

Solutions for cs154 Homework #3

Easy Problem 1

Part (a):

$$\begin{aligned} S &\rightarrow TX \\ T &\rightarrow 0T0 \mid 1T1 \mid \#X \\ X &\rightarrow 0X \mid 1X \mid \epsilon \end{aligned}$$

Intuition: X generates $(0 + 1 + \#)^*$. T generates strings of the form $u\#Xv$, where $u = v^R$. On the whole, S generates TX .

Part (b):

$$\begin{aligned} S &\rightarrow XTX \\ T &\rightarrow 0T0 \mid 1T1 \mid \# \mid \#X\# \\ X &\rightarrow 0X \mid 1X \mid \#X \mid \epsilon \end{aligned}$$

Intuition: X generates $(0 + 1 + \#)^*$. T generates either $u\#v$ or $u\#X\#v$, where $u = v^R$. On the whole, S generates XTX .

Easy Problem 2(a), 2(b)

Drawn later.

Easy Problem 3

Drawn later.

Easy Problem 4

Drawn later.

Easy Problem 5

(a) $L = \{0^i\#1^i \mid i \geq 1\}$ is **Context Free**. This is because there is a simple PDA for this language. The PDA would simply push all 0's, jump to another state when it sees # and then pop as many 0's as the number of 1's before accepting.

$\bar{L} = \overline{0^*\#1^*} \cup \{\#\} \cup \{0^i\#1^j \mid i \neq j \text{ and } i, j \geq 0\}$ is **Context Free**.

Note that $\overline{0^* \# 1^*}$ denotes the complement of the regular expression $0^* \# 1^*$. The language $\{0^i \# 1^j \mid i \neq j \text{ and } i, j \geq 0\}$ is easily seen to be Context Free because there is a simple PDA to recognize it. The union of Regular languages and Context Free languages is Context Free.

(b) $L = \{xy \mid x \neq y\}$ for $\Sigma = \{0, 1\}$ is **Regular**

Well, $x = \epsilon$ and any non empty $y \in \Sigma^*$ satisfy the condition $x \neq y$. So $L = \Sigma^* - \{\epsilon\}$.

\overline{L} is **Finite** as it is $\{\epsilon\}$.

(c) $L = \{xy \mid |x| = |y| \text{ and } x \neq y\}$ with $\Sigma = \{0, 1\}$ is **Context Free**.

The following CFG generates L :

$$S \rightarrow UV \mid VU$$

$$U \rightarrow 0U0 \mid 1U0 \mid 0U1 \mid 1U1 \mid 0$$

$$V \rightarrow 0V0 \mid 1V0 \mid 0V1 \mid 1V1 \mid 1$$

$\overline{L} = \{w \mid \text{the length of } w \text{ is odd}\} \cup \{ww \mid w \in \Sigma^*\}$ is **Not Context Free**.

We need to be careful when classifying \overline{L} . The language $\{w \mid \text{the length of } w \text{ is odd}\}$ is clearly regular. We know that the language $\{ww \mid w \in \Sigma^*\}$ is non-CFG. However, the union of a CFG with a Regular language might be Regular!

To prove \overline{L} non Context Free, we use the fact that the intersection of a CFG and a Regular language is a CFG. If \overline{L} is a CFG, then $\overline{L} \cap \{w \mid \text{the length of } w \text{ is even}\}$ must be a CFG. However, this is simply the language $\{ww \mid w \in \Sigma^*\}$ which is known to be non-CFG. Therefore, \overline{L} must have been a non-CFG.

(d) $L = \{x \# x \# x \mid |x| \leq 10^{10}\}$ is **Finite**

The number of strings in L is finite for any Σ . All finite languages are also regular.

\overline{L} is **Regular** since the complement of a regular language is regular.

(e) $L = \{x \# x \# x \mid |x| \geq 10^{10}\}$ is **not Context Free**

Let us assume that L is context free. Then, taking the union of L with the regular language in Part (d) should result in a CFG. This new language is $\{x \# x \# x \mid |x| \geq 0\}$. However, this language can be shown to be non-CFG using the Pumping Lemma. Therefore, L must have been non-CFG as well.

