Succinct Introduction to Formal Languages

Professor Daniel D. Warner

Fall 2005

1 Basic Definitions.

Let V be a nonempty finite set whose elements are called tokens. By a string of length k over V, we mean a sequence of k tokens from V.

$$\Sigma = v_1, v_2, ..., v_k \quad \text{where} \quad v_i \in V.$$

The unique string containing no tokens will be called the empty string and will be denoted by λ. We will use the notation $|\omega| = k$ to indicate that the string, ω, has length k. Of course, $|\lambda| = 0$. Depending on the context we will also refer to tokens as letters, in which case we will refer to V as an alphabet, and the strings over V will be referred to as words. In a similar vein we will also use the terms words, vocabulary and sentences.

Example 1. For the alphabet $V = \{$A, . . . , Z, 0, . . . , 9$\}$, the strings READ, ELSEIF, DO10I and 150E5 are words over V.

Example 2. For the vocabulary $V = \{$ `<ate>, <apple>, <boy>, <the>` $\}$ the string:
`<the><boy><ate><the><apple>` is a sentence over V. Following the usual conventions we would write this sentence as "the boy ate the apple."

The point is that tokens are indivisible atomic objects, but the level of indivisibility can depend on the context. The set of all strings over V is denoted by V^*. It is a countably infinite set. Any subset of V^* constitutes a formal language over V.

If a and b are two strings over V then their concatenation is the string ab, which is also a string over V. The notation a^k denotes the concatenation of k copies of the string a. Note that $|ab| = |a| + |b|$ and $|a^k| = k|a|$.

Example 3. Let $V = \{a, b\}$, then the following sets are formal languages over V.

$$\begin{aligned}
L_0 &= \emptyset \text{ (the empty set)} \\
L_1 &= \{\lambda\} \\
L_2 &= \{a, b, aba, abbb\} \\
L_3 &= \{a^p | \text{paprime}\} = \{aa, aaa, aaaaa, aaaaaaa, \ldots\} \\
V^* &= \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \ldots\}
\end{aligned}$$

Exercise 1. Prove that V^* with the binary operation of concatenation is a semigroup with an identity. It is known as the free semigroup over V.

2 Formal Grammars.

Given a set of tokens V, any collection of strings over V constitutes a formal language. Suppose that L is a subset of V^*. How can we determine if a string ω is an element of L? Also, can we find an algorithm for systematically generating all the elements of L? We encounter this problem daily. Let V be the vocabulary of English words, and let L be the set of all English sentences. How do we understand the meaning of sentences we've never heard before?

An important part of this understanding derives from the fact that an intelligible English sentence must be constructed in accordance with certain grammatical rules. For example, we immediately recognize that "the dog bit the boy" and "the boy bit the dog" are both grammatically correct sentences and that "the bit boy dog the" is not. In addition the grammar enables us to distinguish which animal, the boy or the dog, is the subject and which animal is the object. That is, the grammar provides the important information about which one is doing the biting and which one is being bitten. It turns out that the idea of a grammar can be formalized so that we can use a grammar as a basis for generating all the elements of a formal language.

A *formal grammar* is a four-tuple, $\mathbf{G} = (\mathbf{N}, \mathbf{T}, \mathbf{P}, \Sigma)$, where

- $\mathbf{N}$ is a finite set of nonterminal symbols,
- $\mathbf{T}$ is a finite set of terminal symbols,
- $\mathbf{N}$ and $\mathbf{T}$ are disjoint,
- $\mathbf{P}$ is a finite set of productions,
- Σ is the sentence symbol, and Σ is not an element of $N \cup T$.

Each production in $\mathbf{P}$ is an ordered pair of strings (α, β) with $\alpha = \phi A \psi$ and $\beta = \phi \omega \psi$, where ϕ, ω and ψ are possibly nonempty strings in $(N \cup T)^*$ and A is either Σ or a nonterminal symbol.

In these notes the production (α, β) will be written as $\alpha \to \beta$. When a grammar is used to define the syntax of a computer language, the productions are often written as $\alpha ::= \beta$.

An element of $(N \cup T)^*$ will be called a sentential form. To generate a sentence according to a formal grammar we start with the sentential form Σ, and then we successively rewrite the sentential form according to one of the production rules. We have a sentence of the formal language defined by the grammar when the sentential form contains only terminal symbols. The sequence of sentential forms required to generate a sentence is called a derivation of the sentence. To state this more precisely, let G be a formal grammar. A string of symbols from $(N \cup T)^* \cup \{\Sigma\}$ is known as a sentential form. If $\alpha \to \beta$ is a production of $\mathbf{G}$ and $\omega = \phi\alpha\psi$ and $\hat{\omega} = \phi\beta\psi$ are sentential forms, we say the $\hat{\omega}$ is *immediately derived* from ω in G, and we indicate this relation by writing $\omega \Rightarrow \hat{\omega}$. If ω_1, ω_2, ..., ω_n, is a sequence of sentential forms such that $\omega_1 \Rightarrow \omega_2 \Rightarrow \ldots \Rightarrow \omega_n$, we say that ω_n is derivable from ω_1 and indicate this relation by writing $\omega_1 \Rightarrow^* \omega_n$. The sequence ω_1, ω_2, ..., ω_n is called a *derivation* of ω_n from ω_1 according to $\mathbf{G}$. The language $\mathbf{L}(\mathbf{G})$ generated by a formal grammar $\mathbf{G}$ is a set of terminal strings derivable from Σ, $\mathbf{L}(\mathbf{G}) = \{\omega \in \mathbf{T}^* | \Sigma \Rightarrow^* \omega_n\}$.

