1. (20 pts). convert to a PDA

From the textbook we know "The PDA P will work by accepting its input w, if G generates that input, by determining whether there is a derivation for w. ...Each step of the derivation yields an intermediate string of variables and terminals." We will simply use the rules of G to determine if any particular sequence of valid substitutions of these intermediate strings as an input of either terminals or variables, we can derive at an identical match to the input word w.

We begin by epsilon transitioning to a state q_1 and writing the start symbol "$" to the stack. we then epsilon transition to qLOOP and write the variable R representing the start state to the stack. Our accept state qACCEPT can be reached by an epsilon move when popping the start symbol from the stack.

Imagine there is a pointer which points to the first character in the input word w. Since different variables map to different characters or substrings we can only compare characters and advance the pointer of w for characters to the left of the variables in the rules.

We will accomplish this by allowing P to compare any terminals on the lefthand side of all Variables in the intermediate string it is currently processing to the corresponding letters in the words belonging to L(G).

In order to accomplish this, we take each rule and process it character by character from right to left, writing each element to the stack. When we have processed the entire substring (rule) the machine begins epsilon transitions as it pops from the stack. if the element being popped is a terminal (and it matches a possible element at the same index in some word which the language accepts) it advances the position of the pointer in w. If the element being popped is a Variable, P repeats this step for all rules associated with this variable.

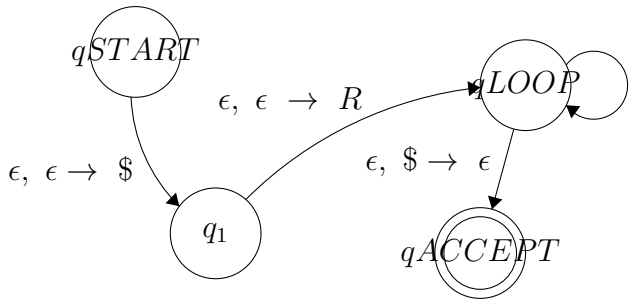
We construct one state in our machine which handles this functionality– qLOOP. For every Rule in G we construct a sequence of states which pushes in reverse order the elements of the substring of said rule. After the sequence of pushes, the machine returns to qLOOP. After we return to qLOOP, if there is a terminal on top of the stack, we determine if that terminal is a valid element in a word accepted by G. if it is not, we transition to the garbage state. In essence this is advancing the pointer of input w, if w is still eligible for acceptance. If the machine ever reads the "$" symbol in the stack then we know every character of the input string has been matched by some rule in G and that input is accepted by our PDA, which means it is an element of L(G). Therefore we have shown how to construct a PDA for G.

Below is a diagram of the above described PDA which is equivalent to G.

As there was not enough room to write all of the functions which transition from qloop to qloop, please insert the following:

$\epsilon, R \to XRX$
$\epsilon, R \to S$
$\epsilon, S \to aTb$

$\epsilon, S \to bTa$
$\epsilon, T \to XTX$
$\epsilon, T \to X$
$\epsilon, T \to \epsilon$
$\epsilon, X \to a$
$\epsilon, X \to b$
$a, a \to \epsilon$
$b, b \to \epsilon$



2. CONVERT A PDA INTO A GRAMMAR WHICH PRODUCES EACH AND ONLY EACH WORD THE PDA ACCEPTS.

We follow the algorithm described in Sipser's textbook to construct a CFG G.

we begin by modifying the PDA given so that the following rules apply.

1. There is only one accept state.

This is accomplished by using the identical PDA which was discussed in class with the only two differences being q1 no longer accepts but there is an epsilon transition between q2 and q3.

2. It empties its stack before accepting. (this is already the case)

3. each transition either pushes or pops– not both.(this is already the case, we are going to allow it to epilon move from q1 to q2 without doing either, I am not sure if this is legal or not)

Now we list our transitions of function δ

$q1(q2, \epsilon, push\$)$
$q2(q2, 0, pushx)$
$q2(garbageState, 1, NA)$
$q2(q3, \epsilon, \epsilon)$
$q3(q3, 1, popx)$
$q3(garbageState, 0, NA)$

$q3(q4, \epsilon, pop\$)$

now we let q4 represent the state q which is the state when the stack is empty.

We let q1 represent the state p which is the state when the stack is empty.

in this case state r and state s which arrive at and leave the in an identical manner are the same as states p and q because there is no way the stack can become empty and then read another element in and still reach the accept state. The PDA is counting the number of 0's and then it counts the number of ones. The only way a word is accepted is if the only time the stack empties is at the completion of reading the word. So then we know anytime we go from state p (or r) to state q (or s), if the stack is empty, then we will accept the word. Similarly every time we apply a rule which constructs this scenario from an identical scenario, we are constructing a word which is accepted by P.

We can let r correspond to p and s correspond to q in which case we would have a single rule for our grammar represented by the start Variable A_{rs}

Then our grammar looks like this:

$A_{rs} \rightarrow 0A_{rs}1|\epsilon$

3. CONSTRUCT A PDA FOR PART A AND CFG FOR PART B WHICH RESPECTIVELY ACCEPTS AND GENERATES WORDS IN LANGUAGE B

a.) to construct a PDA we need a start state qSTART which pushes the startSymbol onto the stack.

we need an Accept state qACCEPT which is entered by popping the StartSymbol from the stack.

we will use the stack to keep track of the number of characters of input word w which have been processed.

