CSCI 1010 - Fall'26 Theory of Computation Lorenzo De Stefani

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. $L_1 = \{w \in \{0,1\}^* | w = w^R \text{ and } |w| \text{ is even}\}$ (w^R denotes the reverse of w)
2. $L_2 = \{w \in \{0,1\}^* |$ in every prefix of w the number of 0's is no more than 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: $S \to 0S0|1S1|\epsilon$

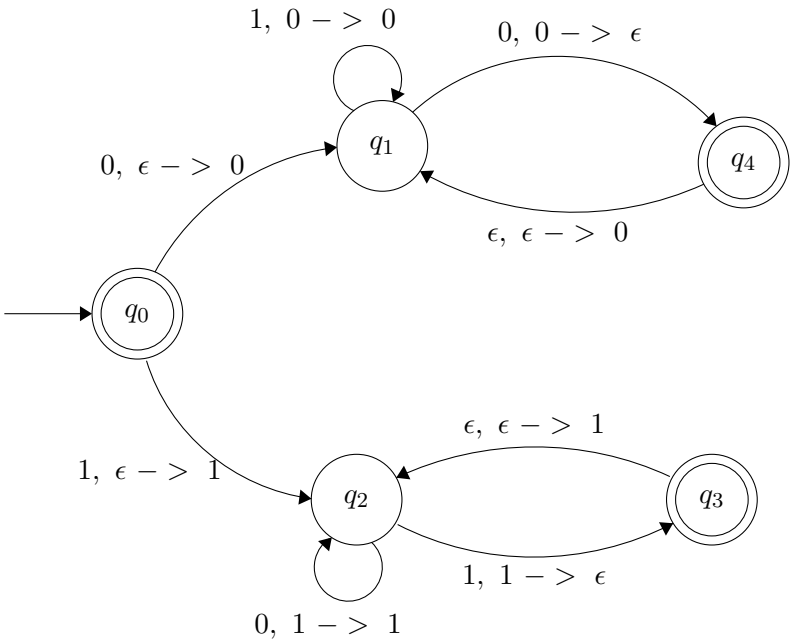
2. Ans: $S \to 1S|1S0S|\epsilon$

Problem 2

Design a *deterministic* push-down automaton for the language $L = \{w \in \{a,b\}^* |$w starts and ends with the same symbol$\}$ and provide a proof of correctness. (note: the empty string should be accepted)

Solution:

A potential DPDA is provided below:



Proof of Correctness:

To prove correctness we must prove that $L(P) = L$, where L(P) is the language recognized by our DPDA P.

First, P accepts all strings in L. Since we see the cases of starting with 0 and 1 are analogous in our DPDA, we will just prove for the case of strings that start and end with a 0 (and then the same will hold true for strings that start and end with a 1). That said, let's analyze what happens for a string that starts and ends with 0 in the DPDA above. Upon reading the first 0, the DPDA transitions from q_0 to q_1 and pushes the 0 onto the stack. For any 1 that is read at q_1, that single 0 gets pushed back onto the stack and the DPDA stays at q_1. For any 0 that is read that isn't the last zero, the DPDA transitions to q_4, making sure to push the single 0 back onto the stack. The DPDA then takes an ϵ transition back to q_1, leaving two branches (one at q_4 that will get terminated on the next input and one at q_1). This means there will always exist a branch at q_1 after the initial 0 is read, meaning once the final 0 is read, the DPDA will transition to the accepting state at q_4 and accept the string. We also note that since the start state is accepting, the empty string will be accepted as well.

Next, P will reject all strings not in L. Once again, since the cases of starting with 0 and 1 are analoggous in our DPDA, we will just prove for the case of

strings that start with 0 and end with 1 (and then the same will hold true for strings that start with 1 and end with 0). Here, the DPDA will transition to q_1 again after the first input symbol. From this point, any 1 that is read will keep the DPDA at the rejecting state of q_1. If a 0 is read, then, as argued above, the DPDA will have branches at both q_1 and q_4. Since we know a 0 is not the last symbol to be read, the branch at q_4 will terminate and thus the DPDA will have only 1 branch at state q_1 after reading through the entire input and, as such, reject the input.

