T-79.148 **Spring 2003**
Introduction to Theoretical Computer Science
Tutorial 8
Solutions to the demonstration problems

4. **Problem**: Design an algorithm for testing whether a given a context-free grammar $G = (V, \Sigma, P, S)$, generates a nonempty language, i.e. whether any terminal string $x \in \Sigma^*$ can be derived from the start symbol S.

   Solution:

   The following procedure ?GENERATESNONEMPTYLANGUAGE(G) takes a context-free grammar G as its input and it returns the value `true`, if the language $L(G)$ is not empty.

   ?GENERATESNONEMPTYLANGUAGE($G = (V, \Sigma, P, S)$: context-free grammar)
   $T \leftarrow \Sigma$
   repeat $|V - \Sigma|$ **times**
   for each $A \to X_1 \cdots X_k \in P$
   if $A \notin T \wedge X_1 \cdots X_k \in T^k$
   $T \leftarrow T \cup \{A\}$
   if $S \in T$
   return **true**
   else
   return **false**

   The basic idea is to start from the set $T = \Sigma$ of terminal symbols and then check whether it is possible to "retreat" to S using productions of P reversed. At each step a nonterminal A is added to the set T if there exists some rule for A such that all symbols in the right side belong to T. These steps are repeated $|V - \Sigma|$ times.

   To see why $|V - \Sigma|$ steps are enough, let us consider the word $z \in L(G)$ such that z has the smallest parse tree of all words in $L(G)$. If z has has a derivation of the form:

   $$S \to^* uAy \to^* uvAxy \to^* uvwxy$$

   where $u, v, w, x, y \in \Sigma^*$, then also $z' = uwy$ can be derived using the rules of the grammar[1]. In that case, the parse tree of z' is smaller than that of z contradicting our earlier assumption. Now we see that in the minimal parse tree of z it is not possible to have two occurrences of a nonterminal A in a single branch so we have to iterate over the set T only as many times as there are nonterminals in the grammar.

   Consider the grammar G:

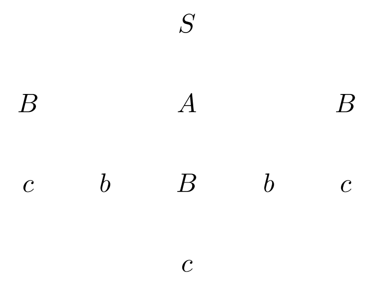
   $$\begin{aligned} S &\to BAB \mid ABA \\ A &\to aAS \mid bBa \\ B &\to bBS \mid c \end{aligned}$$

   The computation of T proceeds as follows:

   $$\begin{aligned} T_0 &= \{a, b, c\} & & \\ T_1 &= \{a, b, c, B\} & & (B \to c) \\ T_2 &= \{a, b, c, A, B\} & & (A \to bBa) \\ T_3 &= \{a, b, c, A, B, C, S\} & & (S \to BAB, S \to ABA) \end{aligned}$$

   Since $|V - \Sigma| = 3$, the algorithm terminates and $T = T_3$ so $L(G)$ is not empty. The smallest parse-tree of a $z \in L(G)$ is:

[1] Compare this with the pumping theorem of context-free languages.



5. **Problem**: Design a pushdown automaton corresponding to the grammar $G = (V, \Sigma, P, S)$, where

$$\begin{aligned} V &= \{S, (,), {}^*, \cup, \emptyset, a, b\} \\ \Sigma &= \{(,), {}^*, \cup, \emptyset, a, b\} \\ P &= \{S \to (SS), S \to S^*, S \to (S \cup S), \\ &\quad\; S \to \emptyset, S \to a, S \to b\} \end{aligned}$$

Solution: For any context-free grammar $G = (V, \Sigma, R, S)$, the corresponding nondeterministic pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ can be formed as follows:

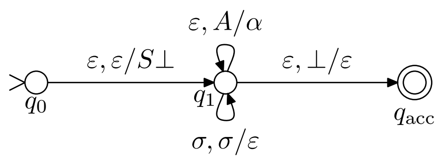
$$\begin{aligned} Q &= \{q_0, q_1, q_{\text{acc}}\} \\ \Gamma &= V \cup \{\perp\} \\ F &= \{q_{\text{acc}}\} \\ \delta &= \{\big((q_0, \varepsilon, \varepsilon), (q_1, S\perp)\big), \big((q_1, \perp, \varepsilon), (q_{\text{acc}}, \varepsilon)\big)\} && (1) \\ &\quad \cup \{\big((q_1, \varepsilon, A), (q_1, \alpha)\big) \mid (A \to \alpha \in P\} && (2) \\ &\quad \cup \{\big((q_1, \sigma, \sigma), (q_1, e)\big) \mid \sigma \in \Sigma\} && (3) \end{aligned}$$

Here symbol $\perp$ denotes the bottom of the stack.

For the grammar given in this exercise, the construction produces the following automaton:

$$\begin{aligned} Q =& \{q_0, q_1, q_{\text{acc}}\} \\ \Sigma =& \{(,), {}^*, \cup, \emptyset, a, b\} \\ \Gamma =& \{S, (,), {}^*, \cup, \emptyset, a, b, \perp\} \\ F =& \{q_{\text{acc}}\} \\ \delta =& \big\{\big((q_0, e, e), (q_1, S\perp)\big), \big((q_1, \perp, e), (q_{\text{acc}}, e)\big), \big((q_1, e, S), (q_1, (SS))\big), \\ & \big((q_1, e, S), (q_1, S^*)\big), \big((q_1, e, S), (q_1, (S \cup S))\big), \big((q_1, e, S), (q_1, \emptyset)\big), \\ & \big((q_1, e, S), (q_1, a)\big), \big((q_1, e, S), (q_1, b)\big), \\ & \big((q_1, (, (), (q_1, e)\big), \big((q_1,),)), (q_1, e)\big), \big((q_1, {}^*, {}^*), (q_1, e)\big), \\ & \big((q_1, \cup, \cup), (q_1, e)\big), \big((q_1, \emptyset, \emptyset), (q_1, e)\big), \big((q_1, a, a), (q_1, e)\big), \\ & \big((q_1, b, b), (q_1, e)\big)\big\} \end{aligned}$$

The automaton is of the form:



Let us look at how the automaton handles input $(a \cup b^*)$:

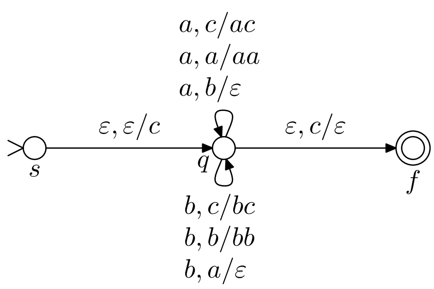
State	Input	Stack	
q_0	$(a \cup b^*)$	ε	
q_1	$(a \cup b^*)$	$S\bot$	(1)
q_1	$(a \cup b^*)$	$(S \cup S)\bot$	(2) $(S \to (S \cup S))$
q_1	$a \cup b^*)$	$S \cup S)\bot$	(3)
q_1	$a \cup b^*)$	$a \cup S)\bot$	(2) $(S \to a)$
q_1	$\cup b^*)$	$\cup S)\bot$	(3)
q_1	$b^*)$	$S^*)\bot$	(2) $(S \to S^*)$
q_1	$b^*)$	$b^*)\bot$	(2) $(S \to b)$
q_1	$^*)$	$^*)\bot$	(3)
q_1	$)$	$)\bot$	(3)
q_1	ε	$\bot$	(3)
q_{acc}	ε	ε	(1)

Note: language $L(G)$ defines all syntactically well-formed regular expressions formed over alphabet $\Sigma = \{a, b\}$.

