After this tutorial you should be able to:

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

Problem 1.

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

$$\begin{aligned} S &\to B; \\ B &\to (B)B \mid \varepsilon \end{aligned}$$

Solution 1.

1. START step: nothing to do
2. TERM step:

$$\begin{aligned} S &\to BN_1 \\ B &\to N_2BN_3B \mid \varepsilon \\ N_1 &\to ; \\ N_2 &\to (\\ N_3 &\to) \end{aligned}$$

3. BIN step:

$$\begin{aligned} S &\to BN_1 \\ B &\to N_2B_1 \mid \varepsilon \\ B_1 &\to BB_2 \\ B_2 &\to N_3B \\ N_1 &\to ; \\ N_2 &\to (\\ N_3 &\to) \end{aligned}$$

4. DEL step:

$$\begin{aligned} S &\to BN_1 \mid N_1 \\ B &\to N_2B_1 \\ B_1 &\to BB_2 \mid B_2 \\ B_2 &\to N_3B \mid N_3 \\ N_1 &\to ; \\ N_2 &\to (\\ N_3 &\to) \end{aligned}$$

5. UNIT step:

$$\begin{aligned} S &\to BN_1 \mid ; \\ B &\to N_2B_1 \\ B_1 &\to BB_2 \mid N_3B \mid) \\ B_2 &\to N_3B \mid) \\ N_1 &\to \ ; \\ N_2 &\to \ (\\ N_3 &\to \ ) \end{aligned}$$

A derivation of w in the original grammar is $S \Rightarrow B; \Rightarrow (B)B; \Rightarrow ()B; \Rightarrow ();$.

A derivation of w in the new grammar is $S \Rightarrow BN_1 \Rightarrow B; \Rightarrow N_2B_1; \Rightarrow (B_1; \Rightarrow ();$.

The CYK table for w is:

S		
B	$\emptyset$	
N_2	N_3, B_2, B_1	N_1, S
(	)	;

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

$$\begin{aligned} S &\to AX \mid AB \mid \epsilon \\ T &\to AX \mid AB \\ X &\to TB \\ A &\to a \\ B &\to b \end{aligned}$$

Solution 2.

X				
S,T				
	X			
	S,T			
A	A	B	B	B
1	2	3	4	5

Table 1: CYK on *aabbb*

S,T			
	X		
	S,T		
A	A	B	B
1	2	3	4

Table 2: CYK on *aabb*

	S,T	
A	A	B
1	2	3

Table 3: CYK on *aab*

Problem 3. The following process takes a CFG G as input and returns a set $\mathbf{X}$ of variables as output.

1. Let $\mathbf{X}$ consist of all variables A such that $A \to \epsilon$ is a rule.
2. Repeat the following until $\mathbf{X}$ no longer changes:
   (a) For every rule $A \to u$ in R such that A is a variable and u only consists of variables,
   (b) if A is not in $\mathbf{X}$ and every variable in u is in X, then
   (c) add A to X.
3. Return $\mathbf{X}$.

1. What problem does this algorithm solve? i.e., which variables end up in $\mathbf{X}$ and which do not?
2. Explain how to use this procedure to transform a CFG G that does not generate the empty-string into a CFG G' that does not have any epsilon-rules, i.e., rules of the form $A \to \epsilon$.

Solution 3. This process returns the set $\mathbf{X}$ of variables A such that $A \Rightarrow^* \epsilon$, i.e., such that A derives the empty-string.

Let G be a CFG that does not generate the empty-string. Compute the set $\mathbf{X}$ of variables that derive the empty-string. For every rule $A \to v$ where v is a string of terminals and non-terminals, add to G all rules of the form $A \to v'$ where v' is any non-empty string obtained by removing one or more occurences of variables in $\mathbf{X}$ from v. Finally, remove all rules of the form $A \to \epsilon$ from G.

Problem 4. Given G in CNF, and a string w, consider an algorithm that checks all checks all derivations of length at most $2|w| - 1$ 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?

Solution 4.

We make three claims.

First we show that if there is a derivation of w then there is one with at most $2|w| - 1$ steps.

To see this note that since G is in CNF, every step of a derivation either shortens the length of the currently derives string by 1 (by applying $A \to a$), or increases the length of the currently derived string by 1 (by applying $A \to AB$). Thus, every derivation of a non-empty string w is of length at most $2|w| - 1$. The reason is that, in the worst case, a derivation of w can apply rules of the form $A \to AB$ at most $|w| - 1$ times (since otherwise it would derive a string of length $> |w|$), and it can apply rules of the form $A \to a$ at most $|w|$ (since otherwise it would derive a string of length $> |w|$).

Second, we give high-level pseudocode for this algorithm and show it is correct and analyse its running time.

1. Input: CFG G in CNF, string w.
2. $W_0 = \{S\}$
3. Let $n = 2|w| - 1$.
4. For $i = 1, 2, \cdots, n$:
   (a) Let W_{i+1} consist of all strings uzv such that $A \to z$ is a rule of G and $uAv \in W_i$ for some strings $u, v \in (\Sigma \cup V)^*$)
5. If $w \in W_n$ return "$w \in L(G)$", else return "$w \notin L(G)$".

We claim that this procedure checks if w is generated by G by a derivation of length at most $2|w| - 1$. To see that it does this, we argue that for $i \geq 0$, the set W_i consists of all partially derives strings that result from derivations of length at most i. The base case, $i = 0$, is correct since W_0 only consists of S since it is the only derivation of length zero. For the inductive step, suppose $i > 0$, and that W_i consists of all partially derives strings that result from derivations of length at most i. Then the set W_{i+1}, constructed in line 4(a), derives strings from those in W_i by the application of a single derivation.

Finally, we claim that the running time of this procedure is, in the worst case, exponential in $|w|$. To see this, simply note that the size of W_i may be exponential in i since, in general, W_i contains at least 2^i strings, each with at least one variable in them. Indeed, $S \in W_0$, and if $uAv \in W_i$ and there is a rule $A \to BC$ then $uBCv \in W_{i+1}$.