Problem 3

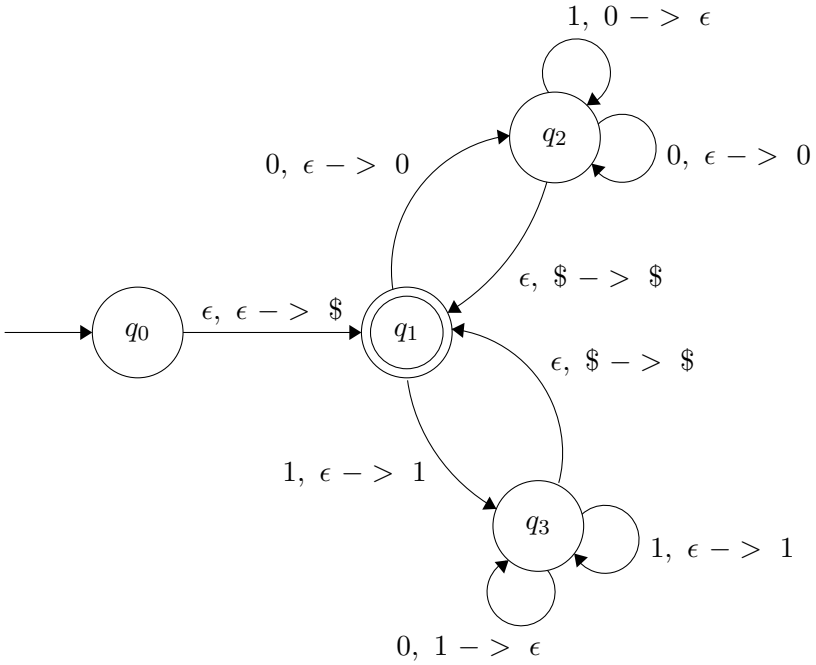
For an alphabet $\Sigma = \{a, b\}$, show that the language consisting of strings that have an equal number of a's and b's but no substring of the form abba is context free. (Hint: you may assume that the language of all strings with no substring abba is regular).

Solution:

For this, the general outline will be to break down the given language into the intersection of two languages: the language of strings with an equal number of a's and b's and the language with no substring of the form abba (which we are told we can assume is regular). From this, we will prove the language of an equal number of a's and b's is deterministically context free and then show that the intersection of a deterministic context free language (DCFL) and a regular language is still context free.

First, prove that the language of an even number of as and bs is deterministically context free:

For this, we provide a DPDA and a proof of correctness, but a context free grammar can also be provided.



First, this PDA accepts all strings in the language of an equal number of 0s and 1s. In the PDA, the dollar sign is the first symbol pushed onto the stack off of the ϵ transition to q_1. Since this is an ϵ transition and q_1 is accepting, the empty string will be accepted. From here, the case of reading a 1 and a 0 is analagous, so we will only focus on the case of a 0 being read first here. If a 0 is read, we transition to q_2 and push a 0 to the stack. For every 0 read at q_2, we push a 0 to the stack. For every 1 read, we pop a 0 from the stack. If the dollar sign is ever the symbol at the top of the stack, we ϵ transition back to q_1. Since the existing transitions either add a 0 for every 0 read or pop a 0 for every 1 read and the dollar sign is the first symbol on the stack, we will see the dollar sign if the number of 0s and 1s read from arriving at q_1 are equal and transition back to q_1 in this case. Since strings in the language have an equal number of 0s and 1s and, for this to be the case, the string must consist of some sequence of substrings (including the case of this only being the entire string) with equal 0s and 1s that will take us back to q_1 after each substring is read, we will end up in state q_1 and accept these strings.

Next, any string that doesn't have an equal number of 0s and 1s will be rejected by this PDA. First, consider a string that, when read from the start, has no position at which the number of 1s and 0s read up to that

point are equal (e.g. 001). Once again utilizing the fact the case of reading 0 or 1 first is analagous, consider the case where 0 is read first (meaning the number of 0s read at any given point in this subcase will be greater than the number of 1s). Then, for every 0 read, a 0 will be pushed onto the stack. Not enough 1s will be read to pop all these 0s from the stack, meaning we never see the dollar sign and transition back to q_1. We then stay in state q_2 and reject. In the previous paragraph, we proved that any string with an equal number of 0s and 1s will end at q_1 and accept. Any string that has an unequal number of zeros that doesn't follow the first subcase considered must then have a final point in the string where the number of 0s and 1s read up to that point are equal and that, from that point on, there will be no position at which the number of 1s and 0s read up to that point are equal. Then, at that final point, the PDA will have 2 branches, the one we proved at q_1 in the previous paragraph and the state that it came from (q_2 or q_3). Since all other transitions with the dollar sign at the top of the stack from q_2 and q_3 result in termination and rejection, those branches will terminate from reading the rest of the input. For the other branch, we proved that this will then end at state q_2 or q_3, meaning this branch will be rejected as well. Therefore, any string not in the language is rejected by our PDA.