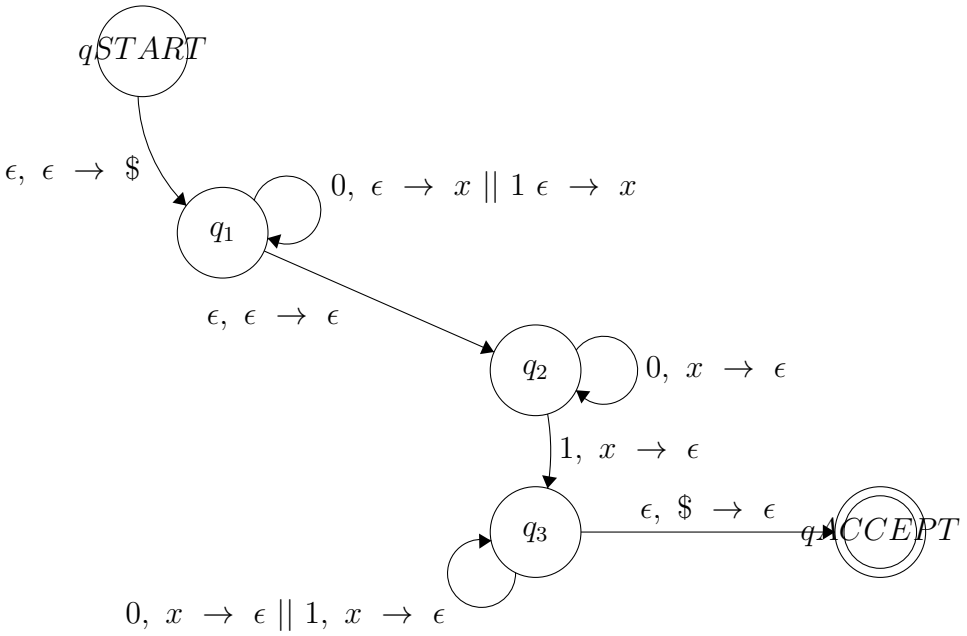
for every element processed we push an x onto the stack and nondeterministically guess that that the next element in w is the first element of the second half of the word. Therefore a new state q2 is entered. While in q2 we pop one x for every new input character we read– in doing this we will know if we guessed the halfway point in w correctly. If a 1 is read while in state q2 then we move to state q3. We pop an 'x' for every new character read in in the same was as state q2. If at any time in states q2 or q3 if we reach the startSymbol in the stack before we are finished reading in every character of w, then we know we guessed a point in w left of center and any findings of a 1 can be disregarded because we do not know if it in the second half of the string.

PDA P is guessing every character to be the first character in the right hand side of w such that if the right hand side will is the same size as the left or smaller and a 1 is found the

word inputted will be accepted . If there was a 1 in the right hand side of w when the input ends we will always see the startSymbol at that point in exactly one of our nondeterministic branches which has reached q3 and not been sent to the garbage state as a result of running out of 'x's on the stack before the word ended. If we did not see any one's in that particular branch that branch will finish in state q2 and the word will not be accepted.

we must take into account the possibility that the word is of length 1, in which case we should accept if it comprises "1". To account for this we will epsilon transition to state q2 from the q1 while pushing an 'x' on to the stack.

The PDA which accomplishes all of this can be seen in the figure below.



Part B.) Find the CFG which constructs words for language B

To find the CFG which constructs the same words which the above PDA recognizes, we begin by deciding which states will start and leave the stack in an identical manner in terms of if we push and pop acceptable characters which will construct legal words in Language B, then we turn these states into states r and s. We then develop rules which go from p to q which in our case is the start and accept state.

we can then let q1 be r and q 3 be s. we will find that any application of any rule which takes us from r to s can be applied any countable number of times, and we will still construct a word in language B . to implement a CFG in this way we add a rule for every occurance of any transition from q1 to q3 such that if the rule is applied any number of times it will be the same as if it was not applied at all and if it was applied multiple times.

In the case of this language we devise a rule which maps from some Variable A_{ps} to $0^*A_{rs}1^+$ One time and then define Variable A_{rs} to $0A_{rs}1||1a_{rs}0||1a_{rs}1||1a_{rs}1||\epsilon$ as many times as we

want. This yields a word which has at least one 1 on the right hand side. Unfortunately this is not correct because it only generates words which end in a 1 :(

Did NOt COMPLETE

4. WE WILL SHOW THAT FOR EVERY WORD OF L(G), THERE ARE AN EQUAL NUMBER OF A'S AND B'S

We will show this with an inductive proof on the size n of any word w in G. for each step in our proof we will look at every rule of grammar G which is applicable to the situation and determine that for every case, there is an equal number of a's and b's

BASE CASE: let n = 0

Then we know the word is 0 character long. In this case the start state will map directly to the output string (emptyString) there are 0 a's and 0 b's. So we have proved the base case.

Inductive Hypothesis (notice for every rule in the grammar exactly 0 or 2 terminals are added to the derived word, so then we do not pay attention to odd length words, because there are none):

——for all words of size (n = k):

Suppose all words of k length have the same number of a's and b's

then for each word of size k + 2 there are also the same number of a's and b's

we consider each case possible by looking at every rule of G

From the start state we transition to T endmarker. No terminals are added so we still have the same number of a's and b's

From T we have three derivation options.

1. $T \to TaTb$ in which case exactly one a and one b are added to the derived word plus 2 times all other possible mappings of T.

2. $T \to TbTa$ in which case exactly one a and one b are added to the derived word plus 2 times all other possible mappings of T.

3. $T \to \epsilon$ in which case exactly zero a's and b's are added to the derived word.

Since we have shown for every rule of G there is either an equal number of a and b terminals or an equal number of Variable T which recursively generate an equal number of a and b

terminals or another instance of itself, then for every word generated by G there will be an equal number of a's and b's.

Therefore, we have proved by induction on the length of w as required.