$\overline{L} = \overline{(0+1)^* \# (0+1)^* \# (0+1)^*} \cup \{x \# y \# z \mid |x| < 10^{10}\} \cup \{x \# y \# z \mid |y| < 10^{10}\} \cup \{x \# y \# z \mid |z| < 10^{10}\} \cup \{x \# y \# z \mid x \neq y\} \cup \{x \# y \# z \mid y \neq z\} \cup \{x \# y \# z \mid x \neq z\}$ is **Context Free**.

The expression $\overline{(0+1)^* \# (0+1)^* \# (0+1)^*}$ is a regular language. The next three languages are also regular. The last three are Context Free (there are simple PDAs to recognize them). All

regular languages are also CFGs. The union of CFGs is a CFG. So \bar{L} is context free.

(f) $\{x\#y \mid x, y \text{ are binary numbers and } x \bmod 5 = y \bmod 5\}$ is **Regular**.

It is possible to construct a DFA for this language using DFAs for C_n (HW #1 Problem).

\bar{L} is **Regular** since complementation preserves regularity.

Problem 1 (10 Points)

(a) Proof by Pumping Lemma

$\forall p \geq 1$

$\exists s \in L$ such that $|s| \geq p$. We choose $s = 0^p 1^p 0^p 1^p$.

$\forall u, v, w, x, y : s = uvwxy$ and $|vwx| \leq p$ and $|vx| \geq 1$

$\exists k \geq 0$ such that $uv^kwx^ky \notin L$.

We show that no matter how u, v, w, x, y are chosen, we can always find k so that $uv^kwx^ky \notin L$. Note that $|vwx| \leq p$ while the length of $s = 0^p 1^p 0^p 1^p$ is $4p$. This means that vx cannot contain some 0's from the first and some from the second 0^p . Similarly, vx cannot contain some 1's from the first 1^p and some from the second 1^p . Thus, the choice $k = 0$ always works, i.e., $uv^0wx^0y \notin L$.

(b) Proof by Pumping Lemma

$\forall p \geq 1$

$\exists s \in L$ such that $|s| \geq p$. We choose $s = 0^p 1^p \# 0^p 1^p$.

$\forall u, v, w, x, y : s = uvwxy$ and $|vwx| \leq p$ and $|vx| \geq 1$

$\exists k \geq 0$ such that $uv^kwx^ky \notin L$.

We show that no matter how u, v, w, x, y are chosen, we can always find a k so that $uv^kwx^ky \notin L$. Note that $|vwx| \leq p$ while the length of $s = 0^p 1^p \# 0^p 1^p$ is $4p + 1$. Here is a case by case analysis: If vx contains $\#$, then $k = 0$ produces a string not in L . Otherwise, we have three cases:

- Both v and x lie on the left of $\#$. In this case, $k = 2$ works because the new string has form $w\#x$ such that w is longer than x . So w is no longer a substring of x .
- Both v and x lie on the right of $\#$. In this case, $k = 0$ works because the new string has form $w\#x$ such that w is longer than x . So w is no longer a substring of x .
- v and x lie on either side of $\#$ and both $|v|, |x| \geq 1$. In this case, $k = 2$ works because the new string has the form $0^p 1^a \# 0^b 1^p$ where $a \geq p$. This string does not lie in L .

(b) The intersection of a CFG with a Regular language is a CFG. Consider $L \cap (a+b)^* \# (a+b)^*$.

This language is $\{x_1 \# x_2 \mid x_1 = x_2\}$. This language can be shown to be non-CFG using the Pumping Lemma. The proof is along the same lines as part (a) above.

Problem 2 (10 Points)

(a) Consider $L = \{ww \mid w \in \Sigma^*\}$ where $\Sigma = \{0,1\}$. It is easy to show that L is a non CFG using the Pumping Lemma for CFGs. Now, $\bar{L} = \{z \mid |z| \text{ is odd}\} \cup \{xy \mid |x| = |y| \text{ but } x \neq y\}$ where $x, y, z \in \Sigma^*$. The first set is regular. The second set is a CFG.

