Genomic Grammars:
Teaching Bioinformatics Using Language Theory

Kathleen M. Kaplan, D.Sc., Lt Col John J. Kaplan (Ph.D., J.D.) USAF

Howard University/USAF

Abstract

Language theory is an important part of engineering. It is usually taught in compiler theory, operating systems, and other courses. Therefore, the use of language theory, which is familiar to most engineering students, is a good tool to use to introduce bioinformatics.

There are three elements to a language: alphabet, grammar, and semantics. The alphabet comprises the words, the grammar defines the rules, and the semantics give the meaning of the language. With respect to genomics, the genomic alphabet has been known for a long time: in DNA it's the set {A, C, T, G} and in RNA the set {A, C, U, G}. The words in genomics are the genes, of which many have been identified. The semantics, or meaning, of the genes may not be known though. With the rush to patent genes during the near-past, shot-gun methods and others were used. Thus, genes were identified, without understanding the genes' specific functions. This is akin in language theory as identifying the words (genes) of the language but not their meaning (specific functions).

This paper describes grammars, not the semantics, of language theory. It also discusses the representation of genes using grammars. Furthermore, it gives examples of project assignments for engineering students.

By teaching bioinformatics using familiar tools, such as language theory, the engineering student gains knowledge of biology, bioinformatics and the relationship of engineering principles to other disciplines.

"Proceedings of the 2005 American Society for Engineering Education Annual Conference & Exposition

Language Theory 101

A language is defined by its alphabet, grammar, and semantics. Loosely, these three elements can be described as its words, rules, and meaning, respectively.

Alphabet

The alphabet is a finite set of tokens that compose words and sentences. The symbol, Σ, represents an alphabet, which is a set of objects. If the alphabet set is the empty set it is designated as $\Sigma = \varnothing$.

STRINGS

A string in the language is a sequence of elements of length one or more and is designated by Σ^+. For example, given an alphabet:

$$\Sigma = \{a, c, t, g\}$$

then the set of strings of length one or more is:

$$\Sigma^+ = \{a, c, t, g, aa, ac, at, ag, ca, cc, ct, cg, ta, tc, \ldots\}.$$

The length of a string is the number of elements in it and is designated by parallel lines surrounding the string. For example,

$$|aga| = 3.$$

The empty string, not to be confused with the empty set, is written as ε, and the length of the empty string is zero:

$$|\varepsilon| = 0.$$

Strings may be concatenated, an operation which has no special symbol. In concatenation, strings are written next to each other. Given the alphabet above, let a string x and y be defined as:

$$x = act$$
$$y = gca$$

Then, some examples of concatenation and length are:

$$xy = actgca$$
$$yx = gcaact$$
$$|xy| = |yx| = |x| + |y| = 6$$

$$x\varepsilon = \varepsilon x = x$$
$$ac\varepsilon tgc\varepsilon a = actgca = x.$$

The set Σ^* is the set Σ^+ and the empty string, ε. Note that the alphabet is a set, not a string, and requires set operations. Combining Σ^+ and the empty string, ε, requires the union operator, $\cup$:

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}.$$

Note that:

$$\{\varepsilon\} \neq \varnothing.$$

This is because a set containing the empty string, i.e., $\{\varepsilon\}$, is not empty; the set contains one element. The empty set, $\varnothing$, on the other hand, contains no members.

LANGUAGE

A language then is a some subset of Σ^*. This is too general though to be of any use. A language must have some type of structure for legal strings; this is called a grammar.

For a grammar to include structure, terminals and nonterminals must be defined. A terminal is any member of the alphabet, Σ. A nonterminal, on the other hand, is not an element of the alphabet, Σ, but is instead a set of strings in Σ^*; the set of nonterminals is defined at a capital N.

Grammar

The grammar is a set of structural rules that define the legal contexts of tokens in the language's sentences. A grammar type is defined by its production rules, or productions. In general, a production is of the form:

$$x \rightarrow y$$

where x and y are strings of tokens. There are four classes of grammars: Type 0, unrestricted; Type 1, context-sensitive; Type 2, context-free; and Type 3, right linear, left linear, or regular. The types are defined by restrictions placed upon the productions; note that Type 0 does not follow any conceivable set of rules, including production rules.

