pgf@stop

THEORY OF COMPUTATION - CS340

2022-11-10

CS340 Assignment2

Distincts

Prof. Manindra Agarwal

Name: Jaya Gupta
Roll Number: 200471

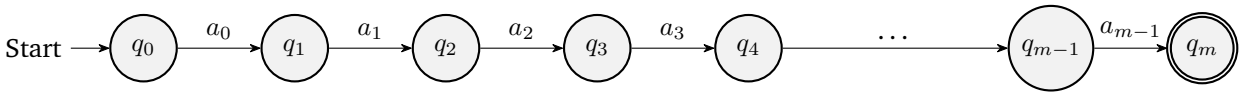
Name: Harshit Bansal
Roll Number: 200428

Name: Avi Kumar
Roll Number: 200229

Q1

1. **To prove :- Input x is accepted by F iff there exist an accepting configuration sequence that starts with** $< q_0, x >$**.** Let's prove it both sides.

   (a) If input x is accepted by $\mathcal{F}$ then, there exist an accepting configuration sequence that starts with $< q_0, x >$.
   Since x is accepted by $\mathcal{F}$, there exist a accepting transitions sequence for x in $\mathcal{F}$. Let it be:-



   where q_0 is starting state and q_m is final accepting state.
   Now let's give an accepting configuration sequence for x.

$$S =< q_0, y_0 >< q_1, y_1 >< q_2, y_3 > .... < q_m, y_m >$$

$$\text{here, } y_0 = x$$
$$x = z_i y_i \forall i \in [1, m]$$
$$a_i \in \sum \cup \{\epsilon\}$$
$$z_i = a_0 a_1 .. a_{i-1}$$
$$y_m \text{ is a empty string since } x = a_0 a_1 ... a_{m-1}$$

   Now, let's prove S is an accepting configuration sequence.

   i. Each $\langle q_i, y_i \rangle$ is a configuration
   By definition of S since, each q_i occur in transition so $q_i \in Q$
   Since, $y_0 = x$ and $x = z_i y_i$, y_i is suffix of x.

   ii. F moves from configuration $<q_i, y_i>$ to $<q_{i+1}, y_{i+1}>$ in one step
   This directly comes from definition of S. It takes one step by reading a_i at state q_i for moving configuration $<q_i, y_i>$ to $<q_{i+1}, y_{i+1}>$.

   iii. $q_m \in F$, and $y_0 = x$.
   This comes from definition of S.

   (b) If there is an accepting configuration sequence that starts with $< q_0, x >$ then x is accepted by $\mathcal{F}$.
   Assume an accepting configuration sequence be:-
   $S =< q_0, y_0 >< q_1, y_1 >< q_2, y_3 > .... < q_m, y_m >$ where,

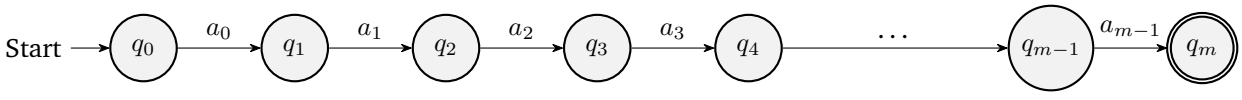
   i. Each $\langle q_i, y_i \rangle$ is a configuration

   ii. F moves from configuration $<q_i, y_i>$ to $<q_{i+1}, y_{i+1}>$ in one step

   iii. $q_m \in F$, and $y_0 = x$.

   iv. y_m is empty string.

   $a_i \in \sum \cup \{\epsilon\}$
   By ii) configuration $< q_i, y_i >$ is reachable to $<q_{i+1}, y_{i+1}>$ in one step.
   so there must be a transition by reading some character a_i (can be ϵ too) from configuration $< q_i, y_i >$ to $<q_{i+1}, y_{i+1}>$
   So, $y_i = a_i y_{i+1}$ and there is a edge form q_i to q_{i+1} by reading a_i
   Also since y_m is empty, by induction on m, $a_0 a_1 .. a_{m-1} = x$. One transition sequence for x on F will be :-



   Since there exist a sequence of transitions for x which ends in accepting state so x is accepting.