If ω is in $\mathbf{L}(\mathbf{G})$, we say that ω is a word in the language generated by $\mathbf{G}$. The terms string and sentence are also used.

Example 4. Let $\mathbf{N} =$ (`<subject>`, `<predicate>`, `<noun phrase>`, `<noun>`, `<article>`, `<verb>`, `<direct object>`). Let $\mathbf{T}$ = (the, boy, dog, bit). Let the set of productions consist of:

1. $\Sigma \to$ `<subject><predicate>`
2. `<subject>` $\to$ `<noun phrase>`
3. `<predicate>` $\to$ `<verb>`

4. <predicate> → <verb><direct object>
5. <noun phrase> → <article><noun>
6. <direct object> → <noun phrase>
7. <noun> → boy
8. <noun> → dog
9. <article> → the
10. <verb> → bit

Now consider the following derivation.

Σ	
<subject><predicate>	by rule 1
<noun phrase><predicate>	by rule 2
<article><noun><predicate>	by rule 5
the<noun> <predicate>	by rule 9
the dog <predicate>	by rule 8
the dog <verb><direct object>	by rule 4
the dog bit <direct object>	by rule 10
the dog bit <noun phrase>	by rule 6
the dog bit <article><noun>	by rule 5
the dog bit the <noun>	by rule 9
the dog bit the boy	by rule 7

Example 5. Let $\mathbf{G_1}$ have $\mathbf{N} = \{A, B, C\}$, $\mathbf{T} = \{a, b, c\}$ and the set of productions

1. $\Sigma \rightarrow A$
2. $A \rightarrow aABC$
3. $A \rightarrow abC$
4. $CB \rightarrow BC$
5. $bB \rightarrow bb$
6. $bC \rightarrow bc$
7. $cC \rightarrow cc$

Production 4 does not satisfy the strict definition of a production. It is actually a shorthand notation for the following four rules:

1. $CB \rightarrow XB$
2. $XB \rightarrow XY$
3. $XY \rightarrow BY$
4. $BY \rightarrow BC$

The following derivation shows that $a^3b^3c^3$ is in $\mathbf{L}(\mathbf{G_1})$.

Σ	
A	by Rule 1
aABC	by Rule 2
aaABCBC	by Rule 2
aaabCBCBC	by Rule 3

aaabBCCBC	by Rule 4
aaabbCCBC	by Rule 5
aaabbCBCC	by Rule 4
aaabbBCCC	by Rule 4
aaabbbCCC	by Rule 5
aaabbbcCC	by Rule 6
aaabbbccC	by Rule 7
aaabbbccc	by Rule 7

Exercise 2. Show that$\mathbf{L}(\mathbf{G_1}) = \{a^k b^k c^k | k \geq 1\}$.

Example 6. The grammar $\mathbf{G_2}$ is a modification of $\mathbf{G_1}$ with the following production rules:

1. $\Sigma \rightarrow A$
2. $A \rightarrow aABC$
3. $A \rightarrow abC$
4. $CB \rightarrow BC$
5. $bB \rightarrow bb$
6. $bC \rightarrow b$

Note that Rule 6 removes the nonterminals, C, from sentential forms. Moreover, it is the only rule which does so. The following derivation shows that a^3b^3 is in$\mathbf{L}(\mathbf{G_2})$.

Σ
A by Rule 1
aABC by Rule 2
aaABCBC by Rule 2
aaabCBCBC by Rule 3
aaabBCBC by Rule 6
aaabbCBC by Rule 5
aaabbBC by Rule 6
aaabbbC by Rule 5
aaabbb by Rule 6

Example 7. The following grammar, $\mathbf{G_3}$, is a simpler, noncontracting, grammar which generates $\{a^k b^k | k \geq 1\}$.

1. $\Sigma \rightarrow A$
2. $A \rightarrow aAb$
3. $A \rightarrow ab$

A derivation of a^3b^3 is

$$\Sigma \Rightarrow A \Rightarrow aAb \Rightarrow aaAbb \Rightarrow aaabbb$$

The reader should verify that $\mathbf{L}(\mathbf{G_3})$ generates $\{a^k b^k | k \geq 1\}$.

