-world problems. Otherwise, whatever we might learn from models lacking this capacity, we are left to wonder what it has to do with real-world problem solving. Interestingly, real problems—computational ones—*can* be solved by cellular automata that incorporate (in structures made of active pixels) all the necessary components of a computational device[2] (see Rendell Turing machine, Figure 2). As you might expect, though, automata that meet this condition are necessarily complex, making them cumbersome subjects for most evolutionary studies. However important these intricate pixel machines may be as computing abstractions, the huge amount of real computation that would be needed to produce and test large numbers of functional variants limits their utility for evolutionary simulation.

Explicit computational models, like Avida [9], offer a considerable performance boost in this regard, but not without their own drawbacks. Like the pixel computers in cellular worlds, organisms in the Avida world are capable of versatile computation, but unlike their pixel counterparts, Avida organisms are endowed with computational hardware as a given. Instead of focusing on hardware, the Avida world focuses on sequential instructions with the idea that these are analogous to genomes [9]. So, sequences are connected to functions in that world, but the structures actually needed for this connection lie outside it, as indicated by the gap in the center column of Figure 2. Like the previous examples, then, Avida includes only part of the causal series represented above (Series 1).

However, Avida's omission of structure masks what is arguably an even more significant departure from the causal pattern of life. Living things build the complex from the simple, a pattern that holds from the molecular level all the way up to the level of the whole biosphere.[3] Avida inverts this pattern by using the complex to build the simple, making it less life-like in this important respect than the other models. The bead in a lattice model, the pixel in a cellular model, and the vector in *Stylus* are all simple in that they can be fully described in terms of their stand-alone properties and their interactions with similarly simple neighbors. The same can be said of the amino acids used to construct proteins. All of these simple things can be used to construct much more complex things, but they are fully intelligible as simple things *in themselves*, without reference to such complexity. Serine, for example, has its particular chemical and physical characteristics as an amino acid, none of which necessitate or presuppose the existence of the much larger and more complex protein molecules that incorporate it. This contrasts sharply with the things—instructions—that are arranged sequentially to build Avida genomes. Since instructions for a computing device make explicit reference to the device (by specifying operations on read, write, and flow heads, input and output buffers, data registers and stacks, etc.) they are only intelligible as part of a full device specification, and they only *work* if that specified device is itself provided and configured to begin processing instructions. This makes instructions complex things *in themselves*, whether or not they are put to any good use. Consequently, because they are the fundamental building blocks of the Avida world, that world is arguably unsuitable as a tool for advancing our understanding of the real living world, where the building blocks are strikingly simple.

All the difficulties notwithstanding, the full causal series above (Series 1) has been captured in a cellular automaton,

[2] See http://rendell-attic.org/gol/tm.htm, and supplemental animation [5].

[3] The fact that some organisms eat others is not an exception to this. Owls, for example, are not built from mice but rather from the simple molecular nutrients liberated by digesting mice (or other animals).

namely the 'universal constructor' elegantly conceived by von Neumann [10] and impressively implemented by Pesavento [11] (Figure 2). Like pixel-based computing devices (e.g., the Rendell Turing machine of Figure 2), a universal constructor is a complex pixel construct,[4] but in this case one that can build any stable pixel pattern according to instructions written on a pixel 'tape.' This makes it analogous to the real machines used in computer-aided manufacturing (CAM), in that both are programmable manufacturing devices. In fact, the relative simplicity of pixel worlds enables universal constructors in those worlds to fully replicate themselves, a feat that is well beyond human technology in the real world. Along with this advantage, though, comes the common limitation of functions in artificial worlds: *Since artificial worlds can easily be constructed in a way that greatly facilitates particular artificial functions, their relevance to real-world problem solving will remain questionable unless they can be used for just that—solving real problems.*

With this background on evolutionary modeling, the advantages of *Stylus* are readily apparent. It fully captures the causal relationships of Series 1, and it does so with the real-world function of written communication. Since language is arguably the most powerful and versatile tool in existence for real-world problem solving, this paves the way for evolutionary experimentation in a model world where functional solutions can have a richness of variety, a range of complexity, a depth of hierarchy, and a practical reality reminiscent of those seen in proteomes. In addition to these benefits, *Stylus* has the pedagogical advantage that language, unlike biology, is familiar to everyone.

The first objective of this work, then, is to realize some of these possibilities by constructing a *Stylus* text that functions like a simple bacterial proteome. Along with this, the second is to make *Stylus* more accessible to new users by making the full set of genes encoding this proteome (a complete *Stylus* genome) freely available as resources for using the *Stylus* world to frame and test ideas that may advance our understanding of the real world.

APPROACH

The first of the above objectives requires us to identify an important aspect of proteomic function in simple life that lends itself to the linguistic analogy on which *Stylus* is based. In considering what this should be, we begin with the observation that the most striking thing accomplished by *all* bacteria is self-replication.

Real-world self-replication may be viewed as having two facets. One of these is self-description, which means carrying a representation of the self that can be implemented to build a self-replica. The other is the implementation itself—the work of building a replica in a natural setting according to the self-description. So far, only life exhibits both of these facets. Computational models commonly exhibit the first facet, but lack (among other things) the real-world interface needed for the second. The universal constructor of Figure 2, for example, depends on an actual computer (something it most definitely cannot construct) for its implementation. Similarly, while robots can implement computational algorithms in the real world, they depend on complex physical structures in order to do this, which inevitably complicates their replication.[5]

Accepting that genuine self-replication is beyond the reach of synthetic systems, we have narrowed our focus to self-description. The next question is, what does self-description look like in simple life? As a starting point, we take a reasonable answer to be that a bacterial cell's chromosome and its gene-expression apparatus are jointly self-descriptive. Although it is common to think of the genome itself as being a 'blueprint' for the cell, genomic sequences are descriptive only when properly decoded, and this decoding is accomplished by the actions of many proteins. In other words, the genome has to be *interpreted* in order for it to serve as a description of the proteome. In the simplest bacteria, much of the proteome is itself devoted to this task of interpretation.

Following this pattern, we seek a *Stylus* genome that encodes a special kind of text, namely, one that describes how to decode the genome. That is, the desired genome will encode a sequence of Chinese characters (in the form of vector proteins) that tells a reader of Chinese how *Stylus* genes are translated into vector sequences, and how those sequences are processed to make readable vector proteins. Although this approach joins all prior ones in falling well short of self-replication, it has the considerable advantage of stating an actual functional requirement. The stated self-descriptive function of the *Stylus* proteome provides a *real* basis for judging the adequacy of any proposed proteome, though certainly not a *simple* basis.

Before genes can be made accordingly, it is necessary to make some assumptions about structural similarities within the encoded proteome, and these will have implications for subsequent investigations. In the first place, we assume that genes encoding the same Chinese character at different locations in the genome should be true homologs, meaning that they descend from a common ancestral gene through a process of neutral evolution. Secondly, we assume that the major structural components from which the vector proteins are composed, referred to here as *domains*, should each have a fixed topology throughout the proteome. In other words, the portions of all vector proteins that form a domain of any particular type should trace the structural elements (*strokes*, discussed under "Properties and comparisons" below) in the same direction and order. This enforces a structural likeness that accords well with the likeness of structural domains in real proteins.

[4] It as actually more complex than Figure 2 implies, in that it uses 32 pixel states instead of the 2 used in Conway's *Game of Life*.

[5] The current state of the art in the field of self-replicating physical machines is exemplified by *RepRap*, an open-source project aiming to build a rapid prototyping machine that can manufacture its own parts (http://reprap.org/wiki/Main_Page). Current versions replicate only their plastic parts, making complete replication dependent both on a supply of their most complex components (e.g., motors, solenoids, and electronics) and on full assembly [12].

RESULTS

A *Stylus* glossary as a genome specification

To implement the above approach, we began by summarizing the algorithm that *Stylus* uses to translate gene-like sequences into vector proteins. This was done in the form of a concise glossary of nine key terms, which was composed in English and then translated into Chinese (Figure 3).

