# Grammars

See Section 3.2 of the text

Warning -- there is a bunch of terminology ahead.  It isn't hard and you don't have to memorize it, but if you want to understand the properties of different types of parsers and why you might one to use one type rather than another, you have to have some idea what this terminology means.

Here are 6 definitions:

1. An *alphabet* V is a finite collection of symbols, such as {0, 1}, {"a", "b", "c", ..., "z"} or {"if", "then", "else"}
2. V* is the collection of all strings made up of 0 or more elements of V.
3. $\varepsilon$ is the *empty string* -- the string of length 0
4. $V^+ = V^* - \{\varepsilon\}$
5. A *language over V* is any subset of V*
6. A *grammar* for a language is a method for satisfying which strings are in the language. This consists of
   a. An alphabet $V_T$ of *terminal symbols*
   b. An alphabet $V_N$ of *nonterminal symbols*
   c. One or more *start symbols* in $V_N$
   d. A set of *production rules* for expanding strings.

The language generated by the grammar is the set of strings in $V_T^*$ that can be generated from the start symbols by following the production rules.

Different classes of grammars (such as regular, context-free, or context-sensitive) have different sorts of grammar rules.

An example should help. Here is a simple grammar for a language of arithmetic expressions:

$V_T$={0, 1, 2, 3,...,9, + *}

$V_N$={E, T, N, D}

E is the start symbol.

Rules:

E ::= E+T

E ::= T

T ::= T*N

T ::= N

N ::= DN

N ::= D

D ::= 0|1|2|...|9 (as with regular expressions, | means "or" )

Here is a derivation that shows that 3+42*5 is a string in the language generated by this grammar:

E ::= E+T
::= T+T*N
::= N+N*N
::= D+DN*N
::= D+DD*D
::= 3+42*5

However, 3++4 is not in the language:

The only rule containing + is E ::= E+T, so both +-symbols need to come from this rule:

E ::= E+T

::= E+T+T

There is no way for that middle T to become an empty string, so no string in the language can have two consecutive +-symbols.

To save space, we often write all of the rules that have the same left side on one line, separating the right sides with |. The previous grammar would be written

E ::= E+T | T
T ::= T*N | N
N ::= DN | D
D ::= 0 | 1 | ... | 9

In general, production rules have the form

$$\alpha ::= \beta$$

where $\alpha$ and $\beta$ are both strings in $(V_T \cup V_N)^*$

A *derivation* is a sequence of steps that replaces the left side of a production rule with the right side of this rule. We usually continue derivations until we have derived a string of terminal symbols.

Here is another grammar:

$V_T$={a, b, c}

$V_N$={S, T, U}

The start symbol is S

Rules:

S ::= aSTU

S ::= abU

bT ::= bb

bU ::= bc

UT ::= TU

cU ::= cc

Here is a quick derivation:

S ::= a<u>bU</u>
::= abc

Here is another derivation:

S ::= a<u>S</u>TU
::= aab<u>UT</u>U
::= aa<u>bT</u>UU
::= aab<u>bU</u>U
::= aabb<u>cU</u>
::= aabbcc

It isn't terribly difficult to show that this grammar generates the language $\{a^n b^n c^n : n >= 1\}$

Types of Grammars:

**Regular**: All production rules are either of the form A ::= a or A ::= aB, where A and B are nonterminal symbols and a is a terminal symbol.

**Context Free**: All production rules have the form A ::= $\alpha$, where A is a single nonterminal symbol and $\alpha$ might have both terminals and nonterminals.

**Context Sensitive**: All production rules have the form $\alpha$ ::= $\beta$, where $|\alpha| <= |\beta|$

**Arbitrary**

# The Chomsky Hierarchy

| Grammar | Machine that Recognizes |
|---|---|
| Regular | DFA |
| Context Free | PDA (DFA+Stack) |
| Context Sensitive | Turing Machine with bounded memory |
| Arbitrary | Turing Machine |

For the rest of the term we will only work with context free grammars.

More terminolgy.  Be patient, we are getting through it.

A *sentential form* is any string in $(V_T \cup V_N)^*$ that can be derived from a start symbol.  In other words a sentential form is one step of a derivation.  A *sentence* is a sentential form with only terminal symbols.

A *phrase* is a substring of a sentential form that can be produced from a single nonterminal symbol.

For example, with the grammar

E ::= E+T | T
T ::= T*N | N
N ::= DN | D
D ::= 0 | 1 | ... | 9

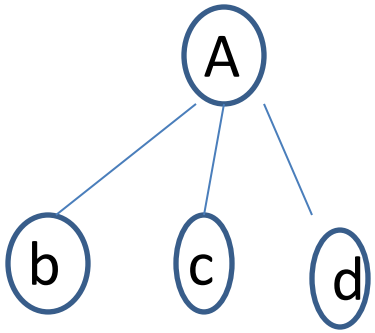
3+4 is a sentence, 3+T is a sentential form, and 4*5 is a phrase (as well as a sentence).

*Parsing* is the process of taking a string of terminal symbols and producing a derivation for it. There are 2 main styles:

LL: read the string from left to right, always expanding the left-most nonterminal symbol.

LR: read the string from left to right, always expanding the right-most nonterminal symbol.

We usually display a derivation as a *parse tree*, where a rule such as  A ::= bcd is displayed as



Top-down parsing builds the parse tree from the top (start symbol) down; most top-down methods are LL.

Bottom-up parsing builds the parse tree from the leaves (terminal symbols) up; most methods are LR.

We will look primarily at 2 parsing techniques: recursive descent, which is top-down and LL, and table-driven LALR parsers, which are bottom-up and LR.