Introduction to Language Theory and Compilation Solutions

Session 3: Introduction to grammars

Exercises

Ex. 1.

a) Right-regular grammar giving all strings made of 1s which may be followed by a single 0. 0 itself is also accepted. This language is the regular language: $1^*(0+1)$.

   1110 can be derived as $S \Rightarrow 1S \Rightarrow 11S \Rightarrow 111S \Rightarrow 1110$.

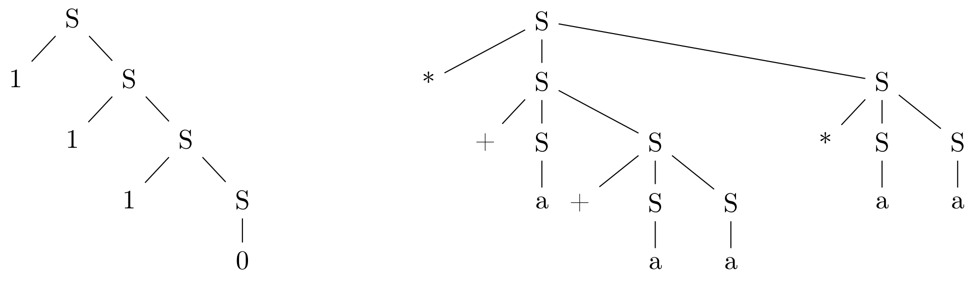
b) Context-free grammar giving all arithmetical expressions (in polish notation) using addition and multiplication with the mathematical variable called a. Not class 3 because two variables ($2 \times S$) in the set *right*.

   $* + a + aa * aa$ can be derived as $S \Rightarrow\ *SS \Rightarrow\ *{+}SSS \Rightarrow\ *{+}aSS \Rightarrow\ *{+}a+SSS \Rightarrow\ *{+}a+aSS \Rightarrow\ *{+}a+aaS \Rightarrow\ *{+}a+aa*SS \Rightarrow\ *{+}a+aa*aS \Rightarrow\ *{+}a+aa*aa$

c) Unrestricted grammar giving all strings made of abc taken at least once. Not context-sensitive because $A \to \epsilon$ and the only rule which can generate ϵ is S.

   $abcabc$ can be derived as $S \Rightarrow abcA \Rightarrow abcS \Rightarrow abcabcA \Rightarrow abcabc\epsilon = abcabc$.

In the following figure are pictured parse trees for the first two words. Note however that we cannot write a parse tree for $abcabc$, since such structure does not fit when there are more than one variable on the left-hand-side.

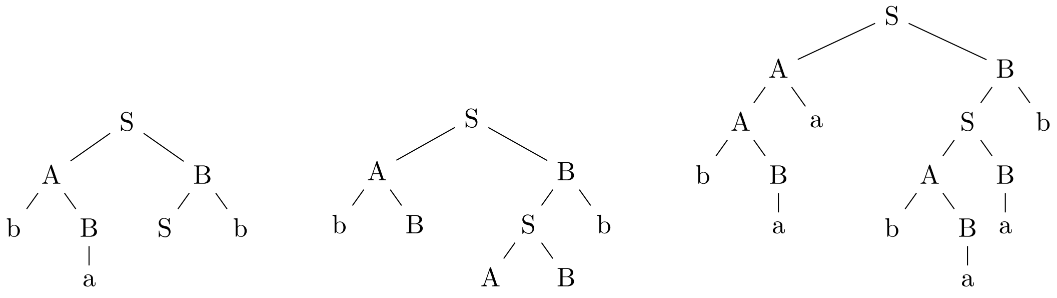


(a) 1110

(b) $* + a + aa * aa$

Ex. 2. 1. The given grammar is context-free but *not regular*: there cannot be a rule like $A \to \alpha B$ combined with a rule like $A \to B\alpha$ in a regular grammar.

2. Give the *parse tree* for



(c) *baSb* (d) *bBABb* (e) *baabaab*

3. The *leftmost* derivation of *baabaab* is: $S \Rightarrow AB \Rightarrow AaB \Rightarrow bBaB \Rightarrow baaB \Rightarrow baaSb \Rightarrow baaABb \Rightarrow baabBBb \Rightarrow baabaBb \Rightarrow baabaab$

   The *rightmost* derivation is: $S \Rightarrow AB \Rightarrow ASb \Rightarrow AABb \Rightarrow AAab \Rightarrow AbBab \Rightarrow Abaab \Rightarrow Aabaab \Rightarrow bBabaab \Rightarrow baabaab$

Ex. 3. ***Trick:*** *when you add a b, you also add a a in order to have at least* $| a | = | b |$

$$S \quad \to \quad bSa \mid aSb \mid abS \mid baS \mid Sab \mid Sba \mid Sa \mid aS \mid a$$

We can derive *baaba* from this grammar:

$$S \Rightarrow baS \Rightarrow baabS \Rightarrow baaba$$

Alternative solution: If w contains strictly more as than b, then either it contains only as, or it is of the form $bw'w''$, $w'bw''$, or $w'w''b$, where w' and w'' contain strictly more as than bs.