The resulting Chinese text is not a vector proteome in itself, but rather a *specification* for building a vector proteome by constructing a set of genes that encode it. For each line of Chinese text in Figure 3, a series of *Stylus* genes is needed to encode the characters (one gene per character). These gene series are analogous to bacterial *operons*—suites of genes encoding a set of proteins used for a coordinated task (e.g., the stepwise chemical conversions of a metabolic pathway). The process used to make these *Stylus* genes will be described later (see Methods Overview). First we discuss how the *Stylus* model needs to be extended in order for the genetic code itself to be encoded.

Genetic code specification and gene structure

By 'genetic code' we mean the reliable association of codons with amino acids that enables genes to encode proteins. Real

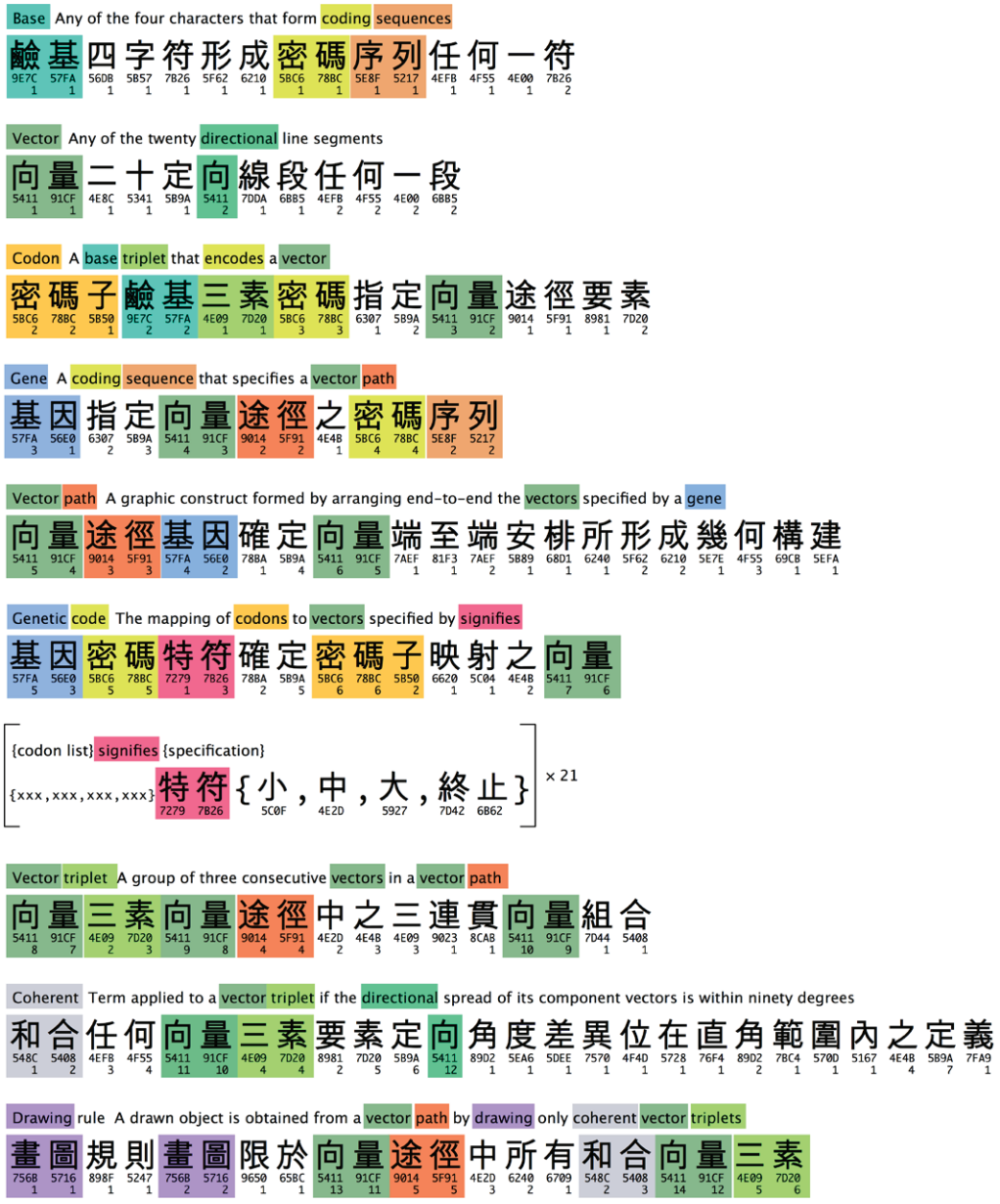

Figure 3: Specification for a compact self-descriptive *Stylus* proteome. Each of nine key terms is presented with its definition on a single line of English text. Immediately below these are the corresponding Chinese translations (written in traditional form, as explained in reference 1) with hexadecimal Unicode numbers appearing below each character. Gene identifiers have the form *uuuu.n*, where *uuuu* is the Unicode number of the encoded character and *n* (shown below the Unicode numbers) indicates the *n*th instance of that character in the genome. Colors show where some of the more heavily used characters appear and pair them with English meanings. As discussed in the text, the bracketed portion shows (in abbreviated form) how the genome and its encoded proteome specify the genetic code. **doi:**10.5048/BIO-C.2011.3.f3

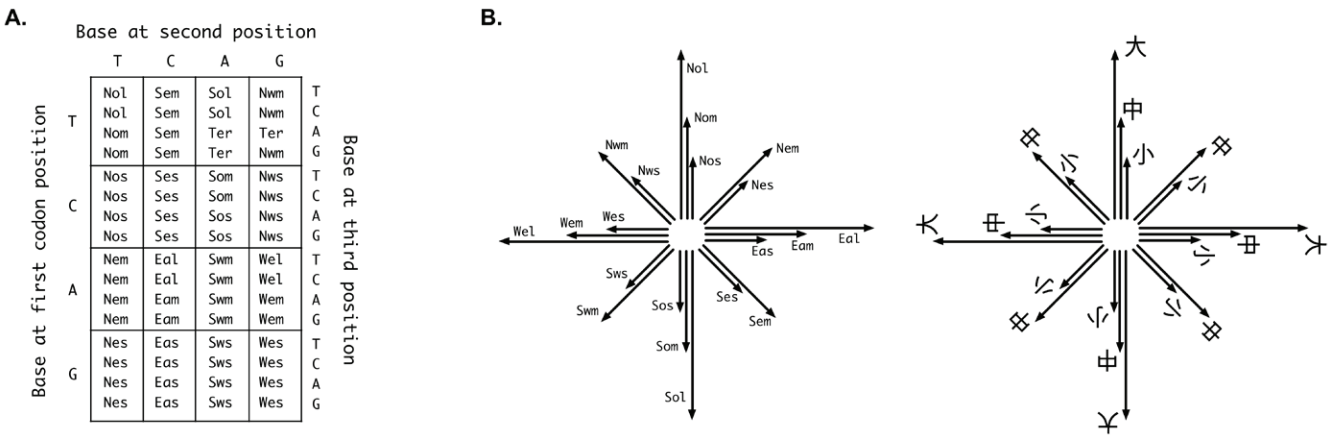

Figure 4: Genetic code implementation in *Stylus*. A) The default mapping of codons to vectors used by *Stylus*, as described previously [1]. B) The twenty vectors used to construct vector proteins, along with their standard [1] three-letter designations (left) and their character or pseudo-character designations (right). Small, medium, and large vectors are of length e^0, $e^{1/2}$, and e^1, as described [1]. Portions of this figure are reproduced from Figure 3 of reference 1. **doi:**10.5048/BIO-C.2011.3.f4

cells depend on a set of tRNAs and their corresponding aminoacyl-tRNA synthetases in order to specify the genetic code. Implementation of that code requires numerous additional components to make the amino acids and to incorporate them into protein chains. In the realm of language, the nine glossary entries of Figure 3 are analogous to the functions that implement a genetic code in that they describe *how* the code is implemented in the *Stylus* world. It is desirable to extend this analogy by devising a way for the code itself to be specified genetically, as it is in real life.