A Type 1, Type 2, and Type 3 grammar is defined as:

$$G = (N, \sum, P, S)$$

where N is the set of nonterminals, $\sum$ is the alphabet, P is the set of productions, and S is the designated start string, S is an element of N. The types differ in their production rules, which are given in Table 1.

Table 1. Grammar Types and Production Forms and Restrictions

Grammar	Production Form	Restrictions
Type 0, Unrestricted	*No Restrictions*	
Type 1, Context Sensitive	$x \rightarrow y$	x and y are members of $(N \cup \Sigma)^*$, x contains at least one member of N, and the length of x is less than or equal to the length of y, $\|x\| \leq \|y\|$. Note $y \neq \varepsilon$.
Type 2, Context Free	$x \rightarrow y$	x is a member of N, y is any string in $(N \cup \Sigma)^*$. Note $y = \varepsilon$ is allowed.
Type 3, Right Linear, Left Linear, Regular	Right Linear: $A \rightarrow xB$ or $A \rightarrow x$ - or - Left Linear: $A \rightarrow Bx$ or $A \rightarrow x$ - or - Regular: $A \rightarrow aB$ or $A \rightarrow a$ or $S \rightarrow \varepsilon$	Right and Left Linear: A and B are in N, x is in Σ^*. Regular: A is in N, a is in Σ, S is the start symbol. If $S \rightarrow \varepsilon$ is in the grammar, then S does not appear on the right side of any other production.

In computing, the three higher grammar types are associated with automaton as seen in Table 2.

Table 2. Structured Grammars and Automaton

Grammar	**Automaton**
Context sensitive, Type 1	Turing machine (two-way, linear bounded automaton) where the tape is not allowed to be larger than the input string
Context free, Type 2	Push down stack
Right linear, Type 3	Finite state automaton

Semantics

The semantics, or meaning, is the set of rules that define the operational effect of any program written in the language when translated and executed on a system. These elements can be defined with respect to a genome. The genomic semantics is not addressed in this paper.

Genomic Language

A genome can be defined as a language. It has an alphabet, valid strings, and a specific grammar, including production rules. Therefore, an engineer can relate terms used in courses involving language theory, including compilers, algorithm analysis, and other subjects, to the genome.

Nucleotide Alphabet

A genome can be defined as a string of nucleotides, which are tokens in language theory. There are four distinct nucleotides, {a, c, g, t} in DNA and {a, c, g, u} in RNA, ribonucleic acid. (Think of DNA as a big reference book in a library. Only a few pages of this big book are needed and copied. In this example, RNA is the copied pages [6].) In this respect, the genomic alphabet can be defined as:

$$\Sigma = \{a, c, g, u\}$$

with $|\Sigma| = 4$.

NUCLEIC STRINGS

A word in the genomic language can be thought of as an amino acid, which is represented by three nucleotides; the list of amino acids is given in Table 3. Since there are four possible elements, there are 4^3, or 64, possible codons. There is redundancy in the mapping, with only twenty amino acids and three stop codons designated by nature, for a 23:64 ratio. This mapping is seen in Table 4.

Table 3. Amino Acids [7]

	One-Letter Code	Three-Letter Code	Name
1	A	Ala	Alanine
2	C	Cys	Cysteine
3	D	Asp	Aspartic Acid
4	E	Glu	Gluetamic Acid
5	F	Phe	Phenylalanine
6	G	Gly	Glycine
7	H	His	Histidine
8	E	Ile	Isoleucine
9	K	Lys	Lysine
10	L	Leu	Leucine
11	M	Met	Methionine
12	N	Asn	Asparagine
13	P	Pro	Proline
14	Q	Gln	Glutamine
15	R	Arg	Arginine
16	S	Ser	Serine
17	T	Thr	Threonine
18	V	Val	Valine
19	W	Trp	Tryptophan
20	Y	Tyr	Tyrosine

