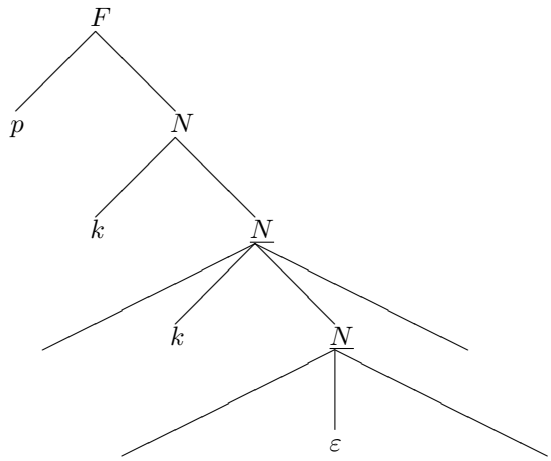
Solutions to Test 2 for CS 3350 Automata
Fall 2024

1–4. Let us consider the grammar with the starting variable F and rules $F \to pN$, $N \to kN$, and $N \to \varepsilon$. The word pkk is generated by this grammar as $\underline{F} \to p\underline{N} \to pk\underline{N} \to pkk\underline{N} \to pkk$.

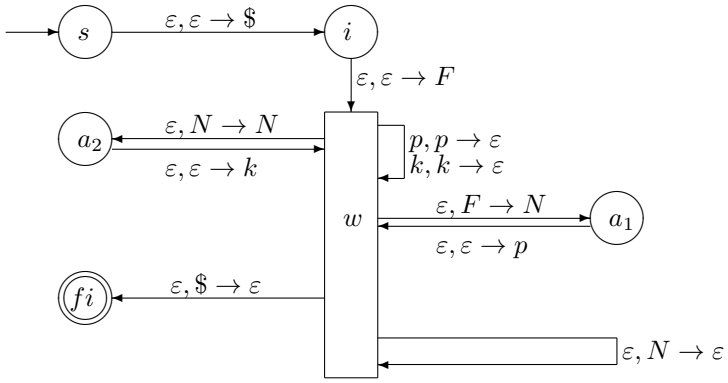
1. Show how the word pkk will be represented as $uvxyz$ according to the Pumping Lemma for context-free grammars.
2. Use a general algorithm to construct a (non-deterministic) pushdown automaton that corresponds to this grammar.
3. Show, step by step, how the word pkk will be accepted by this automaton.
4. Transform this grammar into Chomsky normal form.

Solution to Problem 1. In terms of a tree, the given derivation can be represented as follows:



Here, $u = pk$, $v = k$, $x = y = z = \varepsilon$.

Solution to Problem 2.



Solution to Problem 3. Based on the given rules, we have the following acceptance by the pushdown automaton:

					p			k			k		
s	i	w	a_1	w	w	a_2	w	w	a_2	w	w	w	fi
	\$	F	N	p	N	N	k	N	N	k	N	\$	
		\$	\$	N	\$	\$	N	\$	\$	N	\$		
				\$			\$			\$			

Solution to Problem 4.

Preliminary step. We add a new starting variable S_0 and a rule $S_0 \to F$:

$$F \to pN, \quad N \to kN, \quad N \to \varepsilon, \quad \underline{S_0 \to F}.$$

Step 0. The only inappropriate rule with right-hand side of length 0 is the rule $N \to \varepsilon$. To eliminate this rules, we add the rules $F \to p$ and $N \to k$:

$$F \to pN, \quad N \to kN, \quad S_0 \to F, \quad \underline{F \to p}, \quad \underline{N \to k}.$$

Step 1. We eliminate the rule $S_0 \to F$ by adding the rules $S_0 \to pN$ and $F_0 \to p$:

$$F \to pN, \quad N \to kN, \quad F \to p, \quad N \to k. \quad \underline{S_0 \to pN}, \quad \underline{S_0 \to p}.$$

Step 2. We add new variables V_p and V_k, add new rules $V_p \to p$ and $V_k \to k$, and replace p and k in rules in which the right-hand side has length 2 or more with, correspondingly, V_p and V_k:

$$\underline{F \to V_pN}, \quad \underline{N \to V_kN}, \quad F \to p, \quad N \to k, \quad \underline{S_0 \to V_pN}, \quad S_0 \to p,$$

$$\underline{V_p \to p}, \quad \underline{V_k \to k}.$$

Step 3. We do not have any rules with right-hand side of length 3 or more, so we already have the Chomsky normal form.

5. Give an example of an ambiguous grammar and explain, on an example, why this grammar is ambiguous, why this is not good for compiling.