The linguistic equivalent of a set of tRNAs with their dedicated synthetases would be a set of statements of like, "Codons TGT, TGC, and TGG specify the northwestward vector of medium length," which would establish the desired mapping (see Figure 4). We devised a compact way of representing this kind of statement by using the Chinese characters for small (小, U+5C0F), medium (中, U+4E2D), and large (大, U+5927) to represent the three possible vector lengths. When they are written in their normal orientations, we take these three characters to denote northward vectors of the indicated lengths. Then, to represent the remaining seventeen vectors we form pseudo-characters by rotating 小, 中, or 大 to the appropriate angle, as shown in Figure 4B.

With these single-character representations of the twenty vectors established, the next step is to devise a compact way of specifying the codons that encode them. For this, we chose to elaborate on the concept of genes in the *Stylus* world. Figure 5A shows how the concept of open reading frames, which mirrors biology precisely [1], has now been incorporated into a full conception of genes. Conceived in this way, *Stylus* genes depend on upstream sequence in order for translation to occur, as do real bacterial genes. Analogous to the Shine-Dalgarno sequence found on bacterial mRNAs [13, 14], *Stylus* genes require[6] an AGG sequence separated from the ATG start codon by twelve bases, as depicted.

[6] We speak of requirements here only with respect to the model as currently conceived. *Stylus* software continues to operate on isolated open reading frames, without upstream sequences.

For most genes, these twelve bases are merely a spacer between the AGG and the ATG, but they serve a specific purpose for the special set of genes that specify the genetic code. Figure 5B illustrates this by depicting the operon responsible for designating the codons that specify the northeastward vector of medium length (see Nem in Figure 4A). Here, the twelve bases upstream of the first gene in the operon are interpreted as a series of four codon specifications, namely: ATG, ATC, ATT, and ATA. These combine with the functions of the three genes in the operon to provide this meaning: ATG, ATC, ATT, and ATA *signify* (特符) the Nem vector (represented by 中 rotated clockwise by 45 degrees). Vector encoding is completely specified by twenty operons of this kind that differ only in their third genes and in the twelve-base regions upstream of their first genes. One additional operon is needed to specify stop codons.

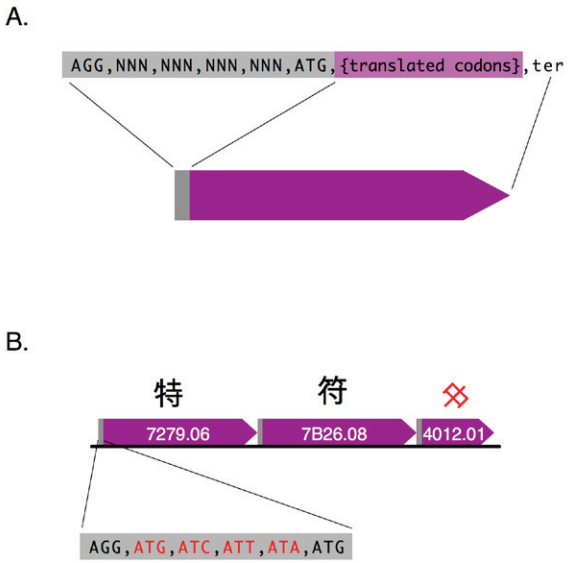

Figure 5: Using upstream sequences to specify the genetic code. A) Expressed genes in the *Stylus* world carry a fifteen-base upstream sequence, beginning with AGG. B) The structure of the operon that assigns codons to the Nem vector (see text for full description). **doi:**10.5048/BIO-C.2011.3.f5

This operon works in the same way as the others, but it has an additional gene because two characters (終止) are used to convey the meaning *terminate*. Table 1 gives the gene identifiers and the upstream sequences for all 21 code-specifying operons.

Table 1: Structure of operons specifying the genetic code

Twelve-base upstream sequence*	Gene order†	Specifies
CTA,CTT,CTC,CTG	7279.02, 7B26.04, 5C0F.01	Nos
TTG,TTA,TTA,TTG	7279.03, 7B26.05, 4E2D.01	Nom
TTT,TTT,TTC,TTC	7279.04, 7B26.06, 5927.01	Nol
GTA,GTG,GTC,GTT	7279.05, 7B26.07, 4011.01	Nes
ATG,ATC,ATT,ATA	7279.06, 7B26.08, 4012.01	Nem
GCT,GCG,GCC,GCA	7279.07, 7B26.09, 4021.01	Eas
ACG,ACA,ACG,ACA	7279.08, 7B26.10, 4022.01	Eam
ACT,ACC,ACC,ACC	7279.09, 7B26.11, 4023.01	Eal
CCT,CCG,CCC,CCA	7279.10, 7B26.12, 4031.01	Ses
TCG,TCC,TCT,TCA	7279.11, 7B26.13, 4032.01	Sem
CAA,CAA,CAG,CAG	7279.12, 7B26.14, 4041.01	Sos
CAT,CAC,CAC,CAC	7279.13, 7B26.15, 4042.01	Som
TAT,TAC,TAT,TAC	7279.14, 7B26.16, 4043.01	Sol
GAT,GAA,GAC,GAG	7279.15, 7B26.17, 4051.01	Sws
AAA,AAC,AAT,AAG	7279.16, 7B26.18, 4052.01	Swm
GGT,GGG,GGA,GGC	7279.17, 7B26.19, 4061.01	Wes
AGG,AGA,AGA,AGG	7279.18, 7B26.20, 4062.01	Wem
AGC,AGT,AGT,AGT	7279.19, 7B26.21, 4063.01	Wel
CGC,CGA,CGG,CGT	7279.20, 7B26.22, 4071.01	Nws
TGG,TGT,TGC,TGT	7279.21, 7B26.23, 4072.01	Nwm
TAG,TAA,TGA,TGA	7279.22, 7B26.24, 7D42.01, 6B62.01	Ter

* Sequences shown are from the final genome.

† The complete gene order for the genome is obtained by inserting gene identifiers from this column into Figure 3 at the brackets. Gene identifiers beginning with 40 do not correspond to actual Unicode characters, but rather to characters that have been rotated as described.

Properties and comparisons

Proteome complexity. Table 2 summarizes a number of key properties of the *Stylus* genome that was constructed (as described below) to encode the Chinese text of Figure 3. To highlight the analogies with real genomes, we compare this *Stylus* genome to several of the smallest known bacterial genomes—those of the obligate endosymbiont *Candidatus Carsonella ruddii* [15], the parasite *Mycoplasma genitalium* [16], and the obligate symbiont *Nanoarchaeum equitans* [17]. At just under half the size of the *Candidatus Carsonella* genome, the *Stylus* genome is smaller than any known bacterial genome. Yet the complexity of its proteome, in terms of the number of encoded proteins and their domain composition, is comparable to that of the real organisms. As seen in Table 2, the *Stylus* proteome has 41 more proteins than does the *Candidatus Carsonella* proteome, and it averages more domains per protein than any of the listed bacterial proteomes.

The distribution of protein chain lengths in the tiny proteome of *Candidatus Carsonella* shows a marked shift toward shorter proteins relative to the other bacterial proteomes (Figure 6). The distribution for the *Stylus* proteome peaks at a similarly short chain length but then drops off more steeply than any of the bacterial distributions. The largest vector protein in the *Stylus* proteome consists of 422 monomer units, making it less than one third the length of the largest protein from any of the bacterial proteomes.

For all proteomes the histograms of Figure 7 show that most domain types are used only once or twice. At the other extreme, all proteomes except that of *Candidatus Carsonella* have at least one domain type that is used twenty or more times. Interestingly, the *Stylus* proteome has many more of these high-use domain types than do the natural proteomes—six, as compared to one each for the proteomes of the bacteria. The proportion of single-use domain types in the *Stylus* proteome is correspondingly low (Table 2), having the overall effect of a flatter domain-use histogram[7] for the *Stylus* proteome compared to the others (Figure 7).