Table 4. Amino Acids and Stop Codons with Corresponding Codons [7]

First Position	Second Position				Third Position
	G	A	C	U	
G	Gly	Glu	Ala	Val	G
	Gly	Glu	Ala	Val	A
	Gly	Asp	Ala	Val	C
	Gly	Asp	Ala	Val	U
A	Arg	Lys	Thr	Met	G
	Arg	Lys	Thr	Ile	A
	Ser	Asn	Thr	Ile	C
	Ser	Asn	Thr	Ile	U
C	Arg	Gln	Pro	Leu	G
	Arg	Gln	Pro	Leu	A
	Arg	His	Pro	Leu	C
	Arg	His	Pro	Leu	U
U	Trp	STOP	Ser	Leu	G
	STOP	STOP	Ser	Leu	A
	Cys	Tyr	Ser	Phe	C
	Cys	Tyr	Ser	Phe	U

Strings in the grammar only make sense if they are of length 3. Let Σ^3 be the set of strings of length 3. Therefore, the number of elements in the set Σ^3 is 64.

Amino Acid Alphabet

On the other hand, the genomic alphabet can be defined as the amino acid alphabet. A genome can be defined as a string of amino acids, instead of nucleotides. There are 20 amino acids, represented by one letter as in Table 3; the amino acid alphabet can be defined as:

$$\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}.$$

Stop codons can be included in this alphabet, for a total of 21 elements, given that all stop codons are represented by Z:

$$\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y, Z\}.$$

AMINO ACID STRINGS

Amino acids can be combined to form proteins, which would be the words in this language representation.

Genomic Grammars

In [6], a valid sentence in the genomic language was given:

AUG{{{A,C,G}{A,C,G,U}+U{C,U}}{A,C,G,U}+U{G{C,G,U}+A{C,U}}}*
U{GA+A{G,A}}

The operators of the language are:

* means zero or more times,
+ means "or," and
{} means choose one of the set.

So, some valid strings in this language are:

AUGUGA
AUGAUAUAG
AUGUCUCGCUGA

Note that the property of concatenation, discussed above, can be applied to the genomic language. A valid string starts with "AUG" and ends with a stop codon. Any number of codons can be concatenated to the middle of the start and stop.

To represent the genomic language, a grammar is constructed for this sentence:

$G = (N, \Sigma, P, S)$
$N = \{S, H, I, J, K, L\}$
$\Sigma = \{a, c, g, u\}$
$S = S$
P:

$S \rightarrow a\ u\ g\ H\ u\ I$
$H \rightarrow J\ K\ H \mid u\ L\ H \mid \varepsilon$
$J \rightarrow a\ K \mid c\ K \mid g\ K \mid u\ c \mid u\ u$
$K \rightarrow a \mid c \mid g \mid u$
$L \rightarrow g\ c \mid g\ g \mid g\ u \mid a\ c \mid a\ u$
$I \rightarrow g\ a \mid a\ g \mid a\ a$

Note that the productions are written using "|" for an "or." In other words, *K* can be replaced by *a*, *c*, *g*, or *u*. Because there is a production of the form $H \rightarrow \varepsilon$, this is not a Type 1 grammar. Also, since at least one production has more than two symbols on the right-hand side, it is not a Type 3 grammar. In fact, this is a Type 2, or context free grammar.

The grammar can be rewritten as a Type 1, or context sensitive grammar by changing the productions so that the right-hand sides do not contain the empty string. A context sensitive grammar for this language is:

$G = (N, \Sigma, P, S)$
$N = \{S, H, I, J, K, L\}$
$\Sigma = \{a, c, g, u\}$
$S = S$
P:

$S \rightarrow a\ u\ g\ H\ u\ I \mid a\ u\ g\ u\ I$
$H \rightarrow J\ K\ H \mid u\ L\ H \mid J\ K \mid u\ L$
$J \rightarrow a\ K \mid c\ K \mid g\ K \mid u\ c \mid u\ u$
$K \rightarrow a \mid c \mid g \mid u$
$L \rightarrow g\ c \mid g\ g \mid g\ u \mid a\ c \mid a\ u$
$I \rightarrow g\ a \mid a\ g \mid a\ a$

In [6] a finite state automata was given for the language above. Therefore, a right linear grammar could be constructed, as indicated in Table 2. This would be a good task for students!

