

Exercise 1 (CFL):**(3 Points)**

Give context-free grammars to generate the following sets:

- a) The set of strings over alphabet $\{a, b\}$ such that each prefix of a string has at least as many a 's as b 's.
- b) The set of strings over alphabet $\{a, b\}$ such that the number of a 's is twice the number of b 's.
- c) The set of strings over alphabet $\{a, b\}$ of even positive length such that the string is NOT of the form ww .

Lösung: _____

a)

$$S \rightarrow a S b S \mid a S \mid \epsilon$$

b)

$$S \rightarrow a a S b S \mid a S a b S \mid a S b a S \mid \\ a b S a S \mid b a S a S \mid b S a a S \mid \epsilon$$

or

$$S \rightarrow S a S a S b S \mid S a S b S a S \mid S b S a S a S \mid \epsilon$$

c)

$$S \rightarrow A B \mid B A \\ A \rightarrow C A C \mid a \\ B \rightarrow C B C \mid b \\ C \rightarrow a \mid b$$

Exercise 2 (NTA):**(2 Points)**Prove soundness of the top down analysis automaton $NTA(G)$ for a grammar $G = \langle N, \Sigma, P, S \rangle$, i.e. show that for all $w \in \Sigma^*$ and all $z \in \{1, \dots, |P|\}^*$:

$$(w, S, \epsilon) \vdash^* (\epsilon, \epsilon, z) \quad \text{implies} \quad S \xRightarrow{z}_I w$$

Lösung: _____More general: $(w, \alpha, y) \vdash^* (\epsilon, \epsilon, yz)$ implies $\alpha \Rightarrow_I^z w$

Induction over $n = |z|$:

$n = 0$: then $\alpha = w$ as there may only be matching steps and thus

$$(w, w, y) \vdash^* (\varepsilon, \varepsilon, y) \text{ implies } w \Rightarrow_I^\varepsilon w$$

$n \rightarrow n + 1$: for $z = iz'$ we know from induction that if $(w', \alpha', y') \vdash^* (\varepsilon, \varepsilon, y'z')$ then $\alpha' \Rightarrow_I^{z'} w'$

Let $\alpha = uA\beta$, $w = uv$, where $u, v \in \Sigma^*$ and let $\pi(i) = A \rightarrow \gamma$, then

$$(w, \alpha, y) = (uv, uA\beta, y) \vdash^* (v, A\beta, y) \vdash (v, \gamma\beta, \underline{yi}) \stackrel{ind. hyp.}{\vdash^*} (\varepsilon, \varepsilon, \underline{yiz'})$$

The above implies:

$$\begin{array}{llll} \gamma\beta & \Rightarrow_I^{z'} & v & \text{(from induction hypothesis)} \\ u\gamma\beta & \Rightarrow_I^{z'} & uv = w & \\ \alpha = uA\beta & \Rightarrow_I^i u\gamma\beta & \Rightarrow_I^{z'} uv = w & \text{(as } \pi(i) = A \rightarrow \gamma \text{)} \\ \alpha = uA\beta & \Rightarrow_I^{iz'=z} & uv = w & \end{array}$$

Exercise 3 (Ambiguous CFG):

(2 Points)

Consider the grammar $G = (N, \Sigma, P, E)$ for arithmetic expressions.

- $N := \{E\}$
- $\Sigma := \{+, *, (,), \mathbf{id}\}$
- $E \rightarrow E + E \mid E * E \mid (E) \mid \mathbf{id}$

a) Show that the grammar is ambiguous.

b) Construct an equivalent unambiguous grammar for all arithmetic expressions with no redundant parentheses. A set of parentheses is redundant if its removal does not change the expression, e.g., the parentheses are redundant in $\mathbf{id} + (\mathbf{id} * \mathbf{id})$.

Lösung:

a) The string $\mathbf{id} + \mathbf{id} * \mathbf{id}$ has two different left derivations.

b) An unambiguous grammar which generates the same expressions without redundant parenthesis is $G' = (N, \Sigma, \mathcal{P}, E)$, where $\mathcal{P}$ is given by:

$$\begin{array}{l} E \rightarrow S \mid P \mid \mathbf{id} \\ S \rightarrow T + S \mid T * T \\ T \rightarrow \mathbf{id} \mid P \\ P \rightarrow F * P \mid F * F \\ F \rightarrow (S) \mid \mathbf{id} \end{array}$$

Exercise 4 (Lexer Implementation):

(3 Points)

The goal of this exercise is to build our own lexer which transforms an input string into a list of symbols.

Hint: as before we provide a framework which can be downloaded from the course webpage!

Implement `lexer.BacktrackingDFA.run(String)`, the method that, given an input string, performs the steps of the backtracking automaton as discussed in the lecture and returns a list of symbols.

Test your implementation! For example, given the following input

```
1      /* GCD-Computation of x and y
2         w/ WHILE */
3      int x; int y;
4      x = read();
5      y = read();
6      while ( x != y ) {
7          if (x <= y) {
8              y = y - x;
9          } else {
10             x = x - y;
11         }
12     }
13     // Output result
14     write("GCD: ");
15     write(x);
```

your implementation should generate a list of symbols like this:

```
[(INT, int), (ID, x), (SEMICOLON, ;), (INT, int), (ID, y), (SEMICOLON, ;), (ID, x),
 (ASSIGN, =), (READ, read), (LPAR, (), (RPAR, )), (SEMICOLON, ;), (ID, y), (ASSIGN, =),
 (READ, read), (LPAR, (), (RPAR, )), (SEMICOLON, ;), (WHILE, while), (LPAR, (),
 (ID, x), (NEQ, !=), (ID, y), (RPAR, )), (LBRACE, {), (IF, if), (LPAR, (), (ID, x),
 (LEQ, <=), (ID, y), (RPAR, )), (LBRACE, {), (ID, y), (ASSIGN, =), (ID, y), (MINUS, -),
 (ID, x), (SEMICOLON, ;), (RBRACE, }), (ELSE, else), (LBRACE, {), (ID, x), (ASSIGN, =),
 (ID, x), (MINUS, -), (ID, y), (SEMICOLON, ;), (RBRACE, }), (RBRACE, }),
 (WRITE, write), (LPAR, (), (STRING, "GCD: "), (RPAR, )), (SEMICOLON, ;),
 (WRITE, write), (LPAR, (), (ID, x), (RPAR, )), (SEMICOLON, ;)]
```

Lösung: _____