2. **We have to prove that the set of all accepting configuration sequences of** $\mathcal{F}$ **is computable.**
   We will give $\mathcal{F}$ and an input S to the Turing machine M.
   Let S be of the form: - $S =< q_0, x >< q_1, y_1 >< q_2, y_3 > .... < q_{m+1}, y_{m+1} >$
   Now, if there is a one-step transition from $< q_i, y_i >$ to $< q_{i+1}, y_{i+1} > \forall i \epsilon [0, m-1]$
   So, $y_i = a_i y_{i+1}$, where a_i is alphabet of the NFA along with ϵ
   So there should be a transition from q_i to q_{i+1} by reading a_i. So now, to check whether S is accepting the configurations sequence of F:-

(a) First, we will check that S starts with configuration $< q_0, x >$ for some $x \epsilon \sum^*$, $\sum$ is the alphabet of the F.

(b) For each $< q_i, y_i > \forall i \epsilon [1, m]$ there should be a edge from q_{i-1} to q_i by reading a_i such that $y_{i-1} = a_i y_i$.

(c) The last configuration should be of the form $< q_m, y_m >$ where q_m is accepting state and y_m is empty string.

Since the above-described steps give an algorithm that can compute if S is an accepting configuration sequence. So, S is computable.

3. **We have to prove there exists an $\mathcal{F}$ such that the set of all accepting configuration sequences of $\mathcal{F}$ are not a CFL.**

Lemma:- For an accepting input of length n, the length of the accepting configuration sequence is $n+1$ in a **DFA**.

Proof:- By 1st part, there will be an accepting configuration sequence for accepting input.
Let accepting input be $x = a_0 a_1 a_2 a_3 a_4 ... a_n$
Now let's give an accepting configuration sequence for x.

$$S =< q_0, y_0 >< q_1, y_1 >< q_2, y_3 > .... < q_m, y_m >$$

By definition of accepting configuration, $< q_i, y_i >$ is reachable to $< q_{i+1}, y_{i+1} >$ in one step $\forall i \in [0, m-1]$.
So, $y_i = t y_{i+1}$ where t cannot be ϵ because it is a DFA. so $|y_{i+1}| = |y_i| - 1$.

Claim: Now the claim is $|y_i| = n - i$, $i \in [0, m-1]$.

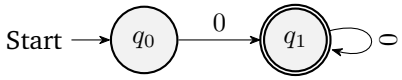
Proving it by induction on i.

Base case: $y_0 = x$, by definition of accepting configuration sequence. So $|y_0| = n$.
Induction Step: Let the above claim be true for some $p < m - 1$.

$$\text{So, by using } |y_{i+1}| = |y_i| - 1$$
$$|y_{p+1}| = |y_p| - 1 \implies |y_{p+1}| = n - (p+1)$$

Hence our claim is proved.
By using the above claim, $|y_m| = n - m = 0 \implies n = m$ .So the length of configuration sequence is $n+1$.

Assume a DFA $\mathcal{F}$ which accepts $\{0^n | n > 0\}$.



Proof by contradiction:
Assume there is a CFL C which accepts all accepting configuration sequences.
Let's use the pumping lemma for context-free languages to prove that C is not a CFL.

- Adversary chooses $m > 0$.
- Let $x = 0^{m+1}$, x is accepted by above DFA.
- Now let's give an accepting configuration sequence for x.

$$S =< q_0, x >< q_1, y_1 >< q_2, y_3 > .... < q_{m+1}, y_{m+1} >$$

$$|S| = m + 2$$

- Adversary breaks $S = puvwq$, such that $|uvw| < m$ and $|uw| >= 1$.
- Pumping 2 times gives $S_1 = pu^2vw^2q$. By pumping lemma, S_1 will also be accepted by the CFL C.
- Assume S_1 is an accepting configuration sequence.

- The first state of S_1 will be $< q_0, x >$. So by definition of accepting configuration sequence, S_1 can only accept x, but $|S_1| = |S| + |uw| >= |S| + 1 > |x| + 1$ which is wrong by above lemma, so we arrive at a contradiction.
- Hence $|S_1|$ is not an accepting configuration sequence, so C is not a CFL. Hence there is not any CFL for above DFA.

