HW3

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

Give, without proof, a CFG that generates each of the following languages

1. $L_1 = \{w \in \{0,1\}^* | \ w \text{ is a palindrome}\}$
2. $L_2 = \{w \in \{0,1\}^* | \text{ in every prefix of } w \text{ the number of 0's is at least the number of 1's}\}$

Solution:

There are more than 1 possible CFG for each of these languages, but here are two that work:

1. Ans: $S \to 0S0|1S1|1|0|\epsilon$

2. Ans: $S \to 0S|0S1S|\epsilon$

Problem 2

Provide a deterministic push-down automata that recognizes the following language L over the alphabet $\Sigma = \{0,1\}$:

$$L = \{w| \ w \text{ has exactly half as many 0's as 1's.}\}$$

Solution :

The main idea behind this solution is every time we see a 0, we will push two 0's to the stack, and every time we see a 1, we will push one 1 to the stack. But, if we see a 0, and there's a 1 on the stack, then we pop two 1's instead. And if we see a 1, and there's a 0 on the stack, then we pop one 0 instead.

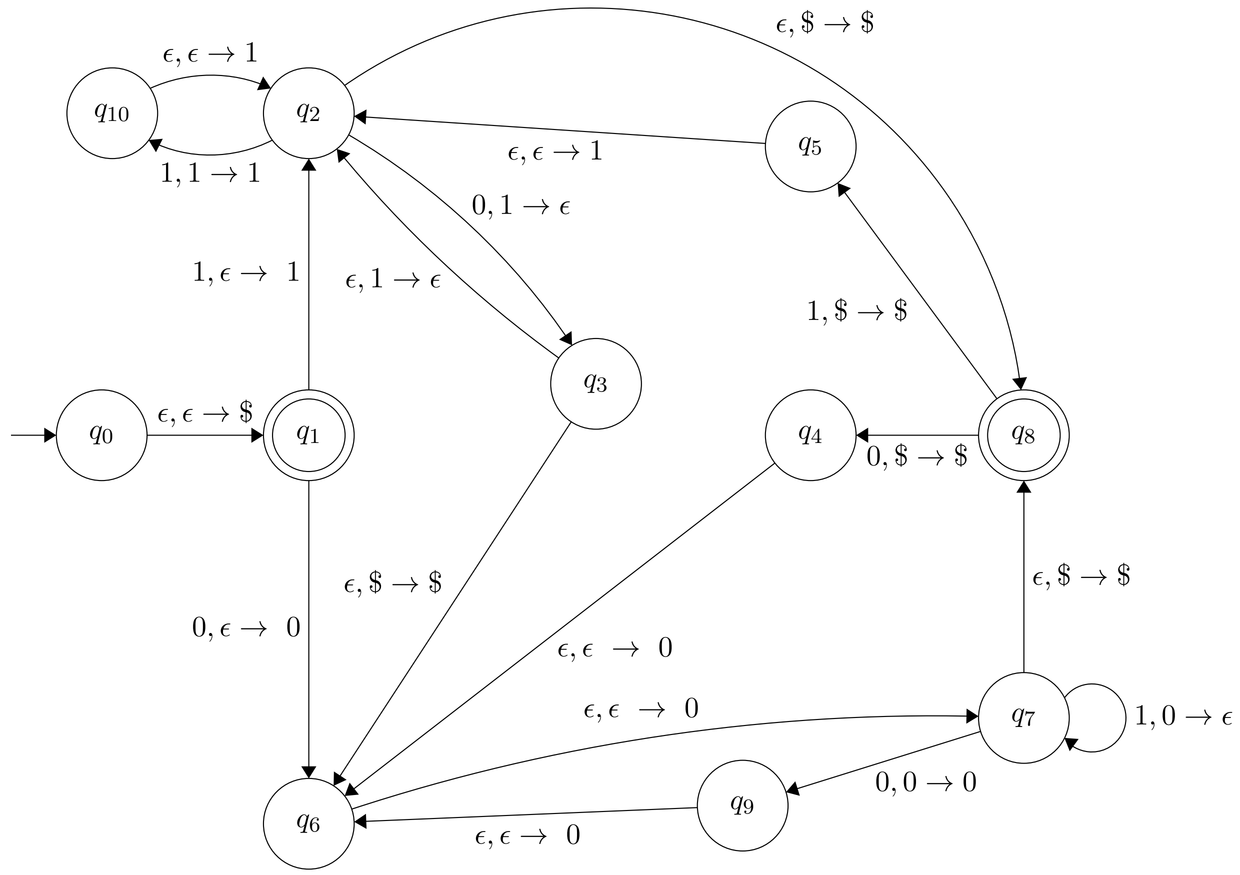
The 0's and 1's cancel each other out in this way, but seeing a 0 is "worth" twice as much as seeing a 1. So, we only accept if two times the number of 0's equals the number of 1's, i.e. the 0's and 1's all canceled each other out and the stack is empty at the end.