Figure 8 shows the major linguistic components of all 87 characters represented by the *Stylus* proteome. Because these components have functional significance in Chinese writing, they provide a basis for dividing vector proteins into functionally significant domain-like regions [1]. To avoid confusion, we apply the term *domain* not to the character components of Figure 8, but only to the corresponding portions of vector proteins, as shown in Figure 9.

The compositional relationships of Figure 8 provide important design constraints for building the proteome. Briefly, each character component consists of one or more strokes, which are continuous penned lines that may include curves or sharp bends. Although conventional writing technique calls for

[7] Relatively speaking. Considering the logarithmic scale, all of these proteomes show a pronounced preference for low-use domains.

Table 2: Comparison of Genome Properties

	***Stylus* genome**	***Candidatus Carsonella ruddii*** [*]	***Mycoplasma genitalium*** [†]	***Nanoarchaeum equitans*** [‡]
Topology	Linear	Circular	Circular	Circular
Directionality	Unidirectional (ssDNA)	Bidirectional (dsDNA)	Bidirectional (dsDNA)	Bidirectional (dsDNA)
Size	70,701 b	159,662 bp	580,076 bp	490,885 bp
% coding	95%	93%	90%	91%
Proteins encoded	223	182	475	540
Avg protein length	99 v	274 aa	369 aa	280 aa
Domain types used	112	136 (172) [§]	278 (463) [§]	228 (407) [§]
Uses per type	4.3	1.5	2.0	2.0
Single-use types [‖]	50 (45%)	106 (78%)	195 (70%)	144 (63%)
Domains per protein	2.1	1.4	1.9	1.5
Avg domain length	55 v	195 aa	194 aa	182 aa

[*] See http://www.ncbi.nlm.nih.gov/genome?term=NC_008512, and http://supfam.cs.bris.ac.uk/SUPERFAMILY/cgi-bin/gen_list.cgi?genome=5s. Data in the bottom five rows are statistical, based upon incomplete assignment of the genome (79% amino acid coverage, according to *Superfamily* database).

[†] See http://www.ncbi.nlm.nih.gov/sites/entrez?db=genome&cmd=search&term=NC_000908, and http://supfam.cs.bris.ac.uk/SUPERFAMILY/cgi-bin/gen_list.cgi?genome=mg. Data in the bottom five rows are statistical, based upon incomplete assignment of the genome (60% amino acid coverage, according to *Superfamily* database).

[‡] See http://www.ncbi.nlm.nih.gov/genome?term=NC_005213, and http://supfam.cs.bris.ac.uk/SUPERFAMILY/cgi-bin/gen_list.cgi?genome=na. Data in the bottom five rows are statistical, based upon incomplete assignment of the genome (56% amino acid coverage, according to *Superfamily* database).

[§] The number of domain types assigned by the *Superfamily* database (see links above) is given first, with a projection of the actual total number in parentheses (based on the proportion of coding sequence that cannot yet be assigned).

[‖] Percentages give the proportion of domain types that are used only once in the respective proteomes. For the bacterial proteomes, these numbers were determined by analyzing the domain-assignment text files provided by the *Superfamily* database (see links above).

strokes to be drawn in a particular order and direction, *Stylus* evaluates the functional proficiency of an encoded character (a vector protein) by examining only the shapes and locations of the strokes. This allows a vector chain to form a particular character component in multiple ways (two directions for each stroke, with $n!$ ways to order n strokes). Consequently, to specify a vector domain type, one must specify both the linguistic component and the order and direction in which its strokes are traced (referred to as the *threading scheme*). In order for all instances of a particular component throughout the proteome to share substantial structural similarity (see Approach), we have used a single threading scheme for each, resulting in 112 domain types (Table 2).

Proficiency and Fitness. As originally described [1], fitness in the *Stylus* world is akin to efficiency, which is calculated by dividing the overall functional proficiency of a genome by its resource cost. This overall proficiency clearly should be a function of the proficiency scores of the individual genes, as calculated by *Stylus*, but the question of what that function should look like is itself an interesting research topic. Given that the overall function of any *Stylus*-world genome should be linguistic meaning at the level of a text, it follows that genomes should divide naturally into operon-like groups of genes that encode sentence-level meanings [1]. One way of calculating the fitness of a whole genome sequence that has been proposed is to classify operons (sentences) as either essential or nonessential [18], but alternative approaches may also be useful.

To maximize the versatility of the genome described here, we applied a generic fitness function in which all genes are treated as equal contributors. This was done by requiring all genes to meet the same gene-level proficiency standard. Rather than setting the standard in an arbitrary way, we used the proficiency distribution for optimized genes as a guideline for setting it (see Methods Overview). The result is a minimum gene proficiency of 0.51 on a scale ranging from 0 to 1 (see Figure 10). Since *Stylus* reliably handles proficiency scores that are many orders of magnitude lower than this, the 0.51 standard falls at the high end of the useful range in logarithmic terms. It consequently proved easier to meet that standard with some characters than with others. One reason for this is that characters with curved strokes are represented only approximately by the linear monomers (vectors) that *Stylus* uses (Figure 4B). Their representation can always be improved by using more vectors (effectively shrinking the scale of the vectors relative to the scale of the character), but since this also increases the gene cost (measured by the number of bases and vectors used) an optimum gene size is reached, beyond which the cost of further gene expansion outweighs the benefit.

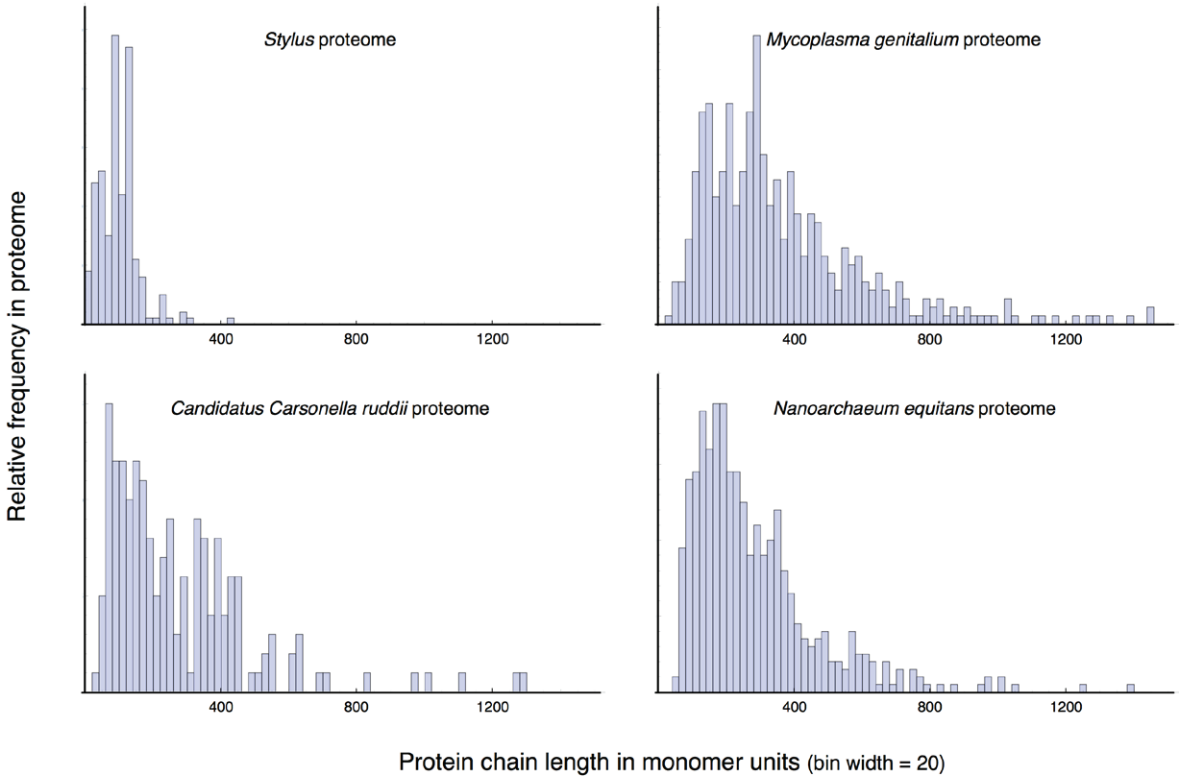

Figure 6: Distribution of protein chain lengths in compared proteomes. Chain length is measured in monomer units: amino acid residues for natural proteins and vectors for *Stylus* proteins. Follow NCBI web links in Table 2 legend to retrieve proteome tables for the bacterial species.
doi:10.5048/BIO-C.2011.3.f6