Indeed, let us write $w = w_1 \ldots w_n$, and let us assume that w contains at least one b. Let i be the index of the first b: then, we can already say that either $w_1 \ldots w_{i-1}$ is empty, or contains strictly more as than bs — since it contains only as.

Then, either:

- we have $i = 1$, and therefore, we have $| w_2 \ldots w_n |_a \geq | w_2 \ldots w_n |_b + 2$. Therefore, there exists an index j such that both $w_2 \ldots w_j$ and $w_{j+1} \ldots w_n$ contain strictly more as than bs, and w is of the form $bw'w''$, using the notations given above;

- or $i \neq 1$, and both $w_1 \ldots w_{i-1}$ and $w_{i+1} \ldots w_n$ contain strictly more as than bs, and then, we have $w = w'bw''$ with notations above;

- or $i \neq 1$ and either $w_1 \ldots w_{i-1}$ or $w_{i+1} \ldots w_n$ contains at least as many bs than as — it cannot be $w_1 \ldots w_{i-1}$, so it is necessarily $w_{i+1} \ldots w_n$. Then, we have $| w_1 \ldots w_{i-1} |_a \geq | w_1 \ldots w_{i-1} |_b + 2$. If $i = n$, then w is of the form $w'w''b$. Otherwise, i is not the index of the last b: we can therefore define j as the index of the next b. Then, we also have $| w_1 \ldots w_{j-1} |_a > | w_1 \ldots w_{j-1} |_b$, and we can make the same disjunction with j instead of i.

Therefore, an alternative grammar generating this language is:

$$S \quad \to \quad bSS \mid SbS \mid SSb \mid aS \mid a$$

Ex. 4.

n°	Rule	Idea
(1)	$S \rightarrow S'$	words
(2)	$S \rightarrow \varepsilon$	no words
(3)	$S' \rightarrow ABC$	$\mid a \mid = \mid b \mid = \mid c \mid$
(4)	$S' \rightarrow ABCS'$	concatenate words
(5)	$AB \rightarrow BA$	swap a and b
(6)	$AC \rightarrow CA$	swap a and a
(7)	$BA \rightarrow AB$	swap b and a
(8)	$BC \rightarrow CB$	swap b and c
(9)	$CA \rightarrow AC$	swap c and a
(10)	$CB \rightarrow BC$	swap c and b
(11)	$A \rightarrow a$	variable to terminal
(12)	$B \rightarrow b$	variable to terminal
(13)	$C \rightarrow c$	variable to terminal

cacbab can be derived from this grammar:

$S \overset{(1)}{\Rightarrow} S' \overset{(4)}{\Rightarrow} ABCS' \overset{(3)}{\Rightarrow} ABCABC \overset{(8)}{\Rightarrow} ACBABC \overset{(6)}{\Rightarrow} CABABC \overset{(8)}{\Rightarrow} CABACB \overset{(6)}{\Rightarrow} CABCAB \overset{(8)}{\Rightarrow}$
$CACBAB \overset{(13)}{\Rightarrow} cACBAB \overset{(11)}{\Rightarrow} caCBAB \overset{(13)}{\Rightarrow} cacBAB \overset{(12)}{\Rightarrow} cacbAB \overset{(11)}{\Rightarrow} cacbaB \overset{(12)}{\Rightarrow} cacbab$

(*Bonus*) $L = \{w \in \Sigma^* \mid |w|_a = |w|_b = |w|_c\}$ cannot be generated by any context-free grammar. We first give an intuition: CFGs are recognized by pushdown automata, *ie* finite automata with a stack. Such stack can be used to count and compare the number of occurrences of two letters (examine a context-free grammar recognizing $\{w \in \Sigma^* \mid |w|_a = |w|_b\}$). However, it cannot be used to compare the number of occurrences of three different letters at the same time, since the comparison is made by incrementing and decrementing the stack (which can here be seen as a counter), therefore losing information.

Now, for those who are interested, here is the formal proof. It uses the pumping lemma for CFLs:

Lemma 1. *Let L be a CFL. Then there exists $p \in \mathbb{N}$ such that for all $z \in L$ s.t. $|z| \geq p$, there exists $u, v, w, x, y \in \Sigma^*$ such that $z = uvwxy$ and:*

1. *$|vwx| \leq p$ (the middle portion is not larger than p)*
2. *$v \neq \epsilon$ or $x \neq \epsilon$ (we will pump v and x, so at least one of the two should not be empty)*
3. *For all $i \geq 0$, $uv^iwx^iy \in L$ (we pump v and x)*

The proof of this lemma is along similar lines as the one for finite automata (p corresponds to the number of states of the pushdown automaton).

Then, by taking the string $a^pb^pc^p \in L$, we have that our vwx can contain at most two distinct symbols (since it is of length at most p). Let us assume for example that $vwx = a^jb^k$ for some $j, k \geq 1$. Then it is easy to see that uv^2wx^2y does not contain as many as as bs and cs: here, there will be more as than cs (and more bs than cs). Thus, L does not satisfy the pumping lemma: it is not context-free.