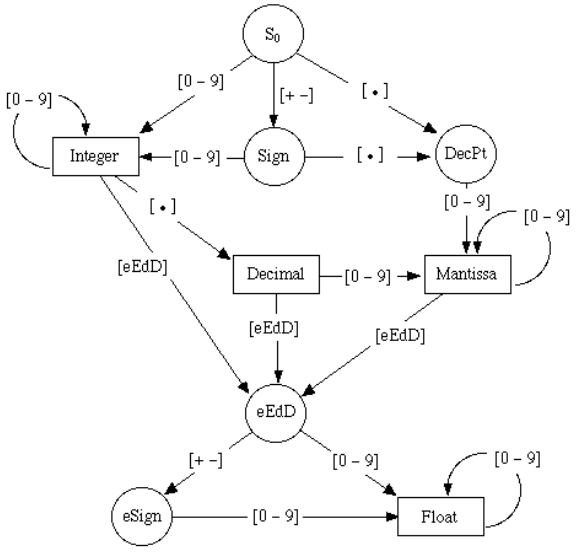
3 Types of Languages.

By placing various restrictions on the allowable production rules, we can classify the formal languages into four types.

Type 0:	$\phi A \psi \rightarrow \phi \omega \psi$	Since we can have $\omega = \lambda$, some productions can cause the derivation to contract. It can be shown that this class of productions is equivalent to the class of unrestricted productions.
Type 1:	$\phi A \psi \rightarrow \phi \omega \psi$, $\omega \neq \lambda$	Context sensitive, noncontracting
	$\Sigma \rightarrow \lambda$	languages.
Type 2:	$A \rightarrow \omega$, $\omega \neq \lambda$	Context free languages.
	$\Sigma \rightarrow \lambda$	
Type 3:	$A \rightarrow aB$	Right linear or regular languages.
	$A \rightarrow a$	(We can also define left linear)
	$\Sigma \rightarrow \lambda$	

The interesting result is that these simple restrictions on the types of production rules divide formal languages into four classes, called the Chomsky heirarchy. Moreover each of the four classes of formal languages can be recognized by a different class of formal automatons. In particular, Type 3 languages can be recognized by finite state machines; Type 2 languages can be recognized by push-down automata; Type 1 languages can be recognized by linear bounded automata; and Type 0 languages can be recognized by Turing machines.

Example 8. From the state transition diagram for the Fortran Floating Point Recognizer, we can construct the following right linear grammar, $\mathbf{G_4}$, to describe Fortran floating point literals.



In this example brackets are used to define one of a set of characters. $[+-]$ denotes either a plus or minus sign and $[0-9]$ denotes one of the digits 0 through 9. This is just a shorthand which allows us to describe the 170 productions rules in only 27 lines.

1. $\Sigma \rightarrow [0-9]A$
2. $\Sigma \rightarrow [0-9]$

3. $\Sigma \rightarrow [+-]B$
4. $\Sigma \rightarrow [.]C$
5. $A \rightarrow [0-9]A$
6. $A \rightarrow [0-9]$
7. $A \rightarrow [.]D$
8. $A \rightarrow [.]$
9. $A \rightarrow [eEdD]E$
10. $B \rightarrow [0-9]A$
11. $B \rightarrow [0-9]$
12. $B \rightarrow [.]C$
13. $C \rightarrow [0-9]F$
14. $C \rightarrow [0-9]$
15. $D \rightarrow [0-9]F$
16. $D \rightarrow [0-9]$
17. $D \rightarrow [eEdD]E$
18. $E \rightarrow [+-]G$
19. $E \rightarrow [0-9]H$
20. $E \rightarrow [0-9]$
21. $F \rightarrow [0-9]F$
22. $F \rightarrow [0-9]$
23. $F \rightarrow [eEdD]E$
24. $G \rightarrow [0-9]H$
25. $G \rightarrow [0-9]$
26. $H \rightarrow [0-9]H$
27. $H \rightarrow [0-9]$

The following derivations illustrate some of the more interesting Fortran floating point constants contained in $\mathbf{L}(\mathbf{G_4})$.

Σ	
1A	by Rule 1
1.	by Rule 8

Σ	
2A	by Rule 1
2eE	by Rule 9
2e+G	by Rule 18
2e+1H	by Rule 24
2e+12H	by Rule 26
2e+123	by Rule 27

Σ	
.C	by Rule 4
.3	by Rule 14

Σ	
-B	by Rule 3
-4A	by Rule 10
-45A	by Rule 5
-45.D	by Rule 7
-45.6F	by Rule 15
-45.6dE	by Rule 23
-45.6d-G	by Rule 18
-45.6d-7	by Rule 25

There are two interesting questions. (1) How do we convert a right linear grammar into a finite state machine which recognizes words in the language? (2) Can we create a program which reads the grammar and builds the finite state machine automatically?