There's some additional details to consider, like what do we do for the string 101, e.g. when there's only one 1 on the stack and we see a zero, but that's the high-level idea.

When constructing our PDA, we will construct it so that our stack only ever contains all 0's or all 1's at one time. This allows us to cancel out the 0's and 1's without resorting to nondeterminism.

We will also need to make some intermediate states in order to push/pop more than one symbol to the stack, which is why it looks like it's more complex than it needs to be.

Consider the following deterministic PDA M:


$\epsilon, \$ \rightarrow \$$
$\epsilon, \epsilon \rightarrow 1$
q_{10}
q_2
$1, 1 \rightarrow 1$
q_5
$\epsilon, \epsilon \rightarrow 1$
$0, 1 \rightarrow \epsilon$
$1, \epsilon \rightarrow 1$
$\epsilon, 1 \rightarrow \epsilon$
$1, \$ \rightarrow \$$
q_3
q_0
$\epsilon, \epsilon \rightarrow \$$
q_1
q_4
q_8
$0, \$ \rightarrow \$$
$\epsilon, \$ \rightarrow \$$
$\epsilon, \$ \rightarrow \$$
$0, \epsilon \rightarrow 0$
$\epsilon, \epsilon \rightarrow 0$
$\epsilon, \epsilon \rightarrow 0$
q_7
$1, 0 \rightarrow \epsilon$
$0, 0 \rightarrow 0$
q_9
q_6
$\epsilon, \epsilon \rightarrow 0$

All undrawn transitions are assumed to go to the sink state (it'll turn out that no input string halts in the sink state by the way that we control the stack, but we must include it anyways).

We will show that $L(M) = L$.

Let $W_w(0) = 2 * \# \text{ 0's in } w$ and $W_w(1) = \# \text{ 1's in } w$, i.e. the 0's are worth twice as much. First, we will show that we have the following invariants for any w (and thus any prefix of w also):

- If $W_w(0) > W_w(1)$: The stack contains a \$ followed by only 0's and the number of 0's is equal to $W_w(0) - W_w(1)$. Also, the PDA halts in state q_7.
- If $W_w(1) > W_w(0)$: The stack contains a \$ followed by only 1's and

the number of 1's is equal to $W_w(1) - W_w(0)$. Also, the PDA halts in state q_2.

- If $W_w(0) = W_w(1)$: The stack contains only a \$ and the PDA halts in either states q_1 or q_8.

Note that states $q_3, q_4, q_5, q_6, q_9, q_{10}$ serve as transition states, which we will use to push/pop two symbols to/from the stack. Note that it is sufficient to prove these invariants, because the PDA ends in an acceptance state if and only if $W(0) = W(1)$ at the end of the input so the number of 1's is two times the number of 0's. We will proceed by induction on the length k of the input string w:

- Base case: $k = 0$ so $w = \epsilon$. In this case, we start in q_0, then we take the transition to q_1. We have that $W(0) = W(1) = 0$. So, the PDA accepts, as desired.

- Inductive step: $k \geq 1$. Suppose we have a string w over the alphabet $\{0, 1\}$ of length k that makes M terminate in the state q_i with the appropriate stack. We will show that concatenating w with any symbol $s \in \{0, 1\}$ to make a new string w' also makes M terminate in an appropriate state with the appropriate stack. We have to consider the following cases based on w:

  1. $W_w(0) > W_w(1)$. In this case, by our hypothesis, we are in state q_7. If $s = 0$, then we transition to state q_9 and then to q_6 and then back to q_7, pushing two 0's to the stack. Previously, by the inductive hypothesis, there are $W_w(0) - W_w(1)$ 0's in the stack. Now, there are $W_w(0) - W_w(1) + 2$ 0's in the stack. Note that w' has one more 0 than w so $W_w(0) - W_w(1) + 2 = W_{w'}(0) - W_{w'}(1)$. Thus, there are $W_{w'}(0) - W_{w'}(1)$ 0's in the stack. If this quantity is greater than zero, we do nothing. If this quantity is equal to zero, then the stack is just a \$, so we transition to q_8, accepting. Therefore, M terminates in the appropriate state with the appropriate stack.

     If $s = 1$, then we would pop a 0 from the top of the stack. Then, we have two cases. If the top of the stack is a \$, then we transition to q_8 in which case $W_{w'}(1) - W_{w'}(0) = 1$. If the top of the stack is a 0, then we stay in q_7 in which case $W_{w'}(0) - W_{w'}(1) =$

$W_w(0) - W_w(1) - 1$. Therefore, M terminates in the appropriate state with the appropriate stack.

2. $W_w(1) > W_w(0)$. This argument follows similarly to that of the first case.
3. $W_w(0) = W_w(1)$. This argument follows similarly to that of the first case.

Problem 3

For any language A, let $PREFIX(A) = \{v | vu \in A$ for some non-empty string $u\}$, i.e. the set of proper prefixes of strings in A. Show that the class of context-free languages is closed under the $PREFIX$ operator.

Ans: Let A be a context free language and G be the context free grammar for it. Assume G is in Chomsky normal form for convenience.

Note that there is an edge case in which $A = \{\epsilon\}$, in which case $PREFIX(A) = \emptyset$, but this is trivially context-free. So we'll assume this isn't the case moving forward.

In order to prove that A is closed under the PREFIX operation, we will create a grammar G' for PREFIX(A) thus proving that it is context free as well.

1. For every rule $X \to YZ \in G$, add to G' the following rules:
   - $X^* \to Y^* Z^*$
   - $X' \to Y^* Z'$
   - $X' \to Y'$
   - And if X is the starting variable, we also add $X \to Y^* Z'$ and $X \to \epsilon$.
2. For every rule $X \to c$ (where c is a terminal symbol) in G, add to G' the rule $X^* \to c$.
3. For every variable X in G, add the rule $X' \to \epsilon$ to G'.

To prove the correctness of this grammar, we would argue first that our initial rule must get us a smaller string and then from there G' can generate any proper prefix.

Instead, here we will prove closure under $PREFIX$ using a proof by construction with a PDA as follows:

Note: the following explanation is much, much longer than needed, but we wanted to provide a proof for why each detail of the PDA works in case it clears up confusion

We will proceed with a proof by construction to show that if L is a context-free language, then PREFIX(L) is also a context-free language. Note that doing so also shows that the class of context-free languages is closed under the PREFIX operation. Let L be a context free-language over the alphabet Σ. Since a language is context-free if and only if some PDA recognizes it, there must exist a PDA that recognizes L. Denote any such PDA that recognizes the context-free language L as $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$. We will construct a PDA P' using P to recognize PREFIX(L) as follows. Let $n := |Q|$.

First, we create one copy of P denoted by P_0. Then, we create a modified copy of P denoted by P_1. This modified copy P_1 has the states $Q \times \{a, b\}$. What this means is that for each state $q_i \in Q$, we have two corresponding states in P_1: (q_i, a) and (q_i, b). We will refer to a given state in P_1 as an "a" state if it is of the form (q_i, a) and a "b" state if it is of the form (q_i, b). The idea behind this is so that the execution of P' first consumes all the symbols of an input string, and then the "b" states represent guessing the execution of non-empty suffixes; but, we still need the "a" states because otherwise the ϵ transitions of P may result in an empty suffix (we will formally look at this later on). So, the states of P' are $Q \cup (Q \times \{a, b\})$. Also, P' has the same alphabet Σ and stack alphabet Γ as P.

Then, we describe the transitions of P' as follows. First, we have that P_0 retains the original transitions of P (and P_1 does not retain the original transitions). Then, for each state q_i of P_0, we add the transition $\epsilon, \epsilon \to \epsilon$ from q_i in P_0 to (q_i, a) in P_1. Finally, for each transition from a state q_i to a state q_j in P, we add a transition in P_1 based on two cases:

1. If the transition is of the form $\epsilon, u \to v$ for $u, v \in (\Gamma \cup \epsilon)$, then we add a transition of the form $\epsilon, u \to v$ in P_1 from (q_i, a) to (q_j, a) and from (q_i, b) to (q_j, b).