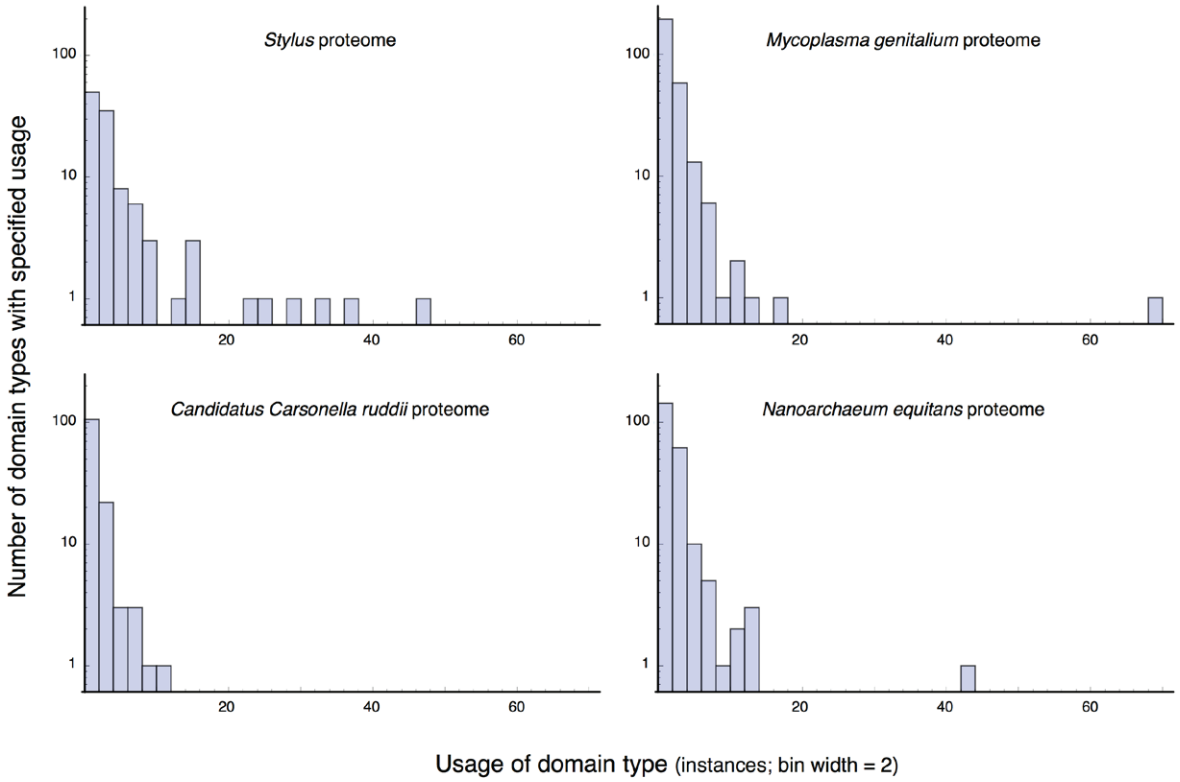

Figure 7: Distribution of domain usage in compared proteomes. Follow the *Superfamily* web links in the Table 2 legend to retrieve domain assignments for each bacterial species.
doi:10.5048/BIO-C.2011.3.f7

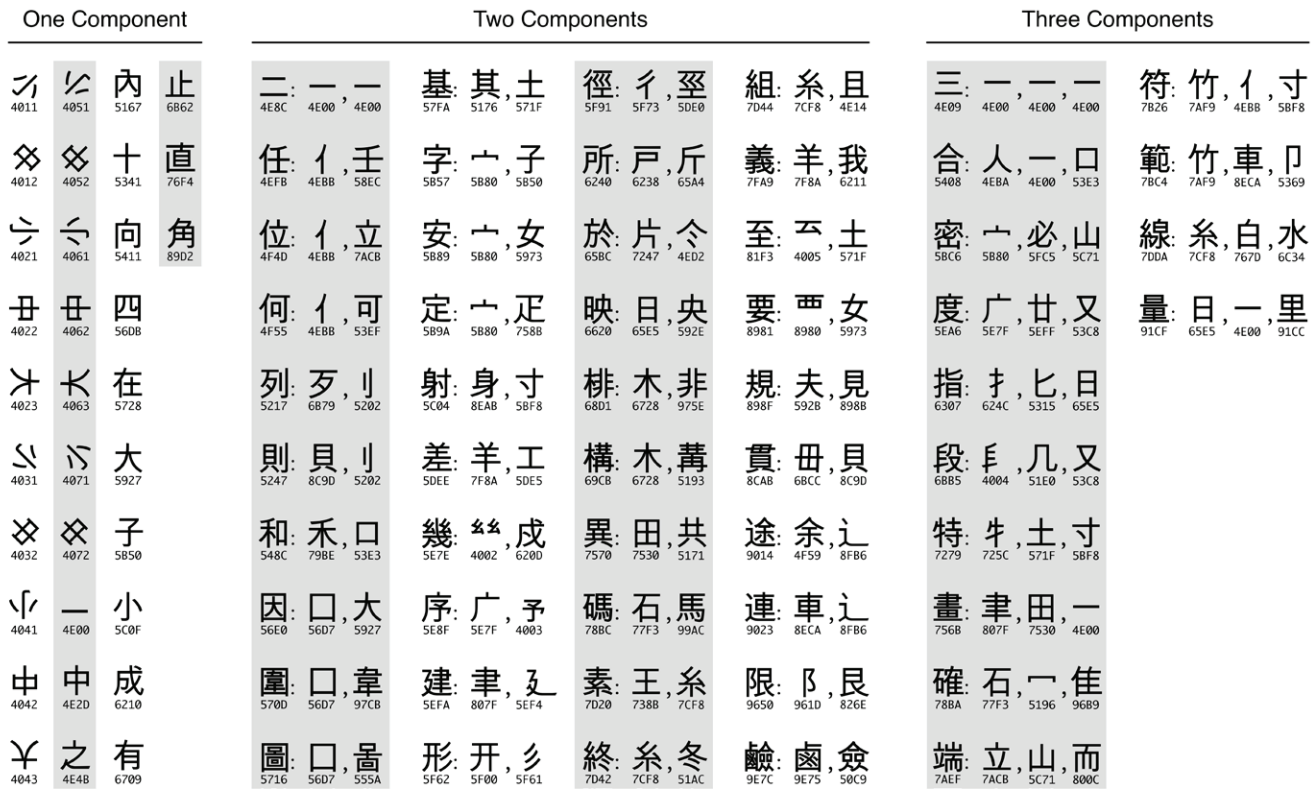

Figure 8: Composition of characters represented in the *Stylus* proteome. All 87 characters represented in the *Stylus* proteome are shown, each under the heading that specifies the number of components used to construct it. Characters built from multiple components (two or three) have a colon to their right, followed by a comma-separated list of their components. Under each heading, characters are arranged in columns according to Unicode order, with shading providing visual separation of columns. Compositions were assigned by a native reader of Chinese (P. Lu). Because most components exist as stand-alone Chinese characters, they are identified with their own Unicode identifiers. Artificial identifiers, all beginning with 40, were given to components without existing identifiers. Pseudo-characters were likewise given artificial identifiers. **doi:**10.5048/BIO-C.2011.3.f8

DISCUSSION

In this paper we have actualized some of what was earlier sketched out in concept [1] by constructing a compact *Stylus* genome that encodes a vector proteome with a real function. Like the gene-expression machinery of bacteria, the *Stylus* proteome specifies how the genomic text is interpreted as instructions for building its encoded proteins (*vector* proteins in the case of *Stylus*), but instead of giving this specification in the form of physical interactions and chemical processes that actually implement it (as in life), the *Stylus* proteome gives it in the form of a written description.