6. **Problem**: Form the grammar corresponding to the pushdown automaton M, where $M = (Q, \Sigma, \Gamma, \delta, s, F)$:

$$
\begin{aligned}
Q =& \{s, q, f\} \\
\Sigma =& \{a, b\} \\
\Gamma =& \{a, b, c\} \\
F =& \{f\} \\
\delta =& \big\{ \big((s, e, e), (q, c)\big), \big((q, a, c), (q, ac)\big), \big((q, a, a), (q, aa)\big) \\
& \big((q, a, b), (q, e)\big), \big((q, b, c), (q, bc)\big), \big((q, b, b), (q, bb)\big) \\
& \big((q, b, a), (q, e)\big), \big((q, e, c), (f, e)\big) \big\}
\end{aligned}
$$

Solution: As a state diagram M look like this::



Determining the context-free grammar corresponding to a given pushdown automaton is a rather tedious task. The algorithm that we use here works only with *simple* pushdown automata that satisfy the following two requirements:

- If $\big((q, u, \beta), (p, \gamma)\big)$ is a transition in the pushdown automaton, then $|\beta| \leq 1$.
- If $\big((q, u, e), (p, \gamma)\big) \in \Delta$, then $\big((q, u, A), (p, \gamma A)\big) \in \Delta$ for all $A \in \Gamma$.

The requirements do not, however, reduce the expressive power of pushdown automata, since every pushdown automaton can be converted into an equivalent simple pushdown automaton (see the book for details).

The goal is to construct a grammar with nonterminals $\langle q, A, p \rangle$, where $q, p \in K$ and $A \in \Gamma \cup \{e\}$. Intuitively, the nonterminal $\langle q, A, p \rangle$ will generate all input strings on which the automaton can move from the state q to the state p while removing the symbol A from the stack.

There are four kinds of grammar rules:

1. For all $f \in F$ there is a rule $S \to \langle s, e, f\rangle$.
2. For all transitions $\big((q, u, A), (r, B_1 \dots B_n)\big) \in \Delta$, where $q, r \in K, u \in \Sigma^*, n > 0, B_1, \dots B_n \in \Gamma$ and $A \in \Gamma \cup \{e\}$, there is a rule

$$\langle q, A, p\rangle \to u\langle r, B_1, q_1\rangle\langle q_1, B_2, q_2\rangle \dots \langle q_{n-1}, B_n, p\rangle$$

   for all $p, q_1, \dots, q_{n-1} \in K$.
3. For all transitions $\big((q, u, A), (r, e)\big) \in \Delta$, where $q, r \in K, u \in \Sigma^*$ and $A \in \Gamma \cup \{e\}$, there is a rule

$$\langle q, A, p\rangle \to u\langle r, e, p\rangle$$

4. For all $q \in K$ there is a rule $\langle q, e, q\rangle \to e$.

The first rule encodes the goal to reach some final state from the initial state such that the stack is finally empty. The rules of the last form tell that no computation is needed if the automaton does not change its state. Rules of type 2 represent a sequence of transitions that move the automaton from the state q to the state p while removing the symbol A from the stack. The right side of the rule constructs the transition sequence one transition at a time. Rules of type 3 are analogous to rules of type 2.

Grammar $G = (V, \Sigma, P, S), V = \Sigma \cup \{S\} \cup \{\langle q, A, p\rangle \mid q, p \in K, A \in \Gamma \cup \{e\}\}$

