In the lecture we saw how to convert a CFG into an equivalent one with restrictions on the rules, called "Chomsky Normal Form" (the same Chomsky of the "Chomsky Hierarchy"). This move is a useful preprocessing step for many algorithms that work on grammars. We then saw how to decide if a given string can be derived (and find a derivation) using a PTIME algorithm called the CYK-algorithm.

After this tutorial you should be able to:

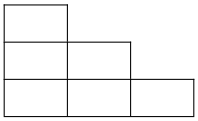
1. Convert a CFG into CNF
2. Understand how to parse a CFG
3. Understand why the naive approach of checking all long enough derivations is not as efficient as the CYK-algorithm.

Problem 1. Consider the grammar

$$S \to B;$$
$$B \to (B)B \mid \epsilon$$

1. Transform the grammar into Chomsky Normal Form (CNF).
2. Let w be the string `();` of length 3.
   Show derivations in both grammars of the string w.
3. Apply the CYK algorithm to the grammar in CNF show that w is generated by it.

The CYK table for a string w of length 3 has this form:



where the i,jth entry stores the variables A such that $Parse(A,i,j)$ is true, indicated as follows:

(0,2)		
(0,1)	(1,2)	
(0,0)	(1,1)	(2,2)

Problem 2. Apply the CYK algorithm on the inputs *aabbb*, *aabb* and *aab*:

$$S \to AX \mid AB \mid \epsilon$$
$$T \to AX \mid AB$$
$$X \to TB$$
$$A \to a$$
$$B \to b$$

Problem 3. Write pseudocode that returns, for a CFG $G = (V, \Sigma, R, S)$ and $X \in V$, whether or not $X \stackrel{+}{\Rightarrow} \epsilon$. Why is your algorithm correct?

Problem 4. Write pseudocode Unit(G, X, Y) that returns, for a CFG $G = (V, \Sigma, R, S)$ and $X, Y \in V$, whether or not $X \stackrel{+}{\Rightarrow} Y$. Why is your algorithm correct?

Problem 5. Given G in CNF, and a string w, consider an algorithm that checks all derivations of length $2|w| - 1$ to see if any derives w. If there is one, return "$w \in L(G)$", else return "$w \notin L(G)$". Argue why this approach is correct. Write high-level pseudocode for this procedure. Argue that your code does check all derivations of length at most $2|w| - 1$. What is the worst-case running time of your algorithm?

Practice problems

Problem 6. Put the following grammar into Chomsky-Normal Form using the technique from the course (the initial variable is S):

$$\begin{aligned} S &\to aSb \\ S &\to cTd \\ T &\to S \\ T &\to \epsilon \end{aligned}$$

Problem 7. Put the following grammar into Chomsky-Normal Form using the algorithm from the course:

$$\begin{aligned} S &\to aAA \mid bB \\ A &\to Baa \mid ba \\ B &\to bAA \mid ab \end{aligned}$$

Problem 8. Put the following grammar in Chomsky-Normal form.

$$\begin{aligned} S &\to AN \\ N &\to SB \\ S &\to AB \\ A &\to a \\ B &\to b \end{aligned}$$

Then apply the CYK algorithm to show that the string *aaab* is not generated by the grammar; show the CYK table. Also, describe the language generated by the grammar.

Problem 9. Convert the following grammar into CNF:

$$S \to ASA \mid aB$$
$$A \to B \mid S$$
$$B \to b \mid \epsilon$$

Then draw the CYK table for the string *abb* and decide if it is generated by the grammar or not.

Problem 10. A parsing algorithm is called *online* if it reads the input letter by letter, and after each read decides if the input read so far is in the language of the grammar.

The CYK algorithm, as presented in lectures, is not online. Why?

Devise a variant of the CYK algorithm that is online.

Problem 11.

1. Explain why the algorithm for removing epsilon rules given in lectures is correct.
2. Explain why the algorithm for removing unit rules given in lectures is correct.

Problem 12.[Hard] In this problem you will show that parsing strings of length n is harder than recognising them by a factor of at most $O(\log n)$.

The *membership problem* for CFGs is to decide, given a grammar G and a string w if $w \in L(G)$. The *parsing problem* for CFGs is to build a parse tree of G that generates w, or to say there is none. Obviously: (1) every algorithm that solves the parsing problem also solves the membership problem; (2) the CYK algorithm can be easily adapted to return parse trees.

Show that if M is an algorithm for the membership problem that runs in time $O(n^c)$ (for some constant c), then there is an algorithm M' for the parsing problem that runs in time $O(n^c \log n)$.