Because these are substantially different meanings of 'specification,' we will return now to the question we started with: *Of what use are artificial-world models to people interested in understanding the real world?* The short answer is that different models may be good for different things. One important topic we have touched on that *Stylus* does not address is self-replication. Other models, like von Neumann's universal constructor or Avida, provide explicit artificial versions of replication, whereas in *Stylus* the ability to replicate is part of the assumed context. What *Stylus* offers that no other model offers, to our knowledge, is an artificial version of gene-to-protein genetic causation that parallels the real thing. This brings a new level of reality to model investigations of one of the most important and con-

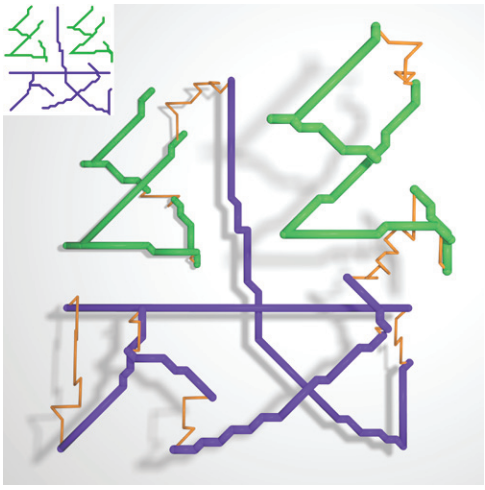

Figure 9: Domain composition of vector protein 5E7E.01. The character compositions listed in Figure 8 determine how vector proteins divide into structural domains. The two domains of 5E7E.01 are differentiated by stroke color in a 3D rendering and an inset 2D rendering. Moves between strokes are shown only in the 3D rendering, with vertical lift added in order to make the path of the vector chain easier to see. Vector domains are often formed by an unbroken stretch of chain, as seen for the 戍 (U+620D) domain depicted here (purple). Exceptions exist, though, both in natural proteins and in vector proteins, as seen here by the purple domain interrupting the green domain. **doi:**10.5048/BIO-C.2011.3.f9

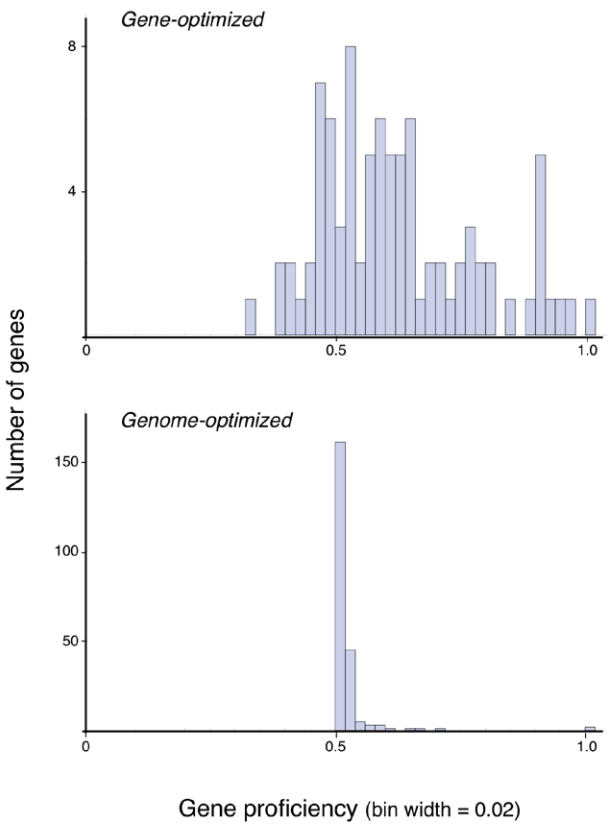

Figure 10: Distribution of gene proficiencies before and after genome optimization. The upper distribution reveals the range of natural proficiencies of the vector proteins, some of them attaining high scores more readily than others. Genome construction took account of this by targeting the genes at the low-scoring end of the distribution for proficiency enhancement. All genes were then allowed to 'relax' to the genomic optimum of 0.51 (see text), which produced the narrow distribution in the bottom graph. **doi:**10.5048/BIO-C.2011.3.f10

troversial subjects in the life sciences—the origin of biological innovations.

Figure 11 underscores the significance of this by showing the asymmetric relationship between evolutionary causation and genetic causation at the molecular level. Whatever the origin of these low-level processes of genetic causation may have been, their physical operation *today* depends only on the molecular systems now implementing them. This of course makes the study of molecular biology *as we now see it* entirely legitimate and feasible as a discipline in itself, wholly uncoupled from questions of origins. But the reverse is not at all true. Evolutionary causation is intrinsically tied to the relationship between genotype and phenotype, which depends on low-level genetic causation. It follows that evolutionary explanations of the origin of functional protein systems must subordinate themselves to our understanding of how those systems operate. In other words, the study of evolutionary causation cannot enjoy the disciplinary autonomy that studies of genetic causation can.

In view of this, the contribution of *Stylus* is to make evolutionary experimentation possible in a model world where low-level genetic causation has the essential role that it has in the real world. Combined with the free *Stylus* software, the complete *Stylus* genome made freely available with this paper[8] paves the way for analogy-based studies on a wide variety of important subjects, many of which are difficult to study by direct experimentation. Among these are the evolution of new protein folds by combining existing parts, the optimality and evolutionary optimization of the genetic code, the significance of selective thresholds for the origin and optimization of protein functions, and the reliability of methods used for homology detection and phylogenetic-tree construction.

METHODS OVERVIEW

Figure 12 summarizes the process used to construct the pair of genes needed to encode proteome character 成 (U+6210).[9] With slight variations (as noted below) the same process was used for all genes. Briefly, steps 1 through 4 of Figure 12 were used to generate prototype genes for each of the 87 proteome characters. Then, in the fifth step, a single prototype was chosen for each character in a way that maximizes the fitness of the complete genome. Finally, the "homologs" of each prototype that are needed to represent the complete proteomic text (Figure 3) were generated from these prototypes. Each of these steps are described in more detail next.

The process of gene construction

Given a character *archetype* (i.e., an ideal geometry [1]), gene construction begins by specifying both the order in which the encoded vector chain forms the strokes, and the end-to-end direction in which each stroke is formed. This joint specification—the threading scheme—is illustrated for 成 (U+6210)

[8] See Table 3 for descriptions of supplementary files and links.

[9] The positions of this character in the proteome (see Figure 3) are: character 7 of line 1, and character 18 of line 5.

