

CS 475: Formal Models of Computation

Homework 3

1. Let L be the language generated by the grammar G below:

$$\begin{aligned} S &\rightarrow XY|YYY \\ X &\rightarrow Yb|\epsilon \\ Y &\rightarrow aY|X. \end{aligned}$$

Transform G into an equivalent grammar in Chomsky Normal Form. Show your work.

[Answer, sketch]

$$\begin{aligned} S_0 &\rightarrow a|b|AY|XY|YB|YY|YY_2 \\ A &\rightarrow a \\ B &\rightarrow b \\ X &\rightarrow b|Yb \\ Y &\rightarrow a|b|AY|YB \\ Y_2 &\rightarrow YY \end{aligned}$$

2. Give a grammar for each language below. You must explain the role of each nonterminal, and describe how your grammar works.

- $L_1 = \{w \in \{0, 1\}^* \mid |w| \equiv 0 \pmod{2} \text{ and } w \text{ is not a palindrome}\}$
- $L_2 = \{a^i b^j c^k \mid j = 2i \text{ or } k = 2j \text{ where } i, j, k \geq 0\}$
- $L_3 = \{w \in \{0, 1\}^* \mid \text{there is no } x \in \{0, 1\}^* \text{ such that } w = xx\}$

[Answer]

$$\begin{aligned} S &\rightarrow A|B|AB|BA \\ A &\rightarrow 0A0|0A1|1A0|1A1|0 \\ B &\rightarrow 0B0|0B1|1B0|1B1|1 \end{aligned}$$

A and B generate all words of odd size, with A generating the words with 0 as the middle character, and B the ones with 1 as the middle character. Clearly, words of odd size are in L_3 , which justifies $S \rightarrow A|B$. Now, the only way a word of even length is generated by S is with $S \rightarrow AB|BA$. Say that $S \Rightarrow AB \Rightarrow w$. We can write w as $w = u0vy1z$, in which $|u| = |v|$ and $|y| = |z|$. Consider $t = vy$ ($w = u0t1z$). We can write t as $t = pq$, in which $p = |y|$ and $q = |v|$. It follows that $w = u0pq1z$, with $|u0p| = |q1z|$, and $u0p \neq q1z$. A similar argument is made for BA .

3. A middle-pop PDA works like a PDA but the pop operation gets its stack symbol from the middle (the one closer to the top in case there are two middle symbols) of the stack instead of the top. Push operations still occur on the top of the stack. Further, the middle-pop PDA cannot view the top of the stack unless it is also the middle of the stack (that is, there are only one or two symbols in the stack). Moreover, in a middle-pop PDA, the only way to read the middle symbol is to “pop it” from the middle of the stack. Thus, in a single step, the middle-pop PDA may either (1) read an input character (or ϵ), and based on the character read, but without regard to the unviewed middle-of-stack symbol, it pushes a string γ on the top of the stack, and changes states, or else it can (2) pop and view the middle stack symbol, and based in both that symbol and the input character (or ϵ), choose to push a string on the (top of the) stack and change state. How does the class of languages accepted by middle-pop PDAs relate to the class of CFLs? As always, prove that your answer is correct.

[Answer, sketch]. We can easily prove that middle-pop PDA accepts the language $\{a^n b^n c^n \mid n \geq 0\}$, which is not a CFL (idea: push all a and b , and then pop one a and one b for each c). Therefore, if we can prove that a regular PDA can be simulated by a middle-pop PDA, then we show that the class of languages strictly contains the class of CFLs.

The issue here is that pops do not occur at the top of the stack, and this can be fixed in several ways. Say the stack looks like this:

$ab \dots cd$ middle-of-stack $wxyz \dots$

To pop the top-of-stack symbol a , we first push a special symbol, say $\#$:

$\#ab \dots cd$ middle-of-stack $wxyz \dots$

Now we keep popping and pushing the last element popped until we find $\#$:

$d\#ab \dots c$ middle-of-stack $wxyz \dots$

$cd\#ab \dots$ middle-of-stack $wxyz \dots$

$b \dots cda\#$ middle-of-stack $wxyz \dots$

$b \dots cda$ middle-of-stack $wxyz \dots$

.

By popping a , we find the top of the stack, and observe we did not modify the order of the stack.

$b \dots cd$ middle-of-stack $wxyz \dots$

.

4. Construct a PDA for the language consisting of all strings over $\{0, 1\}^*$ that are *not* of the form $\{(0^i 1^j)^k \mid 1 \leq i < j, k < 2i\}$. You must explain how your PDA works.

[Answer, sketch]. Construct a PDA for each of the *failing* conditions:

- Word starts with 1, or ends with 0.

- $i \geq j$. Check this only for the first set of 0s and 1s. Push all 0s, and pop one 0 for each 1. If there are zeroes remaining, accept.
- $k \geq 2i$. For each 0 of the first set of 0s, push two 0s into the stack. Afterward, for each transition from 0 to 1, pop a 0 from the stack. If there are zeroes remaining, accept.
- Push the first set of 0s, and then nondeterministically compare it with the 2nd, 3rd, etc, set of 0s (pop one 0 in the stack, for each new 0, and accept if 0s remain in the particular set or in the stack).
- Do the same as the previous point, but for the first set of 1s.

Once each of these five PDAs are constructed, we can put them together with ϵ transitions from a new initial state to the respective initial states.

5. A two-stack PDA is a PDA with two distinct stacks. In a single move, it reads input (or ϵ), and, based on the current top-of-stack symbols on each of its stacks, and on its current state, changes state, pushing or popping either (or both) of the stacks independently.

- Formalize the definition of a two-stack PDA as 7-tuple, explaining what each component of the tuple denotes.
- Are the class of languages accepted by regular PDAs and two-stack PDAs the same? Prove your answer [Hint: Can you think of a language *not* accepted by a two-stack PDA?]

[Answer, sketch]. This one was seen last class. A two-stack PDA is equivalent to a Turing machine. Without the knowledge of Turing machines, we can trivially simulate a single stack PDA with a two-stack PDA, and we can show that a two-stack PDA accepts the language $\{a^n b^n c^n \mid n \geq 0\}$, which it is not context-free (idea: push the a s in one stack, and the b s in the other stack, and for each c pop one a and one b).

6. As seen in class, a nondeterministic recursive automaton over an alphabet Σ is a tuple $(M, main, \{(Q_m, \Sigma \cup M, \delta_m, q_o^m, F_m)\}_{m \in M})$, in which:

- M is a finite set of module names.
- $main \in M$ is the name of the initial module.
- For each $m \in M$, $A_m = (Q_m, \Sigma \cup M, \delta_m, q_o^m, F_m)$ is an NFA over the alphabet $\Sigma \cup M$.
- For any $m, m' \in M$, $Q_m \cap Q_{m'} \neq \emptyset \iff m = m'$.

- (a) Prove that for any nondeterministic recursive automaton, there exists a PDA that can simulate it.

[Answer, sketch]. In class we proved that for each PDA there is a CFG. With this observation, if we prove how to construct a CFG from a RA, then we show the equivalence between PDAs and RAs. The main idea here is to have each module as a variable of the grammar, with the $main$ module as the starting symbol, and with edge transitions that are not modules as terminals. For each module, a path from the initial state to an accepting state becomes a production rule of the grammar (we would have to iron out some details about loops in this paths, but that is the general idea).

- (b) Design a nondeterministic recursive automaton for the language L containing all strings $w \in \{0, 1\}^*$ that have an equal number of 0s and 1s.

There is a single module, *main*:

