Foundations of Computation

Assignment 3

January 3, 2022

Q1: Construct context-free grammars that generate the following languages (give your answers as a set of rules):

(a) $\{a^k w \mid w \in \{a,b\}^*, |w| = k, k \geq 0\}$

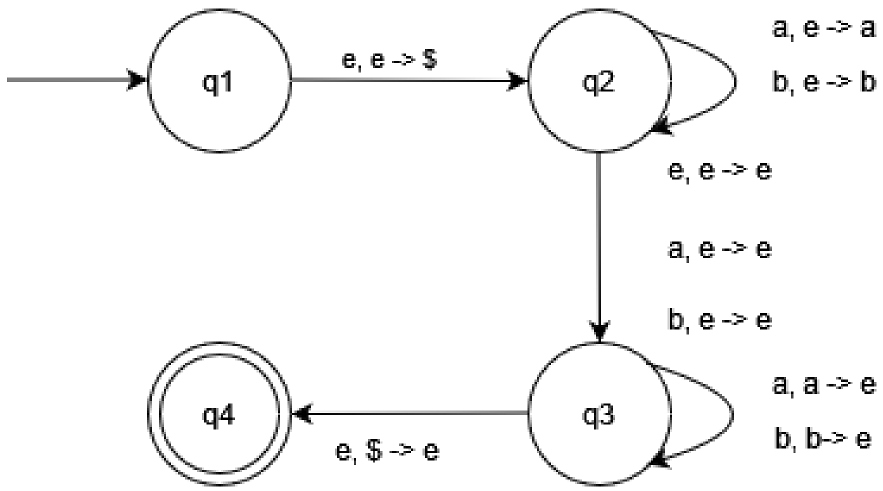
$S_1 \to aS_1a \mid aS_1b \mid \epsilon$

(b) $\{a^n b^{n+k} a^k \in \{a,b\}^* \mid n \geq 0, k \geq 0\}$

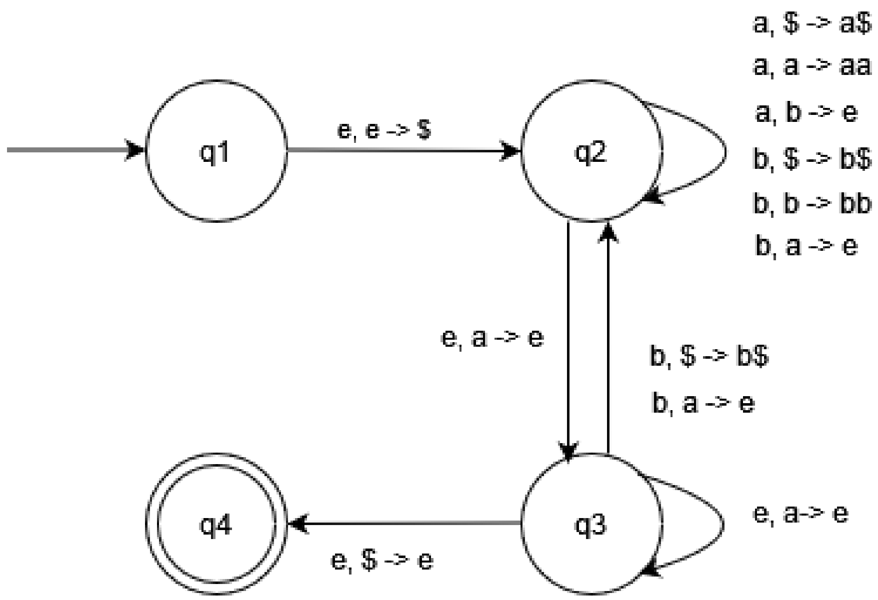
$S_1 \to S_2S_3 \mid \epsilon$
$S_2 \to aS_2b \mid \epsilon$
$S_3 \to bS_3a \mid \epsilon$

Q2: Construct pushdown automata that accept the following languages. You should give your answers in the form of diagrams, and each transition should only push a single symbol to the stack, i.e. you should not use the shorthand used in the proof of Theorem 5.6.

(a) $\{w \in \{a,b\}^* \mid w = w^R\}$, where w^R denotes the reverse of w.



(b) $\{w \in \{a,b\}^* \mid w$ contains more $'a'$s than $'b'$s $\}$



`Q3: Prove that the following languages are not context-free:`

(a) $\{0^k10^k10^k1 \in \{0,1\}^* \mid k \geq 0\}$

Step 1:
Suppose that $L = \{0^k10^k10^k1 \in \{0,1\}^* \mid k \geq 0\}$ is context-free. By the pumping lemma for context-free languages, $\exists$ some $p > 0$ (pumping length) s.t. $\forall$ s with $|s| \geq p$ can be pumped.

Step 2:
Choose $s = 0^p10^p10^p1$. Here $|s| = 3p + 3$. We want $3p + 3 \geq p$, which is true as $p > 0$. Therefore, our choice of s is $0^p10^p10^p1$.

Step 3:
By the pumping lemma, $\exists$ some division $s = uvxyz$ s.t.
1. $uv^ixy^iz \in L, \forall i \geq 0$
2. $|vy| > 0$
3. $|vxy| \leq p$

Step 4:

(1) When both v and y contain only one type of symbol

i. when both v and y contain only 0
Then the string $s = uv^2xy^2z$ can be of the form $0^q10^p10^p1$, $0^q10^r10^p1$, $0^p10^q10^p1$, $0^p10^q10^r1$ or $0^p10^p10^q1$, where $q, r > p$. Hence $s \notin L$

ii. when both v and y contain only 1. This case is not possible as $|vxy|$ has to be less than p

(2) When either v or y contains only 1
Then the string $s = uv^2xy^2z$ will contain extra 1s. Therefore $s \notin L$

(3) When either v or y contains both 0 and 1
Then the string $s = uv^2xy^2z$ will contain extra 1s. Therefore $s \notin L$

Contradiction! Therefore this given language is not context-free.

(b) $\{u\#v \mid u, v \in \{0,1\}^* \text{ and } u \text{ is a substring of } v\}$

Step 1:
Suppose that $L = \{u\#v \mid u, v \in \{0,1\}^* \text{ and } u \text{ is a substring of } v\}$ is context-free. By the pumping lemma for context-free languages, $\exists$ some $p > 0$ (pumping length) s.t. $\forall$ s with $|s| \geq p$ can be pumped.

Step 2:
Choose $s = 0^p1^p\#0^p1^p$. Here $|s| = 4p + 1$. We want $4p + 1 \geq p$, which is true as $p > 0$. Therefore, our choice of s is $s = 0^p1^p\#0^p1^p$.

Step 3:
By the pumping lemma, $\exists$ some division $s = uvxyz$ s.t.
1. $uv^ixy^iz \in L, \forall i \geq 0$
2. $|vy| > 0$
3. $|vxy| \leq p$

Step 4:

(1) When both v and y contain only one type of symbol

 i. when both v and y contain only 0
 Then the string $s = uv^2xy^2z$ can be of the form $0^q1^p\#0^p1^p$, where $q > p$. Here $u = 0^q1^p$ and $v = 0^p1^p$. u is not a substring of v. Hence $s \notin L$

 ii. when both v and y contain only 1.
 Then the string $s = uv^2xy^2z$ can be of the form $0^p1^q\#0^p1^p$, where $q > p$. Here $u = 0^p1^q$ and $v = 0^p1^p$. u is not a substring of v. Hence $s \notin L$

(2) When either v or y contains only 1
Then the string $s = uv^2xy^2z$ can be of the form $0^p1^q\#0^p1^p$, where $q > p$. Here $u = 0^p1^q$ and $v = 0^p1^p$. u is not a substring of v. Hence $s \notin L$

(3) When either v or y contains both 0 and 1
Then the string $s = uv^2xy^2z$ can be of the form $0^m0^i1^j0^i1^j1^n\#0^p1^p$. Here $u = 0^m0^i1^j0^i1^j1^n$ and $v = 0^p1^p$, where $m + i > p$ and $j + n > p$. Here 0s and 1s are not in the correct order. u is not a substring of v. Hence $s \notin L$

Contradiction! Therefore this given language is not context-free.

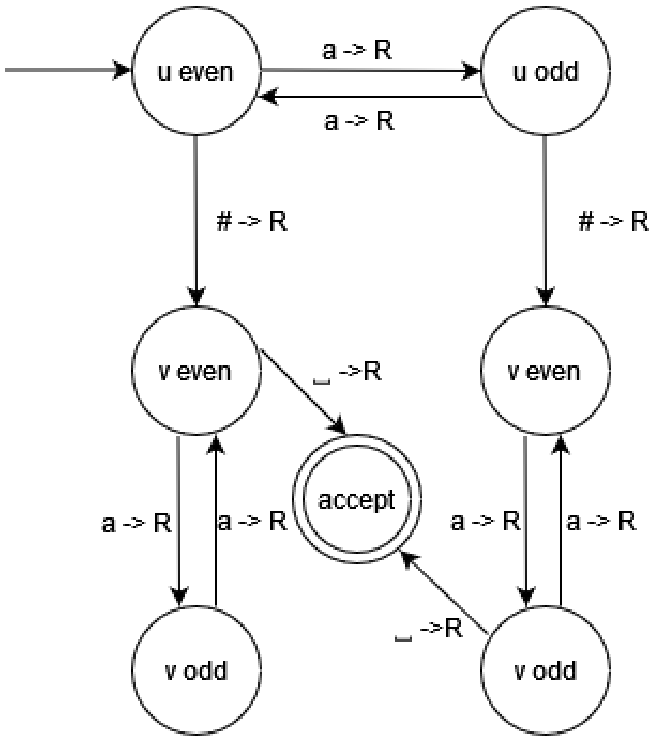
`Q4: Given a language` L`, let` L^R `denote the reversal of the language, that is,` $L^R = \{w^R \mid w \in L\}$`. Is the class of context-free languages closed under reversal? If so, give a construction which shows this. If not, give a counterexample.`

In order to prove the class of context-free languages is closed under reversal, we need to show that for any language L, if L is context-free, then L^R is also context-free.

According to Theorem 2.9 in Sipser book: Any context-free language is generated by a context-free grammar in Chomsky normal form (CNF). If L is a context-free language, then there is a grammar in CNF $G(V, \Sigma, R, S)$ that generates L. Here V is a finite set called variables, Σ is a finite set, disjoint from V, called the terminals, R is a finite set of rules, S is the start variable. Since this grammar G is in CNF, every rule is of the form $A \to BC$ or $A \to a$, where a is any terminal and A, B, C are any variables. B and C may not be the start variable. In addition, we permit the rule $S \to \epsilon$ where S is the start variable.

For every rule of the from $A \to BC$ in R, we can replace it with $A \to CB$, and put this updated rule in R'. For every rule of the from $A \to a$ in R, we leave it the same and put it in R'. Then the language L' generated by $G'(V, \Sigma, R', S)$ will be exactly the reverse of the language L generated by G.

Q5: Consider the language $L = \{u\#v \mid u, v \in a^*,\ |u| \text{ and } |v| \text{ have the same parity}\}$. Parity means odd or even, so the language only includes strings where both parts are odd or both parts are even. Draw a diagram version of a Turing machine that decides the language L.



Q6: A Caesar cipher is one of the simplest and earliest known tools for encrypting text, i.e., hiding it from plain sight with a reversable technique.
To encode a string, you replace each letter with a letter shifted by some fixed number of positions later in the alphabet - wrapping around if necessary. So, for example, with a shift of 2 over the regular English alphabet $\{a, b, c, ..., z\}$, a is replaced with c , b is replaced by d , and so on, with z being replaced by b . Decoding the string is simply a shift in the reverse direction.