$$\begin{aligned}
P = \{&S \to \langle s, e, f\rangle, && (1.)\\
&\langle s, e, s\rangle \to e, \langle q, e, q\rangle \to e, \langle f, e, f\rangle \to e, && (4.)\\
&\langle s, e, s\rangle \to e\langle q, c, s\rangle, && (2./tr.1)\\
&\langle s, e, q\rangle \to e\langle q, c, q\rangle, && (2./tr.1)\\
&\langle s, e, f\rangle \to e\langle q, c, f\rangle, && (2./tr.1)\\
&\langle q, c, s\rangle \to a\langle q, a, s'\rangle\langle s', c, s\rangle && (2./tr.2)\\
&\langle q, c, q\rangle \to a\langle q, a, q'\rangle\langle q', c, q\rangle && (2./tr.2)\\
&\langle q, c, f\rangle \to a\langle q, a, f'\rangle\langle f', c, f\rangle && (2./tr.2)\\
&\langle q, a, s\rangle \to a\langle q, a, s'\rangle\langle s', a, s\rangle && (2./tr.3)\\
&\langle q, a, q\rangle \to a\langle q, a, q'\rangle\langle q', a, q\rangle && (2./tr.3)\\
&\langle q, a, f\rangle \to a\langle q, a, f'\rangle\langle f', a, f\rangle && (2./tr.3)\\
&\langle q, b, s\rangle \to a\langle q, e, s\rangle && (3./tr.4)\\
&\langle q, b, q\rangle \to a\langle q, e, q\rangle && (3./tr.4)\\
&\langle q, b, f\rangle \to a\langle q, e, f\rangle && (3./tr.4)\\
&\langle q, c, s\rangle \to b\langle q, b, s'\rangle\langle s', c, s\rangle && (2./tr.5)\\
&\langle q, c, q\rangle \to b\langle q, b, q'\rangle\langle q', c, q\rangle && (2./tr.5)\\
&\langle q, c, f\rangle \to b\langle q, b, f'\rangle\langle f', c, f\rangle && (2./tr.5)\\
&\langle q, b, s\rangle \to b\langle q, b, s'\rangle\langle s', b, s\rangle && (2./tr.6)\\
&\langle q, b, s\rangle \to b\langle q, b, q'\rangle\langle q', b, q\rangle && (2./tr.6)\\
&\langle q, b, s\rangle \to b\langle q, b, f'\rangle\langle f', b, f\rangle && (2./tr.6)\\
&\langle q, a, s\rangle \to b\langle q, e, s\rangle && (3./tr.7)\\
&\langle q, a, q\rangle \to b\langle q, e, q\rangle && (3./tr.7)\\
&\langle q, a, f\rangle \to b\langle q, e, f\rangle && (3./tr.7)\\
&\langle q, c, s\rangle \to e\langle f, e, s\rangle && (3./tr.8)\\
&\langle q, c, q\rangle \to e\langle f, e, q\rangle && (3./tr.8)\\
&\langle q, c, f\rangle \to e\langle f, e, f\rangle && (3./tr.8)
\end{aligned}$$

Many of these rules are redundant. The rules that need to be included in the grammar can be found by starting from the rule $S \to \langle s,e,f\rangle$ and checking which rules can ever be used in a derivation. This results in the following set of rules:

$$
\begin{aligned}
P = \{S &\to \langle s,e,f\rangle \\
\langle s,e,f\rangle &\to e\langle q,c,f\rangle \\
\langle q,c,f\rangle &\to a\langle q,a,q\rangle\langle q,c,f\rangle \\
\langle q,c,f\rangle &\to b\langle q,b,q\rangle\langle q,c,f\rangle \\
\langle q,c,f\rangle &\to e\langle f,e,f\rangle \\
\langle q,a,q\rangle &\to a\langle q,a,q\rangle\langle q,a,q\rangle \\
\langle q,a,q\rangle &\to b\langle q,e,q\rangle \\
\langle q,b,q\rangle &\to b\langle q,b,q\rangle\langle q,b,q\rangle \\
\langle q,b,q\rangle &\to a\langle q,e,q\rangle \\
\langle q,e,q\rangle &\to e \\
\langle f,e,f\rangle &\to e\}
\end{aligned}
$$

The grammar can still be simplified. Let $\langle q,c,f\rangle = S, \langle q,b,q\rangle = B, \langle q,a,q\rangle = A$. This gives the result

$$
\begin{aligned}
P = \{S &\to aAS \mid bBS \mid \varepsilon \\
A &\to aAA \mid b, \\
B &\to bBB \mid a\}
\end{aligned}
$$

Appendix: Chomsky normal form and CYK-algorithm

Let's change the grammar of the last exercise into Chomsky normal form, and check with CYK-algorithm whether words abb and $abba$ belong to language $L(G)$.

A grammar is in Chomsky normal form, if the following conditions are met:

1. Only the initial symbol S can generate an empty string
2. All rules are of form $A \to BC$ or $A \to a$ (where A, B ja C are nonterminals and a a terminal symbol), except for rule $S \to \varepsilon$ (if such a rule exists).

The grammar is put into the normal form in phases.

1. **Initial symbol is removed from right side of the rules**.

   Because there are rules $S \to aAS$ and $S \to bBS$ in the grammar, let's add a new starting symbol S' and a rule $S' \to S$. The resulting set of rules is

$$
\begin{aligned}
S' &\to S, \\
S &\to aAS \mid bBS \mid \varepsilon \\
A &\to aAA \mid b, \\
B &\to bBB \mid a
\end{aligned}
$$

2. **ε-productions are removed**.

   Because in the Chomsky normal form only the initial symbol S' may generate ε, other ε rules must be removed from the grammar. We start by computing the set of erasable nonterminals: NULL:

$$
\begin{aligned}
\mathrm{NULL}_0 &= \{S\} && (S \to \varepsilon) \\
\mathrm{NULL}_1 &= \{S, S'\} && (S' \to S) \\
\mathrm{NULL}_2 &= \{S, S'\} = \mathrm{NULL}
\end{aligned}
$$

Next, the rules $A \to X_1 \cdots X_n$ are replaced by a set of rules

$$A \to \alpha_1 \cdots \alpha_2, \qquad \text{where } \alpha_i = \begin{cases} X_i, X_i \notin \text{NULL} \\ X_i \text{ or } \varepsilon, X_i \in \text{NULL} \end{cases}$$

Finally, we remove all rules of form $A \to \varepsilon$ (except for rule $S' \to \varepsilon$). As the result we get rule set[2]:

$$\begin{aligned} S' &\to S \mid \varepsilon \\ S &\to aAS \mid aA \mid bBS \mid bB \\ A &\to aAA \mid b, \\ B &\to bBB \mid a \end{aligned}$$

3. **Unit productions are removed.**
Next we remove from the grammar all rules of form $A \to B$ where both A and B are nonterminals.
First, we compute sets $F(A)$ for all $A \in V - \Sigma$:

$$\begin{aligned} F(A) &= F(B) = F(S) = \emptyset \\ F(S') &= \{S\} \end{aligned}$$

Nonterminal B belongs to set $F(A)$ exactly when we can derive B from A using only unit productions:
Rule $A \to B$ is replaced by $\{A \to w \mid \exists C \in F(B) \cup \{B\} : C \to w \in P\}$. As the result we get a set of rules

$$\begin{aligned} S' &\to aAS \mid aA \mid bBS \mid bB \mid \varepsilon \\ S &\to aAS \mid aA \mid bBS \mid bB \\ A &\to aAA \mid b, \\ B &\to bBB \mid a \end{aligned}$$

4. **Too long productions are removed.**
In the last phase we add into the grammar a new nonterminal C_σ and a rule $C_\sigma \to \sigma$ for all $\sigma \in \Sigma$ and divide all rules $A \to w$ ($|w| > 2$) into a chain of rules, all of which consist of exactly two symbols.
The Chomsky normal form for the given grammar is the following set of rules:

$$\begin{aligned} S' &\to C_a S'_1 \mid C_a A \mid C_b S'_2 \mid C_b B \mid \varepsilon \\ S'_1 &\to AS \\ S'_2 &\to BS \\ S &\to C_a S_1 \mid C_a A \mid C_b S_2 \mid C_b B \\ S_1 &\to AS \\ S_2 &\to BS \\ A &\to C_a A_1 \mid b \\ A_1 &\to AA \\ B &\to C_a B_1 \mid a \\ B_1 &\to BB \\ C_a &\to a \\ C_b &\to b \end{aligned}$$

[2] To be exact, now we should add a new initial symbol S'' and rules $S'' \to \varepsilon | S'$, but in this case we can use S' as the starting symbol without problems.

Using CYK-algorithm we can check whether word $x = x_1 \cdots x_n$ belongs to the language defined by grammar G. During the progress of algorithm we compute nonterminal sets $N_{i,j}$. Set $N_{i,j}$ includes all those nonterminals, which can be used to derive substring $x_i \cdots x_j$. We can apply dynamic programming for computing the sets:

$$
\begin{aligned}
N_{i,i} &= \{A \mid (A \to x_i) \in P\} \\
N_{i,i+k} &= \{A \mid \exists B, C \in V - \Sigma \text{ s. t. } (A \to BC) \in P \text{ and} \\
&\qquad \exists j : i \leq j < i + k \text{ s. e } B \in N_{i,j} \wedge C \in N_{j+1,i+k}\}
\end{aligned}
$$

Let's look at the grammar we got above and word *abba*. First we compute sets $N_{i,i}$, $i \leq 4$:

	$N_{i,i+k}$	$i \rightarrow$ $1 : a$	$2 : b$	$3 : b$	$4 : a$
$k \downarrow$	0	$\underline{a}bba$ $\{B, C_a\}$	$a\underline{b}ba$ $\{A, C_b\}$	$ab\underline{b}a$ $\{B, C_a\}$	$abb\underline{a}$ $\{A, C_b\}$

On each square of the array it has been denoted, which substring the square corresponds to.

Next we compute $N_{1,2}$. Now the only possible $j = 1$, so we look at sets $N_{1,1} = \{B, C_a\}$ ja $N_{2,2} = \{A, C_b\}$. The only rules of form $A \to BC$, $B \in N_{1,1}$ and $C \in N_{2,2}$, are: $\{S' \to C_a A, S \to C_a A\}$, so $N_{1,2} = \{S', S\}$. The same way we can compute sets $N_{2,3} = \{A_1\}$ and $N_{3,4} = \{S', S\}$, so the second row of the array is

	$N_{i,i+k}$	$i \rightarrow$ $1 : a$	$2 : b$	$3 : b$	$4 : a$
$k \downarrow$	0	$\underline{a}bba$ $\{B, C_a\}$	$a\underline{b}ba$ $\{A, C_b\}$	$ab\underline{b}a$ $\{B, C_a\}$	$abb\underline{a}$ $\{A, C_b\}$
	1	$\underline{ab}ba$ $\{S', S\}$	$a\underline{bb}a$ $\{A_1\}$	$ab\underline{ba}$ $\{S', S\}$	

At square $N_{1,3}$ we have to look at two alternatives,

$$
j = 1 \quad \Rightarrow \quad \begin{aligned} N_{1,1} &= \{C_a, B\} \\ N_{2,3} &= \{A_1\} \end{aligned} \qquad\qquad j = 2 \quad \Rightarrow \quad \begin{aligned} N_{1,2} &= \{S', S\} \\ N_{3,3} &= \{C_b, A\} \end{aligned}
$$

The nonterminal set corresponding to case $j = 1$ is $\{A\}$ ($A \to C_a A_1$) and that of case $j = 2$ is $\emptyset$, so $N_{1,3} = \{A\}$. We can continue the same way and and get the final table

	$N_{i,i+k}$	$i \rightarrow$ $1 : a$	$2 : b$	$3 : b$	$4 : a$
	0	$\underline{a}bba$ $\{B, C_a\}$	$a\underline{b}ba$ $\{A, C_b\}$	$ab\underline{b}a$ $\{B, C_a\}$	$abb\underline{a}$ $\{A, C_b\}$
$k \downarrow$	1	$\underline{ab}ba$ $\{S', S\}$	$a\underline{bb}a$ $\{A_1\}$	$ab\underline{ba}$ $\{S', S\}$	
	2	$\underline{abb}a$ $\{A\}$	$a\underline{bba}$ $\{S'_1, S_1\}$		
	3	$\underline{abba}$ $\{S', S, A_1\}$			

Since $S' \in N_{1,4}$, $abba \in L(G)$. But, $S' \notin N_{1,3}$, so $abb \notin L(G)$.