1 Introduction

- Thus far we have mainly dealt with **language recognizers**, which are devices to recognize whether a given string belongs to a language or not.
- There are also **language generators**, which generate all and only the grammatical sentences of a language.
- Let us look at the regular expression $a(a^* \cup b^*)b$ from the generation perspective:

  First output an a, then
  either output a number of a's or a number of b's, then
  output a b, and stop.

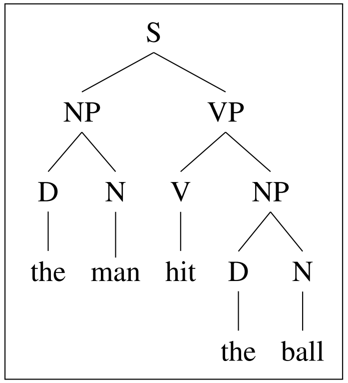
- Let's see a generator – a context-free grammar – for a tiny fragment of English:

  i. $S \rightarrow NP + VP$
  ii. $NP \rightarrow D + N$
  iii. $VP \rightarrow V + NP$
  iv. $D \rightarrow the$
  v. $N \rightarrow man, ball,$ etc.
  vi. $V \rightarrow hit, took,$ etc.

- A **leftmost**[1] derivation of the string $the + man + hit + the + ball$:

Derivation	*Rules*
S	$S \rightarrow NP + VP$
$NP + VP$	$NP \rightarrow D + N$
$D + N + VP$	$VP \rightarrow V + NP$
$D + N + V + NP$	$D \rightarrow the$
$the + N + V + NP$	$N \rightarrow man$
$the + man + V + NP$	$V \rightarrow hit$
$the + man + hit + NP$	$NP \rightarrow D + N$
$the + man + hit + D + N$	$D \rightarrow the$
$the + man + hit + the + N$	$N \rightarrow ball$
$the + man + hit + the + ball$	

[1]In every step of the derivation, you expand the leftmost nonterminal in the current string. The notion of **rightmost** derivation is defined similarly.



- The parse tree (constituent structure) contains less information than the derivation (Why? Check Example 2.2).
- A terminal string σ covered by a single node X is a constituent of type X.
- The terminal string covered by the root node is the **yield** of the parse tree.

Exercise 1.1
Introduce new rules to the grammar so that it can handle modification of NPs and VPs by PPs.

2 Grammars, derivations, parse trees

In this section we have a more formal look at the concepts we encountered in the introduction.

Definition 2.1
A **context-free grammar** G is a quadruple $\langle V_N, V_T, R, S\rangle$ where

V_N is the set of **non-terminal** symbols,

V_T is the set of **terminal** symbols,

R is the set of **rules** each of the form $A \rightarrow x$, where $A \in V_N$ and $x \in (V_N \cup V_T)^*$,

$S \in V_N$ is the **start** symbol.

A **step** in a **derivation** can be characterized as follows. For any $u, v \in (V_N \cup V_T)^*$, we write $u \Rightarrow v$ (and say v is derived from u in one step) if and only if there are strings $x, y \in (V_N \cup V_T)^*$ such that $u = xAy$, $v = xty$, and the rule $A \rightarrow t$ is in R.

We call a sequence of the form:

$$w_0 \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n$$

a **derivation** of w_n from w_0 in n **steps**, where $n \geq 0$. When $w_0 = S$, we say that G **generates** w_n in n steps.

The language of a grammar G is $L(G) = \{w \in V_T^* \mid w \text{ is generated by } G\}$.

□

Example 2.2
Let $G = \langle V_N, V_T, R, S\rangle$ where

$V_N = \{S, A, B\}$,

$V_T = \{a, b\}$,

$R = \{S \rightarrow BaBaB, B \rightarrow bB, B \rightarrow \varepsilon\}$,

S is the **start** symbol.