(b) Let $\Sigma = \{a, b, 0\}$. Let $L = (a + b)^* \cup \{zz0^i \mid z \in (a + b)^* \text{ and } i \geq 1\}$. We have to show the following:

$$\exists p \geq 1$$

$$\forall s : |s| \geq p \text{ and } s \in L$$

$$\exists u, v, w, x, y : s = uvwxy \text{ and } |vx| \geq 1 \text{ and } |vwx| \leq p$$

$$\forall k \geq 0 : uv^kwx^ky \in L.$$

If the string has the form $(a + b)^*$, it can easily be pumped. Otherwise, the string has the form $zz0^i$ where $i \geq 1$. Therefore, if we choose $u = zz$, $v = w = \epsilon$, $x = 0$ and $y = 0^{i-1}$, it turns out that $\forall k : uv^kwx^ky \in L$.

How do we show that L is non-CFG. The intersection of a CFG with a Regular Language yields another CFG. Consider $L' = L \cap (a + b)^*0 = \{zz0 \mid z \in (a + b)^*\}$. Language L' can easily be shown to be non-CFG using the Pumping Lemma for CFGs. Therefore, L must have been a non-CFG.

The proof that L is non-CFG could also proceed by using the fact that the union of two CFG's is a CFG. (Or that the concatenation of two CFGs is a CFG).

Problem 3 (10 Points)

TM drawn later.

Problem 4 (10 Points)

Let T_1 and T_2 denote a pair of Turing machines.

(a) *Union*: A string should be accepted by T if it is accepted by either T_1 or T_2 . Therefore T , at the beginning, makes a non-deterministic choice whether to simulate T_1 or T_2 . T accepts if either machine accepts.

Here is another design: T has two tapes. It makes a copy of the input (which is on one of the tapes) onto the second tape. Then it simulates T_1 on the first tape. If T_1 rejects the string, T simulates T_2 on the second tape. T accepts if either T_1 or T_2 accepts. Note that this idea does not work for Problem 5(a).

Note: If you write that " T simulates T_1 and if T_1 rejects, we jump to the start state of T_2 ", your answer is wrong. Please note that when you start T_2 , the input tape could have been completely

garbled by T_1 . Similarly, if you draw a figure with a box showing T_1 with an ϵ transition to T_2 , the diagram is not very meaningful. Such a diagram for DFA's makes sense, but for Turing Machines, remember that the first machine could overwrite any part of the tape.

(b) *Concatenation*: T should accept a string if it can be split into two strings such that the first string is accepted by T_1 and the second part by T_2 . Well, T is non-deterministic and has two tapes. The input lies on the first tape. T splits the input into two strings, one string per tape. The point where the input is split is chosen non-deterministically. T then simulates T_1 on the first string using the first tape. If T_1 accepts, T starts simulating T_2 on the second string on the second tape. T accepts if T_2 also accepts.

Notes: If you write that we first run T_1 and then T_2 if T_1 accepts, your solution is wrong. Similarly, if you draw a diagram with ϵ transition from q_{reject} or q_{accept} of T_1 to the start state of T_2 , the solution is wrong.

(c) *Star*: T should accept a string if it can be split into smaller strings such that each of the smaller strings is accepted by T_1 . Here is a machine that works: T is non-deterministic with two tapes. It breaks the input string w into k pieces $x_1, x_2, x_3, \dots, x_k$ such that $w = x_1x_2 \dots x_k$. The value of k and the pieces are chosen non-deterministically. The machine iterates i from 1 to k , copying x_i from the first tape onto the second and running T_1 on x_i . If T_1 accepts all x_i , T accepts w .

(d) *Complementation*: We simply swap q_{accept} with q_{reject} .

(e) *Intersection*: T should accept a string if both T_1 and T_2 accept it. So T is a machine with two tapes. First, the input is copied onto the second tape. Then T_1 is simulated on the first tape. When T_1 accepts, T_2 is simulated on the second tape. If T_2 also accepts, T accepts.

Problem 5 (10 Points)

The constructions for this problem are analogous to those for Problem 4. Care must be taken that we do not *pipeline* T_1 and T_2 for union.

Easy Problem 3 and Moderate Problem 3 require state diagrams of TM's. These are drawn later.