Tutorial 6: Context-Free Tutorial

---

1. Solve the following problems. Assume the alphabet to be $\Sigma = \{a, b, c\}$

   (a) Consider the language of non-palindromes (words that are not palindromes). A palindrome is a word that spells the same way forward and backwards. For example, 'abcba' is a palindrome but 'abbaa' is not). Describe a PDA and a CFG for the language.

   (b) Consider the language $\{w \mid w \neq uu \text{ for any } u \in \Sigma^*\}$. Describe a PDA and a CFG for the language.

2. Determine if the following languages are context-free or not. If yes, provide a CFG and PDA for the same, else prove, using the Pumping Lemma, that they are not Context Free

   (a) $L_1 = \{w \mid w = uu \text{ for any } u \in \Sigma^*\}$

   (b) $L_2 = \{0^p | \text{p is prime}\}$

   For more on L_2, look at the takeaway problems

3. **Deterministic Context-Free Languages**

We know that Context-Free Languages (CFL) are accepted by Push-Down Automata (NPDA), where we had allowed non-determinism in PDA transitions. In this question, we will explore Deterministic Push-Down Automata (DPDA). Recall that a (not necessarily deterministic) PDA M can be defined as a 7-tuple:

$$M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$$

h where Q is a finite set of states. Σ is a finite set of input symbols. Γ is a finite set of stack symbols. $q_0 \in Q$ is the start state. $Z_0 \in \Gamma$ is the starting stack symbol. $A \subseteq Q$, where A is the set of accepting, or final, states δ is a transition function, where $\delta : (Q \times (\Sigma \cup \varepsilon) \times \Gamma) \to \mathcal{P}(Q \times \Gamma^*)$ Here, $\mathcal{P}(X)$ is the power set of a set X, and ε denotes the empty string. We say that M is a deterministic PDA (or DPDA) if it satisfies both the following conditions:

For any $q \in Q, a \in \Sigma \cup \varepsilon, x \in \Gamma$, the set $\delta(q, a, x)$ has at most one element.

For any $q \in Q, x \in \Gamma$, if $\delta(q, \varepsilon, x) \neq \emptyset$, then $\delta(q, a, x) = \emptyset$ for every $a \in \Sigma$.

We call the languages accepted by DPDAs as DCFLs (Deterministic Context Free Languages). We have studied in class that PDAs can accept by *empty stack* or by *final state*, and that these provide equivalent accepting power. Interestingly, this is not so for DPDAs, so we need to make up our mind about which acceptance criterion to use. For purposes of this question, we will use acceptance by *final state*. We investigate acceptance by empty stack in a takeaway question at the end of this tutorial.

1. Consider the language of balanced parentheses. A string of parentheses is balanced if the number of opening parentheses in any proper prefix is at least as much as the number of closing parenthesis in the same prefix. Also, the total number of opening and closing parenthesis in the entire string must be equal.

   Draw a DPDA (accepting by final state) that accepts the language of balanced parentheses strings.

2. Construct a deterministic PDA for the complement of the above language. Does this give you an idea why DCFLs (recognized by DPDAs by final state) are closed under complementation?

4. **Takeaway: The Curious Case of the Unary Alphabet** Prove that any language over a unary alphabet (the alphabet has exactly one element) is context-free if and only if it is regular.

5. **Takeaway: Expressions in Intermediate Code**
   Consider the following language for expressions in some (familiar) programming languages

   $\langle expr \rangle$ ::= $\langle term \rangle$ '+' $\langle term \rangle$
   | $\langle term \rangle$

   $\langle term \rangle$ ::= $\langle factor \rangle$ '*' $\langle factor \rangle$
   | $\langle factor \rangle$

   $\langle factor \rangle$ ::= '(' $\langle expr \rangle$ ')'
   | $\langle number \rangle$

   $\langle number \rangle$ ::= [0-1]+

   Now, though expressions can be a sum of as many terms, it is essential, during an intermediate step of compilation, that every expression must be a sum of at most two terms and every term must be a product of at most two terms. Draw a Deterministic PDA (Definition provided in an earlier question) to recognise strings which are of the form as stated above. Note that the alphabet is $\Sigma = \{0, 1, (,), *, +\}$.

6. **Takeaway: Null-stack DPDAs**

   In Problem 3, we used acceptance by final state for a DPDA. Let's see what happens if we now allow a DPDA to accept by emptying its stack (regardless of which state it is in, when the stack becomes empty). We will call such a DPDA a *null-stack DPDA*, i.e. it's a DPDA just like we had earlier, but it accepts by emptying its stack.

   1. Prove that no null-stack DPDA can accept the language of balanced parentheses. Recall that a string of parentheses is balanced if the number of opening parenthesis in any prefix of the string is at least as much as the number of closing parenthesis in the same prefix, and the total number of opening and closing parenthesis ar equal.

      [Hint:] Can a null-stack DPDA accept two strings u and $u.v$, where one is a proper prefix of the other?

      Note: This is quite damaging news in the DPDA world, since we saw in Problem 3 that the language of balanced parentheses can be accepted by a DPDA accepting by final state. The above proof should now convince you that unlike normal PDAs, acceptance by final state and acceptance by empty stack are not equally powerful in the DPDA world.

   2. A string is said to be *minimally balanced parentheses* if the number of opening parenthesis in any proper prefix is strictly more than the number of closing parenthesis in the same prefix, and the total number of opening and closing parenthesis in the entire string are equal. Thus, $(()())$ is a minimally balanced parentheses string, but $()()$ and ε are not. Any string of balanced parentheses can be written as either ε or a concatenation of a finite number of minimally balanced parentheses strings. Show that for every given value of $k > 0$, we can construct a null-stack DPDA that accepts the language of balanced parentheses strings containing exactly k minimal valid parentheses substrings. Can we construct a null-stack DPDA if we want to accept the language of balanced parentheses containing upto (instead of exactly) k minimally valid parentheses substrings?