Solution. Let us gave a grammar with rules $E \to a$, $E \to E+E$, and $E \to E \cdot E$. Then, the word $a + a \cdot a$ can be derived in two different ways:

$$\underline{E} \to \underline{E} + E \to a + \underline{E} \to a + \underline{E} \cdot E \to a + a \cdot \underline{E} \to a + a \cdot a;$$

$$\underline{E} \to \underline{E} \cdot E \to \underline{E} + E \cdot E \to a + \underline{E} \cdot E \to a + a \cdot \underline{E} \to a + a \cdot a.$$

This is not good, because, as we have learned, the first branching corresponds to the last operation. We want the last operation to be addition, but the second derivation promotes multiplication as the last operation.

6. Show, step by step:

   1. how the stack-based algorithm will transform the expression $(2-0)\cdot(2-4)$ into a postfix expression, and then

   2. how a second stack-based algorithm will compute the value of this expression (use integer operations in Java).

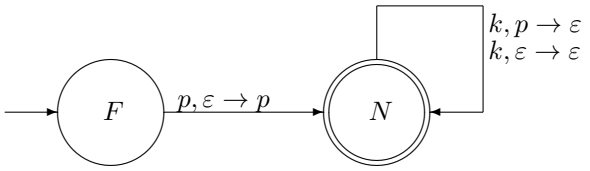
Solution. Let us show, step by step, how the above expression is transformed into the postfix form:

(	2	−	0	)	·	(	2	−	4	)	
	2		0	−			2		4	−	·
(		−		)	·	(		−		)	
		(		(		·		(		(	
								·		·	

Let us now show how this expression will be computed:

2	0	−	2	4	−	·
2	0	2	2	4	−2	−4
	2		2	2	2	
				2		

7–8. Let us consider the following pushdown automaton:



This pushdown automaton accepts the word pkk as follows:

	p	k	k
F	N	N	N
	p		

Use the general algorithm to show how this word will be generated in the corresponding context-free grammar.

Solution. We end up in the final state N, so first, we use the rule $S_0 \to A_{FN}$.

We have an intermediate stage with an empty stack, so we use the transitivity rule $A_{FN} \to A_{FN}A_{NN}$.

In the first transition, we push p, and then we pop p, so we have the two rules:

So, we form the rule $A_{FN} \to pA_{NN}k$. The variable A_{NN} describes the transition to the same state without any changes, so we can use the rule $A_{NN} \to \varepsilon$.

In the second transition, we only have one rule, so we add a fictitious rule:

This leads to the rule $A_{NN} \to kA_{NN}\varepsilon$, i.e., $A_{NN} \to kA_{NN}$. After that, we use the rule $A_{NN} \to \varepsilon$ that covers a trivial transition.

So, the derivation of the string pkk takes the following form:

$$\underline{S_0} \to \underline{A_{FN}} \to \underline{A_{FN}}A_{NN} \to p\underline{A_{NN}}kA_{NN} \to pk\underline{A_{NN}} \to pkk\underline{A_{NN}} \to pkk.$$

9-10. In the old days, farmers would eat, at breakfast (B), twice more calories than at lunch (L), and at lunch, twice more calories than at dinner (D). For example, the sequence $BBBLLD$ would fit this pattern, as well as the sequence $BBBBBBBLLLLDD$. Prove that the language S of all such sequences is not context-free.

Solution: Proof by contradiction. Let us assume that this language is context-free. Then, by the pumping lemma for context-free grammars, there exists an integer p such that every word w from this language whose length is at least p can be represented as $w = uvxyz$, where $\text{len}(vy) > 0$, $\text{len}(vxy) \le p$, and for every i, we have $uv^i xy^i z \in S$.

Let us take the word $w = B^{4p} L^{2p} D^p \in S$, in which first we have B repeated $4p$ times, then L repeated $2p$ times, and then D repeated p times. The length of this word is $4p+2p+p = 7p > p$, so this word can be represented as $w = uvxyz$.

Where is vxy? Since the length of this part does not exceed p, this word cannot contain three different digits, i.e., B's, L's, and D's – otherwise, it will have to contain all the L's between B's and D's – there are $2p$ of these symbols, and also at least one of B's and one of D's, to the total of more than p. So, we have the following possible cases:

- vxy is in B's;
- vxy is between B's and L's;
- vxy is in L's;
- vxy is between L's and D's;
- vxy is in D's.

In the first case, v and y contain only B's. So, when we go from $uvxyz$ to $uvvxyyz$, we add B's, but we do not add L's or D's; thus, the desired balance between numbers of B's, L's, and D's is disrupted, and so $uvvxyyz \notin S$ – while by pumping lemma, we should have $uvvxyyz \in S$. Thus, this case is impossible.

In the second case, v and y contain only B's and L's. So, when we go from $uvxyz$ to $uvvxyyz$, we add B's and L's, but we do not add any D's; thus, the desired balance between numbers of B's, L's, and D's is disrupted, and so $uvvxyyz \notin S$ – while by pumping lemma, we should have $uvvxyyz \in S$. Thus, this case is impossible.

Similarly, we can prove that the other cases are also not possible. So, none of the cases is possible, which means that our assumption that the language S is context-free is wrong.