Now, we've shown the language of an equal number of a's and b's to be context free and we've been told we can assume that the language with no abba substring is regular. We can then move to proving that the intersection of a DCFL and a regular language is a CFL:

Let L_1 be a deterministic context-free language. It has associated PDA $M_1 = \{Q_1, \Sigma, T, \delta_1, q_0, F_1\}$.

Let L_2 be a regular language. It has associated DFA $M_2 = \{Q_2, \Sigma, \delta_2, q_0', F_2\}$.

To prove that the intersection of L_1 and L_2 is a deterministic CFL, we will construct a deterministic PDA that represents the intersection of L_1 and L_2. The bottom line of this construction is that we execute M_1 and M_2 in parallel where each character in the string makes transition happens only if it is in the transition function of both $PDAs$, and only using the stack of M_1 since transitions of M_2 don't use the stack or if there is an empty string transition in M_1.

Define $M = \{Q_1 \times Q_2, \Sigma, T, \delta, (q_0, q_0'), F_1 \times F_2\}$ where δ is defined in the following way:

For any $(q_{11}, a, x) \to (q_{12}, y)$ in δ_1 and for any $(q_{21}, a) \to q_{22}$ in δ_2, we define

$((q_{11}, q_{21}), a, x) \to ((q_{12}, q_{22}), y)$ in δ. Moreover, for any $(q_{11}, \epsilon, x) \to (q_{12}, y)$ and for an $q_{21} \in Q_2$, we define $((q_{11}, q_{21}), \epsilon, x) \to ((q_{12}, q_{21}), y)$.

Notice that in this definition, since M_1 is deterministic and so is M_2, each transition maps a state not to a set of possible states, but only to one possible state,pair of the two only possible states in M_1 and M_2. Therefore, M is a deterministic PDA.

Now, we must prove correctness, meaning that $L(M) = L_1 \cap L_2$.

First we will prove that any string $w \in L_1 \cap L_2$ is accepted by M. Since w is in $L_1 \cap L_2$, it is accepted by both M_1 and M_2. The start state of M is the pair of start states in M_1 and M_2. When the M reads the first character of w, since the transition from the start state to the next state exists in both M_1 and M_2, it exists in M. Similarly, given any pair of states in M, the next character that will be consumed in w will yield a transition in M_1 and M_2 and therefore in M. If no character is consumed and there is an epsilon transition in M_1, this will also correspond to a transition in M. Finally, once w is consumed in both M_1 and M_2 and lands in accept states in $f_1 \in F_1$ and $f_2 \in F_2$, M is in the state (f_1, f_2) which is in $F_1 \times F_2$ the accept states of M. Therefore, w is accepted by M so $w \in L(M)$.

On the other hand, we will prove that any string $w \notin L_1 \cap L_2$ is not accepted by M. Since w is not in $L_1 \cap L_2$, it must not be in L_1 or not in L_2 (not an exclusive or). If it is not in L_1, then it ends in a state $q_n \in Q_1$ that is not an accept state in M_1. By construction, the consumption of w in M would end in (q_n, s_m) where $s_m \in Q_2$. $(q_n, s_m) \notin F_1 \times F_2$ so M does not accept w. If it is not in L_2, then it ends in a state $q_n \in Q_2$ that is not an accept state in M_2. By construction, the consumption of w in M would end in (s_m, q_n) where $s_m \in Q_1$. $(s_m, q_n) \notin F_1 \times F_2$ so M does not accept w. Therefore, if $w \notin L_1 \cap L_2$, w is not in $L(M)$.

We can thus conclude that $L(M) = L_1 \cap L_2$.

We have constructed a deterministic PDA that recognises $L_1 \cap L_2$, so the intersection of a deterministic context-free language and a regular language is also a deterministic context-free language. Since we showed the language in the problem to be the intersection of a regular and context free language, it must also then be context free.

Problem 4

For any language A, let $PREFIX(A) = \{v | vu \in A \text{ for some non-empty string } u\}$, 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.

Solution:

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.