Other Assignments

As the engineering student learns to integrate language theory and genomic concepts, he or she will be able to understand the importance of applying this engineering knowledge to different fields. An example using the nucleotide alphabet was given, but the student could easily apply language theory concepts to the amino acid alphabet, given above. If a student can create a state diagram for any genomic concept, then he or she should be able to obtain a grammar for it as well, as indicated by Table 2. Also, a type of genomic grammar could be given to the engineering students and they should be able to create an automaton, either a push-down stack for a context free grammar, a Turing machine for a context sensitive grammar, or a finite state automaton for a right linear grammar. These would be good programming assignments for undergraduates.

Conclusion

Language theory is usually taught in compiler theory, operating systems, and other courses, that are in an engineering student's curriculum. Using these familiar concepts is a good way to introduce bioinformatics without spending a great deal of time on reviewing genomic concepts at the onset of a bioinformatics course.

This paper included a brief review of language theory, including alphabets and grammars. It also included ways to represent a genomic language by two of the four types of grammars. Also discussed were other possible assignments for the engineering student taking a bioinformatics course.

The authors have been teaching Computational Biology and Bioinformatics to both undergraduate and graduate students for a few years. Through their trials they have come to the conclusion that the best way to integrate genomic constructs to engineers is to have the students work genomic problems using engineering techniques. A long introduction to the genome is not necessary at the outset of the course; the material can be integrated at every level by using appropriate engineering concepts. This way, the students relate what they have learned in their engineering courses to real world genomic problems, albeit simple ones. The students seem to enjoy these assignments and an early introduction in this manner tends to keep their attention for the duration of the course.

By introducing bioinformatics using language theory, a familiar tool of the engineering student, the student gains knowledge of the concepts by the best possible method: *doing!*

References

Note: The first four references are good references for the beginning bioinformatics student; the fifth reference contains language theory information which may be found in any algorithm, compiler, or operating system textbook.

[1] Alcamo, Lawrence M. Elson, *Microbiology Coloring Book*, Addison-Wesley, January, 1997.

[2] The National Human Genome Research Institute, http://www.genome.gov, accessed Jan. 2004.

[3] Gonick, Larry, and Mark Wheelis, *The Cartoon Guide to Genetics*, Harper & Row Publishers, 1983.

[4] Rosenfield, Israel, and Edward Ziff, *DNA for Beginners*, Writers and Readers Publishing, 1984.

[5] Barrett, William A., et. al., *Compiler Construction*, Science Research Associates, 1986. See Chapter 2, Introduction to Language Theory.

[6] Kaplan, Kathleen M., and John J. Kaplan, "The Switching Circuits of Biology," *ASEE 2004 Conference Proceedings*, June 2004.

[7] Setubal, Joao Carlos, and Joao Meidanis, *Introduction to Computational Molecular Biology*, PWS Publishing Company, 1997.

Biographical Information

KATHLEEN M. KAPLAN, D.Sc.
Dr. Kaplan is an Assistant Professor in the Department of Systems & Computer Science at Howard University. She is also a Registered Patent Agent licensed to practice before the United States Patent and Trademark Office. She can be reached at kkaplan@howard.edu.

JOHN J. KAPLAN, Ph.D., J.D.
Dr. Kaplan is a Lieutenant Colonel (Lt Col) in the United States Air Force and a Patent Attorney. Lt Col Kaplan is the Deputy Director of Policy and Integration at the Air Force Office of Scientific Research (AFOSR) in Arlington, Virginia. He can be reached at john.kaplan@afosr.af.mil.