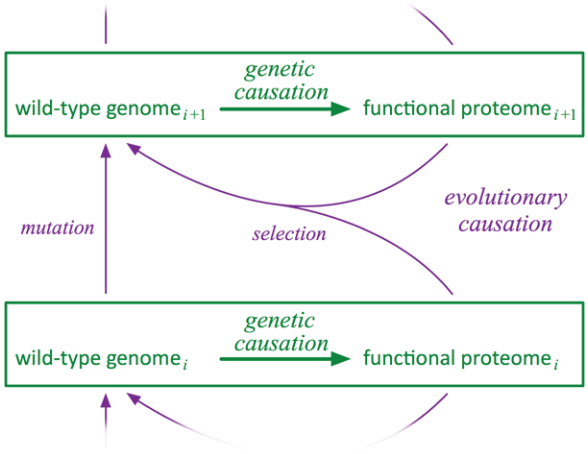

Figure 11: The relationship between low-level genetic causation and evolutionary causation. Each box in a long vertical succession (abbreviated) represents the state of a bacterial species at a particular time in its evolutionary history. The process of genetic causation (i.e., the complete process by which genes give rise to traits, like metabolic capabilities) is always a present reality that operates *as is* and may therefore be properly studied and explained *as is*, irrespective of the vertical succession. The vertical succession, on the other hand, cannot be explained properly without due consideration of genetic causation.
doi:10.5048/BIO-C.2011.3.f11

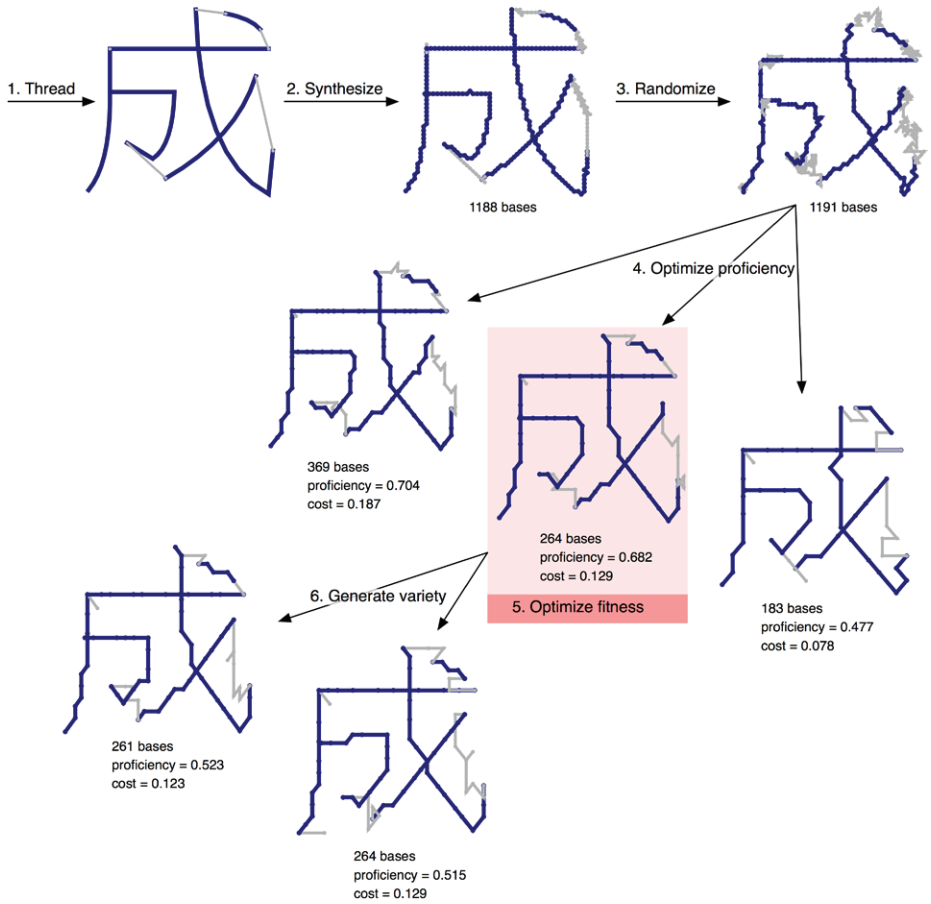

Figure 12: Steps used to produce the pair of genes encoding 成 (U+6210). The vector proteins resulting from steps 2 through 6 are shown with blue strokes and grey moves. The product of the first step is simply a scheme for tracing the strokes, depicted by connecting the stroke termini of the archetypal form with lines representing moves. In this first graphic only, grey dots mark the beginning of each move. The threading scheme therefore starts at the extreme lower left and proceeds through six strokes (the first two have abutting ends). The fifth step, selecting the gene prototype that maximizes genomic fitness, is illustrated by highlighting one of the products of step 4. Costs are based on the number of bases and the total vector length as described in Supplement 3 [18]. **doi:**10.5048/BIO-C.2011.3.f12

in the first step of Figure 12. The algorithm used to generate threading schemes attempts to minimize the total length of stroke connections (grey lines in step 1). That algorithm was applied to character components, as shown in Figure 8, with the resulting component threading schemes combined (by visual inspection) in order to build threading schemes for multi-component characters. In cases where none of the components of a multi-component character appear in any other proteome character, the algorithm was applied directly to the whole character.

The second step, synthesis of a *Stylus* gene to serve as a starting point for further processing with *Stylus*, is automated with *Inscribe*, the same Flash application used for manual construction of archetypes [1]. Using the archetype and the threading scheme as inputs, *Inscribe* produces a gene encoding a vector chain that traces the archetype strokes. As seen in Figure 12, the resulting vector protein is both large (in terms of the number of vectors used) and highly regular in structure. Further processing is needed to transform this into a structure that uses vectors efficiently and shows the variability that is typical of biological structures.

A simple randomization step (step 3) removes the regularity, with a subsequent optimization step paring the size down in order to maximize efficiency. Optimization was achieved by applying both single and double mutations (including three-base insertions and deletions) within a window that scans the length of the gene, selecting either for increased proficiency or for an increased ratio of proficiency to cost (in *Stylus* terminology, this ratio is referred to as the 'fitness' of a gene). As with real biological sequences, the vast space of sequence possibilities in the *Stylus* world and the complexity of their mapping to function preclude global optimization, so the product of any optimization process is only *locally* optimal.

Incorporating genes into a complete genome

Figure 10 (top) shows the distribution of gene proficiencies obtained when each prototype gene is optimized to maximize the ratio of proficiency to cost. The mean and mode of this distribution are both seen to be above 0.5, suggesting that a genome with a uniform proficiency standard above 0.5 may be achievable. With this aim, genes whose optimized prototypes had proficiencies below 0.54 were re-optimized to maximize

proficiency instead of proficiency ÷ cost. This has the effect of allowing genes at the low end of the proficiency distribution to be larger in order to improve their proficiencies. Multiple prototypes were produced in this way for low-proficiency genes, as shown for 成 in Figure 12 (step 4).

Taking all available prototypes into consideration, along with their associated proficiencies and costs, one can calculate the total cost of a genome that meets a specified proficiency standard by choosing for each proteome character the least costly gene that meets the standard. As the standard is increased, more costly prototypes have to be chosen for an increasing number of genes, eventually making the cost of further increase outweigh the benefit. The simple genomic fitness measure we used to perform this analysis is the ratio of the proficiency standard to the total cost of meeting that standard. Plotting this measure against the proficiency standard shows that the most fit genome that can be constructed from the available gene prototypes meets a proficiency standard of 0.51 (Figure 13).

Using this standard, we produced as many 'homologous' genes as needed for each character, starting with the chosen prototype for that character (step 6 of Figure 12). In the process, all genes were allowed to 'relax' to the proficiency standard of 0.51 by generating long mutation histories with selection applied simply to meet this standard (irrespective of the extent to which it was exceeded). As expected, the resulting set of genes shows a proficiency distribution with a sharp peak just above 0.51 (Figure 10, bottom).