Q2: A $2-PDA$ is a pushdown automata with two stacks. In a transition, the automata can push/pop both stacks independently. Prove that any computable set can be accepted by a $2-PDA$

Theorem: Turing Machine can simulated by 2-PDA.
Proof: The idea is that **the first stack** holds whatever is to the left of the head, while **the second stack** holds what is at the head and right of the head, neglecting all the infinite blank elements beyond the leftmost and rightmost of the head.
Let the one-tape TM M be defined as:

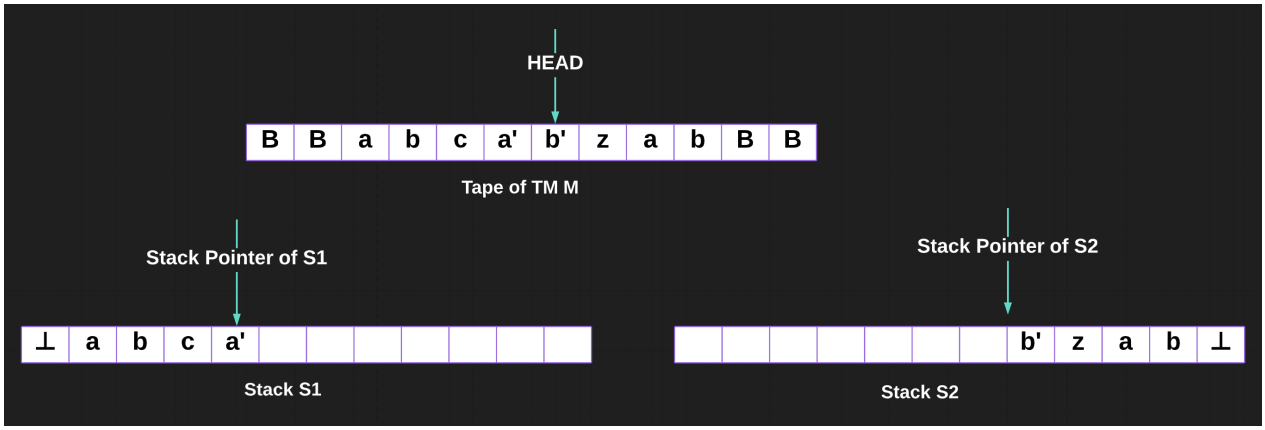
$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$$

So lets define our new PDA with finite control(states and transitions) but two stacks (S_1 and S_2) as follows:

$$P = (Q^{'}, \Sigma^{'}, \Gamma^{'}, \delta^{'}, q_0^{'}, \perp, F^{'})$$

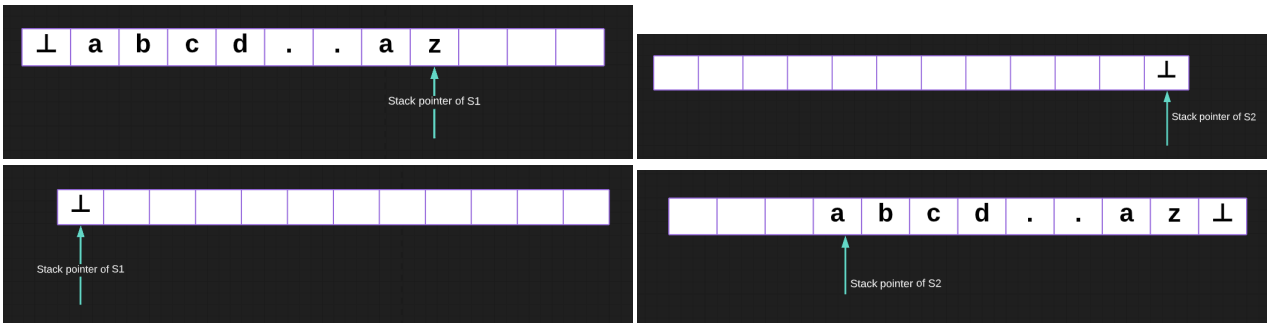
where

- $\perp$ represents the bottom of both stacks.
- $\Gamma^{'} = \Gamma \cup \perp$. Stack alphabets are same for both the stack.
- First lets simulate the tape with the help of two stacks. As mentioned above, the contents to the left of the head in TM M will be written on the first stack S_1 and the contents at and on the right of head will be written on S_2. **This means the Stack Pointer of S_2 simulates the head of TM M.**



- $\Sigma^{'} = \{\epsilon\} \cup \Sigma$
  - Suppose that string w is the input to the TM M. So the PDA P will copy the input w onto its first stack and stops when the input is completed.
  - P then pops the input from its first stack S_1 and pushes it onto the second stack S_2, such that S_2 holds input w, with the left end of w is at the top of S_2.

Let $w = abcd.....az$
where $a, b, c, d, z.... \in \Sigma^{'}$



  - The empty first stack indicates the fact that M has a blank to the left of the head. P has second stack holding w indicates that the head is pointing to the start of the input.

- $F = \{q_A\} \cup \{q_R\}$
The PDA will stop as soon as it reaches either accepting or rejecting state. If it reaches accepting state, it accepts the input, else rejects.

- The transition in the PDA will be as follows:
  - **Case 1**: If M reads X from head position and writes Y and moves right, then P pops X from second stack and pushes Y onto its first stack, representing the fact that Y is now at the left of M's head (see the following diagram).

Figure: **Transition in Turing Machine** M

Corresponding transition in the PDA P will be:

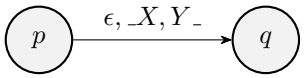
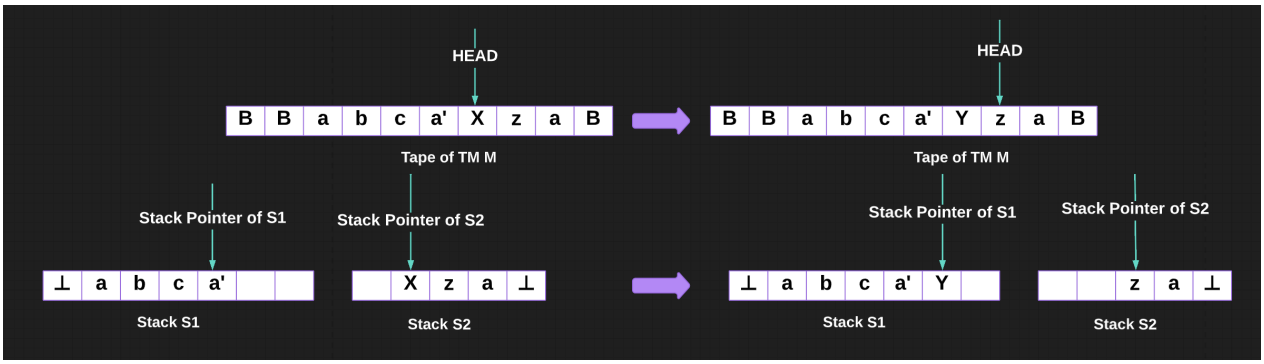


Figure: **Transition in PDA** P



B in the figure is the blank symbol in tape.

The transition in PDA P are of the form $(input, AB, PQ)$, where input is the input symbol read, AB means A is popped from first stack and B is popped from second stack. PQ represents that P is pushed into first stack and Q is pushed into second stack. $(input, \_B, P\_)$, here $\_$ means that nothing is pushed or popped from the corresponding stack.

– **Case 2**: If M reads X from head and writes Y and moves left, P pops the top of the first stack , say Z, then replaces X by ZY on the second stack. This represents what used to be one position left of the head is now at the head (see the following diagram).

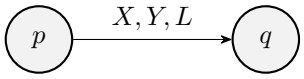


Figure: **Transition in Turing Machine** M

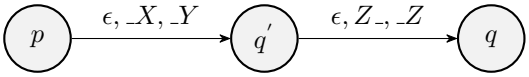
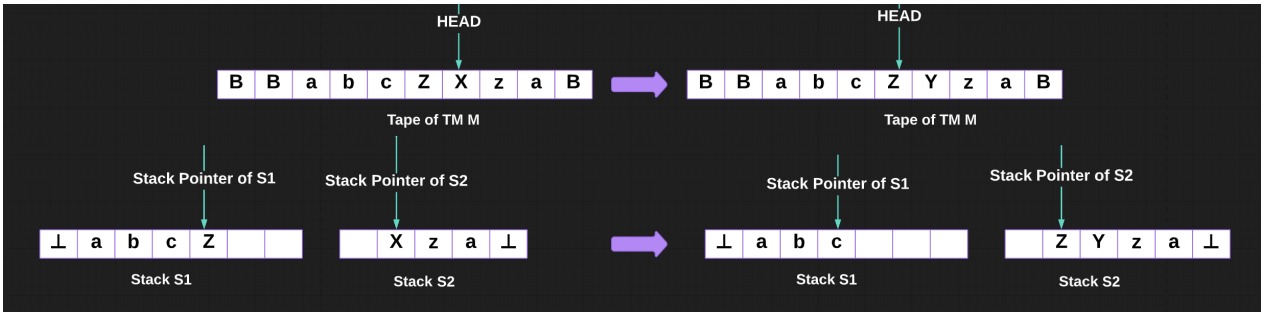


Figure: **Transition in PDA** P



B in the figure is the blank symbol in tape.

