CSCI 1010 - Fall'25 Theory of Computation Lorenzo De Stefani

HW4

Due: n/a

Reminder: Submit your assignment on Gradescope by the due date. Submissions must be typeset. Each page should include work for only one problem (i.e., make a new page/new pages for each problem). See the course syllabus for the late policy.

While collaboration is encouraged, please remember not to take away notes from any labs or collaboration sessions. Your work should be your own. Use of other third-party resources is strictly forbidden.

Please monitor Ed discussion, as we will post clarifications of questions there.

Problem 1

1. Prove that the set of Context-Free Languages is not closed under the complement operator.
2. Prove that the set of Deterministic Context-Free Languages is not closed under the union operator.
3. If A and B are languages, define

   $$A \circ B = \{xy | x \in A \text{ and } y \in B \text{ and } |x| = |y|\}.$$

   Given A and B are regular languages, argue whether $A \circ B$ is a CFL.

Solution:

1. Prove that the set of Context-Free Languages is not closed under the complement operator.

   We proved in class that the set of Context-Free Languages is closed under the union operator, and we considered examples that proved that the set of Context-Free Languages is not closed under intersection. We will use these facts to prove that it is not closed under the complement operator.

   Assume by way of contradiction that the set of CFLs is closed under the complement operator. Since for any two CFLs L_1 and L_2, $L_1 \cup L_2$

is a CFL and CFLs are closed under the complement operator, $\overline{L_1 \cup L_2}$ is also in the set of CFLs. By de Morgan's Laws, this is equivalent to saying that $\overline{L_1} \cap \overline{L_2}$ is also a CFL.

Since CFLs are closed under complement by assumption, every CFL can be realized as the complement of another CFL. As such, we have just proven that the set of CFLs is closed under intersection, which is a contradiction. Our assumption that the set of CFLs is closed under complement must be false.

2. Prove that the set of Deterministic Context-Free Languages is not closed under the union operator.

   Consider the language $L = \{a^i b^j c^k : i = k \text{ or } i = j\}$. It is the union of the two languages $L_1 = \{a^i b^j c^k : i = k\}$ and $L_2\{a^i b^j c^k : i = j\}$. We discussed in class that L_1 and L_2 were deterministic context-free languages (by providing a PDA) and that L was a non-deterministic context-free language. Hence, the set of Deterministic Context-Free Languages is not closed under the union operator.

3. Circ

   Since A and B are regular, we know there are DFAs D_A and D_B that decide A and B.

   Using D_A and D_B, we can construct a non deterministic PDA in the following way:

   (a) The start state of the PDA has a transition to the start state of D_A where "$" is pushed to the stack without consuming any characters.

   (b) We append the rest in of D_A to the start state of D_A.

   (c) There is a non deterministic transition from every accept state of D_A to the start state of D_B.

   (d) The stack pushes an "a" every time D_A consumes a character. Once it transitions to D_B it pops and "a" every time D_B consumes a character. If the stack is empty and the string hasn't been consumed move to a sink state.

   (e) The PDA accepts a string if and only if all the characters have been consumed and it is in an accept state of D_B and the stack is empty.

   For full credit you would also need to provide a proof of correctness.

Problem 2

Let

$L_1 = \{w \in 1, \#^* | w = t_1 \# t_2 \# \ldots \# t_k \text{ for } k \geq 0, \text{ each } t_i \in 1^*, \text{ and } t_i \neq t_j \text{ whenever } i \neq j\}$.

- Prove that L_1 is not a Context-Free language.
- Prove that L_1 is a Turing-Decidable language. If you choose to do so by showing an accepting Turing Machine M_1 such that $L_1 = L(M_1)$, you need to show a proof of correctness.

Solution: Assume towards contradiction that Y is a CFL. By the pumping lemma the string

$w = 1^p \# 1^{p+1} \# 1^{p+2} \# \ldots \# 1^{2p-2} \# 1^{2p-1} \# 1^{2p} \# 1^{2p+1} \# 1^{2p+2} \# \ldots \# 1^{3p-2} \# 1^{3p-1} \# 1^{3p}$

can be partition as $w = uvxyz$ such that $uv^ixy^iz \in L$ for any $i \geq 0$.