Finally, since the product of the above steps is a set of 223 *Stylus* gene files (each carrying an open reading frame and information on the vector protein it encodes), additional steps were needed to convert this set into a genome. First, the gene structure shown in Figure 5A calls for fifteen bases, beginning with AGG, upstream of each open reading frame. For 21 genes (all encoding 特: U+7279) these upstream sequences have the critical function of specifying the genetic code. For the rest of the genes, the upstream sequences are arbitrary apart from the initial AGG. Arbitrary upstream sequences were generated stochastically (using Mathematica), as were the order of codon specifications in the critical upstream sequences (first column of Table 1). Next, because the processing performed with *Stylus* leaves the stop codons at the end of each gene untouched, these were stochastically reassigned from the three possibilities: TAA, TAG, TGA. After these changes were made, a full genome sequence was assembled and tested for unintended occurrences

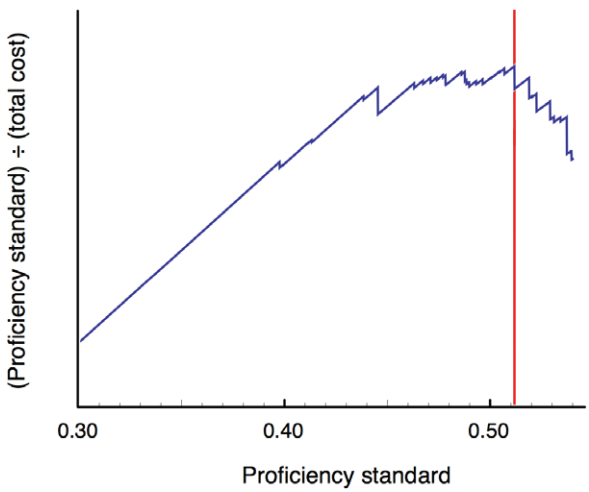

Figure 13: Dependence of genome fitness on the gene proficiency standard. In the simple case where the genomic fitness is equal to the proficiency of the least proficient gene divided by the total cost (genome + proteome; [18]), the vertical scale in this plot represents fitness (assuming the least proficient gene barely meets the required standard). The red line marks the proficiency standard that maximizes fitness.
doi:10.5048/BIO-C.2011.3.f13

(in all three reading frames) of the gene-initiation pattern: AGG, NNN, NNN, NNN, NNN, ATG. Twelve unintended occurrences were found, each of which was disrupted by altering the AGG pattern with a silent mutation. A final check confirmed that the corrected genome has precisely 223 occurrences of the gene-initiation pattern, each at its expected location.

Gene distribution and identification

In keeping with the open-source/open-access philosophy that has characterized the *Stylus* project, all gene and archetype files are made freely available with this paper (see Table 3) under the terms of the Creative Commons Attribution License described on page 1. The uniqueness of *digital object identifiers* (DOIs) allows sets of *Stylus*-world objects to be referred to unambiguously in future works. Additionally, *Stylus* automatically assigns *universally unique identifiers* (UUIDs) to all genes it produces, as does *Inscribe* for both genes and archetypes. These unique identification systems facilitate open sharing of *Stylus* objects on a large scale without compromising the ability of researchers to refer unambiguously to specific instances.

Table 3: Supplementary Files*

Description	File name	Size	doi link
Gosper glider gun animation†	glider_gun.cdf	66 K	doi:10.5048/BIO-C.2011.3.s1
Rendell Turing machine animation†	Rendell_Turing.cdf	19.4 M	doi:10.5048/BIO-C.2011.3.s2
Reference 1 supplement (for genomic fitness model)	Stylus-scoring-algorithm.pdf	1.6 M	doi:10.5048/BIO-C.2011.3.s3
Stylus gene files (XML) for all 223 ORFs in ZIP archive	Stylus-gene-files.zip	885 K	doi:10.5048/BIO-C.2011.3.s4
Stylus proteome archetype files (XML) in ZIP archive‡	Stylus-proteome-archetypes.zip	160 K	doi:10.5048/BIO-C.2011.3.s5
Complete *Stylus* genome sequence as FASTA file	full-Genome.fasta	74 K	doi:10.5048/BIO-C.2011.3.s6
FASTA files for all 223 full-length genes in ZIP archive	full-gene-FASTA-files.zip	94 K	doi:10.5048/BIO-C.2011.3.s7
FASTA files for all 223 ORFs in ZIP archive	orf-FASTA-files.zip	86 K	doi:10.5048/BIO-C.2011.3.s8
Ordered list of gene identifiers, grouped by operon§	gene-identifiers-by-operon.txt	8 K	doi:10.5048/BIO-C.2011.3.s9
Composition of proteome characters‖	proteome-character-composition.txt	4 K	doi:10.5048/BIO-C.2011.3.s10

† Requires the free Wolfram CDF player (http://www.wolfram.com/cdf-player/).

‡ One archetype file is provided for each of the 87 proteome characters. See reference 1 for information on archetypes.

§ Operon groupings correspond to lines of text in Figure 3.

‖ The file has one line for each of the 87 proteome characters, each line giving the Unicode identifier for the character followed by a list of one or more identifiers specifying its components, as in Figure 8.

1. Axe DD, Dixon BW, Lu P (2008) *Stylus*: A system for evolutionary experimentation based on a protein/proteome model with non-arbitrary functional constraints. PLoS ONE 3(6):e2246. doi:10.1371/journal.pone.0002246
2. Hirst JD (1999) The evolutionary landscape of functional model proteins. Protein Eng 9:721-726. doi:10.1093/protein/12.9.721
3. Weisstein EW "Game of Life." From *MathWorld*—A Wolfram web resource. http://mathworld.wolfram.com/GameofLife.html
4. Supplement to this paper (S1; doi:10.5048/BIO-C.2011.3.s1). Requires the free Wolfram CDF player (http://www.wolfram.com/cdf-player/).
5. Supplement to this paper (S2; doi:10.5048/BIO-C.2011.3.s2). File size: 19.4 Mb. Requires the free Wolfram CDF player (http://www.wolfram.com/cdf-player/).
6. Blackburne BP, Hirst JD (2005) Population dynamic simulations of functional model proteins. J Chem Phys 123:154907. doi:10.1063/1.2056545
7. Zeldovich KB, Chen P, Shakhnovich BE, Shakhnovich EI (2007) A first-principles model of early evolution: Emergence of gene families, species, and preferred protein folds. PLoS Comput Biol 3:e139. doi:10.1371/journal.pcbi.0030139
8. Mitchell M, Crutchfield JP, Hraber PT (1994) Evolving cellular automata to perform computations: Mechanisms and impediments. Physica D 75:361-391. doi:10.1016/0167-2789(94)90293-3
9. Lenski RE, Ofria C, Pennock RT, Adami C (2003) The evolutionary origin of complex features. Nature 423:139-144. doi:10.1038/nature01568
10. von Neumann J (1966) Theory of Self-Reproducing Automata. University of Illinois Press, Urbana.
11. Pesavento U (1995) An implementation of von Neumann's self-reproducing machine. Artif Life 2:337-354. doi:10.1162/artl.1995.2.4.337
12. Sells EA (2009) Towards a self-manufacturing rapid prototyping machine. PhD Thesis. University of Bath, UK. http://opus.bath.ac.uk/20452/
13. Shine J, Dalgarno L (1975) Determinant of cistron specificity in bacterial ribosomes. Nature 254:34-38. doi:10.1038/254034a0
14. Ma J, Campbell A, Karlin S (2002) Correlations between Shine-Dalgarno sequences and gene features such as predicted expression levels and operon structures. J Bacteriol 184:5733-5745. doi:10.1128/JB.184.20.5733-5745.2002
15. Nakabachi A, Yamashita A, Toh H, Ishikawa H, Dunbar HE, et al. (2006) The 160-kilobase genome of the bacterial endosymbiont *Carsonella*. Science 314:267. doi:10.1126/science.1134196
16. Fraser CM, Gocayne JD, White O, Adams MD, Clayton RA, et al. (1995) The minimal gene complement of *Mycoplasma genitalium*. Science 270:397-403. doi:10.1126/science.270.5235.397
17. Waters E, Hohn MJ, Ahel I, Graham DE, Adams MD, et al. (2003) The genome of *Nanoarchaeum equitans*: Insights into early archaeal evolution and derived parasitism. Proc Natl Acad Sci USA 100:12984-12988. doi:10.1073/pnas.1735403100
18. Supplement to this paper (S3; doi:10.5048/BIO-C.2011.3.s3). First published as a supplement to the original *Stylus* paper (see reference 1 above).