In specifying a grammar it is customary to give only the rules. Given the convention that start symbol is S, all the properties of the grammar are recoverable from the rules. The above grammar would be given as:

$$S \rightarrow BaBaB$$
$$B \rightarrow bB$$
$$B \rightarrow \varepsilon$$

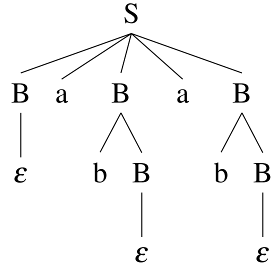
It should not be too difficult to see what language is generated by the grammar. It is the set of strings in $\{a, b\}^*$ containing exactly two a's. Now let us have a closer look to the derivation of the string *abab* by this grammar. Actually more than one derivation is possible for the given grammar and the string. Here is one:

Step	Current string	Rule applied
0	S	
1	$BaBaB$	$S \rightarrow BaBaB$
2	$aBaB$	$B \rightarrow \varepsilon$
3	$abBaB$	$B \rightarrow bB$
4	$abaB$	$B \rightarrow \varepsilon$
5	$ababB$	$B \rightarrow bB$
6	$abab$	$B \rightarrow \varepsilon$

Observe that each step consists of **rewriting** exactly one non-terminal by using exactly one rule. The order in which the non-terminals are re-written and the order in which the rules with the same left-hand-side non-terminal – rules for B for instance – is left non-specified in the grammar. These decisions are left to the mechanism which actually generates the strings. Therefore the above derivation is not the only possible derivation. Here is another one:

Step	Current string	Rule applied
0	S	
1	$BaBaB$	$S \to BaBaB$
2	$aBaB$	$B \to \varepsilon$
3	$abBaB$	$B \to bB$
4	$abBabB$	$B \to bB$
5	$abBab$	$B \to \varepsilon$
6	$abab$	$B \to \varepsilon$

Both derivations correspond to the same **parse tree**:



□

In general, it is not guaranteed that for a given grammar and a string all distinct derviations correspond to the same tree. The next example provides a case.

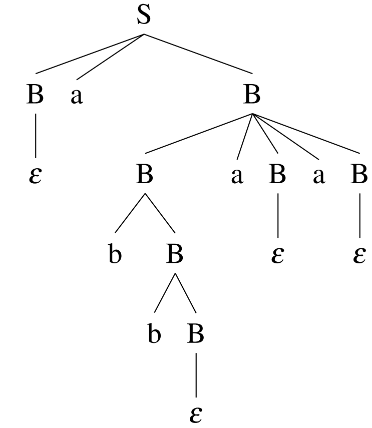
Example 2.3
First the grammar:

$$S \to BaB$$
$$B \to bB$$
$$B \to BaBaB$$
$$B \to \varepsilon$$

The grammar generates all and only the strings in $\{a,b\}^*$ that has an odd number of a's. For the string *abbaa*, the following is one possible derivation – given in a more compact notation:

$$S \Rightarrow BaB \Rightarrow aB \Rightarrow aBaBaB \Rightarrow abBaBaB \Rightarrow abbBaBaB \Rightarrow abbaBaB \Rightarrow abbaaB \Rightarrow abbaa$$

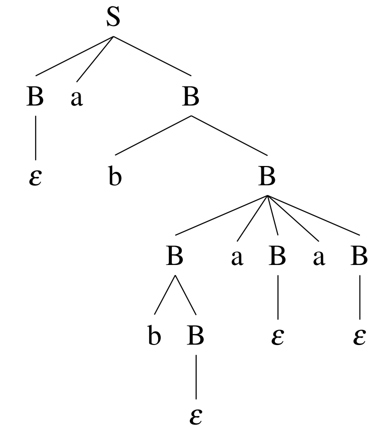
which results in the tree:



Now another derivation of the same string,

$$S \Rightarrow BaB \Rightarrow aB \Rightarrow abB \Rightarrow abBaBaB \Rightarrow abbBaBaB \Rightarrow abbaBaB \Rightarrow abbaaB \Rightarrow abbaa$$

this time resulting in a different tree:



Important note: The notation we have been using for derivations allows for some vagueness. For instance take the following intial steps of a derivation:

$$S \Rightarrow BaB \Rightarrow BaBaBaB \ldots$$

In the second step it is not clear whether the initial or the final B is rewritten as $BaBaB$. This is a crucial piece of information as it affects the resulting parse tree. If you want to be explicit about which symbol is meant to be rewritten in a derivation, you can either give the corresponding tree, or you can indicate the particular symbol by some notation. One method would be to indicate the symbol to be rewritten in the next step by an overhead dot:

$$S \Rightarrow Ba\dot{B} \Rightarrow BaBaBaB\ldots$$

□

Exercise 2.4
Write context-free grammars for the languages:

(a) $a(a^* \cup b^*)b$

(b) $\{ww^R \mid w \in \{a,b\}^*\}$

(c) $\{w \in \{a,b\}^* \mid w = w^R\}$

(d) $\{a^n b^m a^n \mid n,m \geq 1\}$

(e) $\{a^n b^n a^m b^m \mid n,m \geq 1\}$

(f) $\{a^n b^m c^m d^{2n} \mid n,m \geq 1\}$

(g) $\{a^n b^m \mid 0 \leq n \leq m \leq 2n\}$

(h) Set of strings with exactly two b's.

(i) ...with at least two b's.

(j) ...with an even length.

(k) ...with an even number of b's.

(l) ...with a positive even number of b's.

(m) ...with equal number of a's and b's.

3 CFLs and Regular Languages (FALs)

- Not every context-free language is a regular language. (We have already seen this through counterexamples.)
- Every regular language is a context-free language (proof by *direct construction*).
- **Direct Construction:** For any deterministic finite state machine $M = \langle K, \Sigma, \delta, q_1, F\rangle$, you can construct a context-free grammar $G_M = \langle K, \Sigma, R, q_0\rangle$ with

  $$R = \{q \to ap \mid \delta(q,a) = p\} \cup \{q \to \varepsilon \mid q \in F\}$$

  such that $L(M) = L(G_M)$.
- Such grammars are called **regular grammars**.

4 Closure Properties of CFLs

Union:

Given two context-free grammars $G_1 = \langle V_{N_1}, V_{T_1}, R_1, S_1\rangle$ and $G_2 = \langle V_{N_2}, V_{T_2}, R_2, S_2\rangle$ form the grammar $G = \langle V_N, V_T, R, S\rangle$ in the following way,

i. If the non-terminals of G_1 and G_2 are not disjoint, make them so (e.g. by putting primes to those of G_2).
ii. Let $R = \{S \to S_1, S \to S_2\} \cup R_1 \cup R_2$.

G will generate all and only the strings that are generated by G_1 or G_2, or both, namely $L(G) = L(G_1) \cup L(G_2)$. Therefore, CFL's are closed under union.

Concatenation:

The method of constructing G from G_1 and G_2 is the same except that $R = \{S \to S_1S_2\} \cup R_1 \cup R_2$.

Kleene star:

Given a context-free grammar $G = \langle V_N, V_T, R, S \rangle$ one can construct a grammar G' that generates $L(G)^*$ in the following way:

i. Let S' be the start symbol of G'.
ii. Let $R' = \{S' \rightarrow S'S, S' \rightarrow \varepsilon\} \cup R$.

- CFLs are *not* closed under *intersection* and *complementation*.
- Intersection of a CFL with a FAL is a CFL.

Solutions to selected exercises

2.4a $S \rightarrow aTb$
$T \rightarrow A$
$T \rightarrow B$
$A \rightarrow aA$
$B \rightarrow bB$
$A \rightarrow \varepsilon$
$B \rightarrow \varepsilon$

2.4b $S \rightarrow aSa$
$S \rightarrow bSb$
$S \rightarrow \varepsilon$

2.4c $S \rightarrow aSa \mid bSb \mid a \mid b \mid \varepsilon$

2.4d $S \rightarrow aSa \mid aBa$
$B \rightarrow bB \mid b$

2.4e $S \rightarrow AA$
$A \rightarrow aAb \mid ab$

2.4f $S \rightarrow aSdd \mid aCdd$
$C \rightarrow bCc \mid bc$

2.4g $S \rightarrow aSb \mid aSbb \mid \varepsilon$

2.4h $S \rightarrow AbAbA$
$A \rightarrow aA \mid \varepsilon$

2.4i $S \rightarrow AbAbA$
$A \rightarrow aA \mid bA \mid \varepsilon$

2.4j $S \rightarrow abS \mid baS \mid bbS \mid aaS \mid \varepsilon$

2.4k $S \rightarrow SbSbS \mid aS \mid \varepsilon$

2.4l $S \rightarrow AbAbA$
$A \rightarrow AbAbA \mid aA \mid \varepsilon$