By rule 3 of the Pumping lemma $|vxy| \leq p$. This means the string vxy contains at most one "#" and is spread out across at most two different sets of 1's. We break down the analysis in three cases

- $v = 1^a \# 1^b$ for $0 \leq a, b \leq p$. Then the string

$$\begin{aligned} w' &= uv^3xy^3z && (1)\\ &= u1^a \# 1^b 1^a \# 1^b 1^a \# 1^b xyz && (2)\\ &= u1^a \# 1^{a+b} \# 1^{a+b} \# 1^b xyz && (3)\\ &\notin Y && (4) \end{aligned}$$

  As you can see, we have different subsets with an equal amount 1's, which is not allowed. The case for $y = 1^a \# 1^b$ is analogous.

- $v = 1^a$, $y = 1^b$ such that, by rules 2 and 3 of the Pumping Lemma, $0 < a + b \leq p$. For this case, assume x does not include "#". Then, vxy is a substring of some $t_i = 1^c$. If $c \leq 2p$, we can pump once and end with $t_i' = 1^{c'}$ where $c' = i + a + b$, meaning $p < c' \leq 3p$ and $c' \neq c$. In each of these cases, t_k is equal to some t_j, where $k \neq j$, which is not allowed for strings in Y. If $c > 2p$, then we can delete v and y, leaving us with $t_k' = 1^{c'}$, where $c' = i - a - b$, meaning $2p - p = p < c' \leq 2p = 3p - p$ and $c' \neq c$. This gives the same property of a t_k where t_k is equal to some t_j and $k \neq j$.

- $v = 1^a$, $y = 1^b$ such that, by rules 2 and 3 of the Pumping Lemma, $0 < a + b \leq p$. For this case, assume x does include "#". Then, v is a substring of $t_i = 1^c$ and y is a substring of $t_{i+1} = 1^{c+1}$ for $p \leq c < 3p$. If either $v = \epsilon$ or $y = \epsilon$ this reduces to the previous case. If $c \leq 2p$, then $w'' = uv^2xy^2z$ will be such that t_i is replaced by $t_k = 1^{c'}$, where $p < c' \leq 3p$, meaning t_k is equal to some other t_j where $k \neq j$. If $i > 2p$, then $w'' = uv^0y^0z$ will be such that t_{i+1} is replaced by a $t_{i'} = 1^{c'}$ such that $p* \leq c' < 3p$, giving us the same property as before. In both cases, $w'' \notin Y$.

Problem 3

Let $L_2 = \{w \in \{0,1\}^* | w = w^R$ and there are the same number of 0's as 1's in $w\}$.

- Prove that L_2 is not a Context-Free language.
- Prove that L_2 is a Turing-Decidable language. If you choose to do so by showing an accepting Turing Machine M_1 such that $L_1 = L(M_1)$, you need to show a proof of correctness.

Solution:
Assume that the language B is context free. This means it satisfies the pumping lemma for context free languages.

For this, consider the string in the language B $s = 0^p1^{2p}0^p$. By the pumping lemma for context free languages, there must be a way to split s into $s = uvxyz$ such that

1. $uv^ixy^iz \in B$ $d\forall$ i
2. $|vy| > 0$
3. $|vxy| \leq p$

By the second and third conditions, the cases for vxy are then :

1. v and y both contain only 0s (or one of them is empty) from either side of the 1s of the 1s or the same thing but with all 1s. In these cases, a string $s' = uv^2xy^2z$ will yield more 0s than 1s or more 1s than 0s, breaking the first condition of the pumping lemma for context free languages.

2. vxy contains some 0s and some 1s, where we can generalize for both sides of the 1s. If the number of 1s and 0s contained within v or y are not equal, then we can pump as in the first case and end up with more 0s than 1s or more 1s than 0s. Otherwise, pumping as we did before will still work, as it will yield more 0s on one side than the other, making the resulting string not a palindrome. This then breaks the first condition of the pumping lemma for context free languages.

   Since it is shown that there is no way to split the string $0^p1^{2p}0^p$ such that it can satisfy all three conditions of the pumping lemma for context free languages, we arrive at a contradiction and reject our initial assumption that B is context free.

   Now, we must show that L_1 is Turing decidable. A brief justification for this is to construct a Turing machine that matches the left of the input string with the right of the input string and then also doing another check for if the number of 0s is equal to the number of 1s by matching 0s and 1s in the input string.