2. If the transition is of the form $s, u \to v$ for $s \in \Sigma$ and $u, v \in (\Gamma \cup \epsilon)$, then we add a transition of the form $\epsilon, u \to v$ in P_1 from (q_i, a) to (q_j, b) and from (q_i, b) to (q_j, b).

The starting state of P' is the starting state q_0 of P. The accepting states of P' are precisely the "b" states in P_1 whose corresponding state in P was accepting (note that none of the states in P_0 are accepting states).

Now, we will show that P' recognizes PREFIX(L). To do so, we will show that given any input string v, P' first simulates P's executions of v (in P_0) and then in a separate process, attempts to guess the executions of strings u in P (in P_1) such that $vu \in L$. We will show that this is equivalent to the original statement by also showing that P' only accepts such executions in which u is non-empty (thereby only accepting v if it is a proper prefix of at least one string in L).

First note that since there are no transitions going from a state in P_1 to a state in P_0 and since P_1 only contains ϵ transitions (with respect to the symbol in the input string), any execution of P' that uses a transition from P_0 to P_1 without completely consuming all the symbols in v beforehand will either continuously loop forever in P_1 without ever being able finish consuming the symbols in v (which is a condition required to halt) or go to the dump state. Therefore, when P' receives v as an input, the only executions that can possibly end in an accepting state of P' are the ones in which all the symbols of v are consumed by P_0 (as opposed to P_1). Moreover, since none of the states in P_0 are accepting, the only executions that can possibly halt in an accepting state of P' are the ones in which all the symbols of v are consumed by P_0 and which use a transition from P_0 to P_1. Along with the fact that P_0 retains the transitions of P, we have that P' first completely handles P's executions of v in P_0 and then in a distinctly isolated process handles u.

A given execution of P' will first fully consume the input string v and then since the transitions from P_0 to P_1 preserve the same corresponding state in P and also do not modify the stack (since these transitions are of the form $\epsilon, \epsilon \to \epsilon$), P' will be in the same state with the same stack as if P received v, once transitioning to P_1 from P_0. Now, we will show that the distinctly isolated process that handles guesses of u in P_1 makes P' accept the input string v if and only if $vu \in L$ and u is non-empty. First, we have that P_1's transitions with respect to the states and stack of P are identical with the only change being that transitions that consume a non-empty symbol from the input string are changed to instead take ϵ. So, since any execution in P_1 starts in the same state as it would have when P receives v as an input, we have that P_1 can nondeterministically simulate any number of subsequent input symbols in the same way as P strictly after P receives input v. Moreover, for a given execution to be in P_1, all the symbols of v must have been consumed, so P_1 and thus P' can halt at any time. Note that an input string is merely a finite sequence of input symbols, so P_1 can nondeterministically simulate the execution of any string $u \in \Sigma^*$ in the same way as P after P receives input vu. So, when receiving input v, P' can simulate the execution of any string vu in the same way as P after P receives input vu.

Now, it remains to show that in order for P' to accept v, the guess u must be non-empty. We note that all transitions from P_0 to P_1 go to "a" states in P_1, none of which are accepting. Then, all of the transitions in P that were ϵ transitions with respect to the input symbol make "a" states transition to "a" states. So, prior to taking a transition in P_1 that represents consuming an input symbol, implying that the guess u is non-empty, it is guaranteed that no possible execution of P' accepts v. Moreover, if P_1 is in an "a" state and takes a transition that was a non-empty transition in P with respect to the input symbol, P_1 must go to an "b" state by construction. And if P_1 is in an "b" state, it can only transition to "b" states. So, a given execution in P' can only possibly be accepted if it takes a transition in P_1 that represents a non-empty transition in P, since none of the "a" states can possibly be accepting. Then, we have that P' halts in an accepting state if and only if there is an execution in which the guess u causes P_1 to transition from its "starting" state (the state directly after transitioning from P_0) to a "b" state in P_1 that corresponds to an accepting state of P. And since P_1 can only be in a "b" state if the guess u is non-empty, we have that when given input v, P' simulates P's possible executions of strings of the form vu, for non-empty u, and accepts v if only if P accepts vu. So, noting that P_0 first fully simulates the execution of v in P and P_1 thereafter simulates the remaining execution of a guess u in P, we have that P' accepts an input string v if and only if vu is accepted by P, for some non-empty string u. This implies that P' accepts an input string v if and only if $vu \in L$ for some non-empty string u, by choice of P, completing the proof.