Now, consider the language:
$L = \{u\#v \mid u, v \in \{a, b, c, d\}^*,\ v$ is equal to u with a Caesar cipher shift of 2$\}$
So $adab\#cbcd \in L$, but $adab\#adab < L$.

Write a description of a Turing machine algorithm that decides L . You should not give a Turing machine diagram for this question.

Here we describe a Turing Machine M that decides $L = \{u\#v \mid u, v \in \{a, b, c, d\}^*,\ v$ is equal to u with a Caesar cipher shift of 2$\}$.

$M =$ "On input string w:

1. If the letter is #, move right; if the letter is being crossed off (marked as x), keep moving right until it encounters the end symbol, then accept;
   If the letter is not #, mark this letter as x, and move right.
2. Keep moving right until encountering #
3. If the letter is marked as x, keep moving right
4. If the first letter which is not x on the right of # is not the same letter as the letter that was previously marked as x with a Caesar cipher shift of 2, reject.
   If it's the same as the letter with a Caesar cipher shift of 2, cross off the letter, and move left
5. Keep moving left until encounter #
6. If the letter is among the a, b, c, d, keep moving left. Until the letter is x, move right. Then repeat from stage 1

For every character of a, b, c, d, have a branch - a set of states and transitions - which replaces the specific character with an x; then moves right on the tape until reaching the # char; then moves right over x; then on the first char that's not crossed off (marked as x) or blank, check if it is the correct Carsar cipher character the branch is for: if not reject; go left over xs until a #; then go left until an x; The branch is done and we go back to the initial state (and move one right to be on the next character that's not x)

- $Q = \{q_1, ..., q_{accept}, q_{reject}\}$
- $\Sigma = \{a, b, c, d, \#\}$
- $\Gamma = \{a, b, c, d, \#, x, \sqcup\}$
- the start, accept, reject states are q_1, q_{accept}, q_{reject} respectively

`Q7: Note: you must have correct solutions for Questions 5 and 6 before attempting this question.`

1. Give the big O notation complexity for the Turing machine you drew in Question 5
   O(the Turing Machine in Q5) $= n$, here n is the length of the string.
   It goes over each character once without going back (left) on the tape.
2. Give the big O notation complexity for the Turing machine you described in Question 6.
   O(the Turing Machine in Q6) $= (1 + n + 1 + n) * n = (2 * n + 2) * n = 2 * n^2 + 2 * n = O(n^2)$
   Here n is the length of the string before #. The length of the whole string is $2 * n + 1$
   It has a loop that goes scans right and left for each character going over about half the whole string each time.

`Q8: Prove that the class P of languages with known polynomial time algorithms is closed under the operations of union and concatenation.`

To prove that the class P of languages with polynomial time is closed under union and concatenation, we want to show that for languages $L1$ and $L2 \in P$, $L1 \cup L2 \in P$ and $L1L2 \in P$.

1. Union
   Define $M1$ and $M2$ to be the Turing Machines that decide $L1$ and $L2$ respectively. By definition they run in polynomial time. Define a new Turing Machine M that uses $M1$ and $M2$ to decide the union of $L1$ and $L2$.
   $M =$ "On input w,

(a) For any string $w \in L1 \cup L2$, we run $M1$, $M2$.

(b) If either accepts, then accept.

The complexity of M is $O(M) = O(M1) + O(M2)$, which is the sum of two polynomial times, which is also polynomial.

2. Concatenation
Define $M1$ and $M2$ to be the Turing Machines that decide $L1$ and $L2$ respectively. By definition they run in polynomial time. Define a new Turing Machine M' that uses $M1$ and $M2$ to decide the concatenation of $L1$ and $L2$.
M' = "On input w,

(1) For every possible cut (every index), divide w into two substrings $w = w_1 w_2$

(2) Run $M1$ and $M2$ on the divisions ($M1$ on w_1 and $M2$ on w_2); if both $M1$ and $M2$ accept, then accept;

(3) if no cut is accepted, reject; "

The decision per cut is polynomial, and there are at most n cuts to be checked, so M' is polynomial