– **Case3:** Case 1 and 2 works in trivial cases but there might be possibility that the right stack goes empty and we move more right or the left stack goes empty and we move left.
So, to take care those states we will introduce two extra states and four transitions on each state.

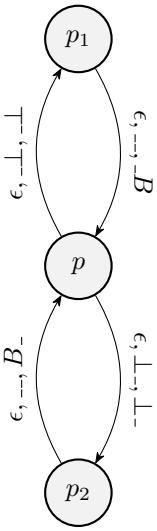


Figure: **Transition in Turing Machine** M

So combining all these three cases and transitions we will get transitions of 2-PDA.

Q3.A $counter$ TM is a Turing machine with an input tape that is read-only (input is initially written on the tape and cannot be changed during computation) and a finite number of counters. In one move of the TM, a counter either remains unchanged, or is incremented by one, or is decremented by one. Prove that every computable set can be accepted by a counter TM.

Solution: A $k - counter\ TM$ is a machine with a two-way read-only input tape and k integer counters.
Let us show that a stack can be simulated by 2 counters.

- We can assume, without loss of generality, that the stack has only two symbols 0 and 1. This is because if there are n stack symbols, then each symbol can be represented in binary by $log_2 n$ bits. Pushing or popping one symbol in the previous case can be simulated by pushing or popping $log_2 n$ binary symbols .
- Then the contents of stack can be viewed as a binary number whose **least significant bit** is the top of the stack.
- First counter stores this binary number(in decimal form).
- When 0 is pushed into the stack, the number is first counter has to be **doubled**. To simulate this we enter a loop which subtracts 1 from the first counter and adds 2(add 2 means adds 1 two times) to the second counter till the first counter is 0. The second counter now holds twice the value in first counter. Now the roles of two counters are switched.
- To simulate pushing of 1, add 1 to the second counter after executing the loop described above.
- When a element has to be popped, value in first counter has to be halved. To do this, subtract 1 from first counter and add 1 to the second counter **every second step**. From the parity of original contents, we can determine the element being popped is 0 or 1.

Since two stacks can simulate any arbitrary TM and we can simulate one stack by two counters. Hence any arbitrary TM can be simulated by four counters.

We can also show that four counters can be simulated using two counters.

- Let the values in four counters be i, j, k, l, then the value in first counter will be $2^i 3^j 5^k 7^l$.
- When value of j is increased by 1, first counter is decremented by 1 and second one is incremented by 3(means 1 is added three times) till first counter becomes 0 and then switch roles of the counters.
- Similarly when j is decremented by 1, decrement first counter by 1 and increment second one every third step. Similar operation can be done for i, k, l.

Hence two counters are as powerful as two-stacks. Therefore, two counter TM can simulate any arbitrary TM.

Q4. Design a Turing Machine (draw the state diagram) to multiply two numbers represented in unary alphabet, that is, 1 is represented as 1, 2 as 11, 3 as 111,. . .. The input alphabet is $\sum = \{1, \times, =\}$. Input has the form $1^n \times 1^m =$, which denotes multiplication of number n with m. The TM should write the result of the multiplication,1^{nm}, as output immediately after the $=$ sign on the tape.

Solution: Lets visualise the state of tape just before and after multiplication.

1	1	1	×	1	1	=	B

State of tape before multiplication

1	1	1	×	1	1	=	1	1	1	1	1	1	B

State of tape after multiplication

Description of TM:

- We assume initially the head is at the starting of the input string. The two numbers m and n in unary format are separated by $\times$.
- q_0- Looking at the first unused symbol in input.
- q_1- Found a 1 in the first factor; Write B on that spot; skipping to the x symbol.
- q_2- Found x or found Blank symbol.
- q_5- Found a 1 in the second factor; changed it to B; skipping to the =
- q_6- Found =; Skipping to end of tape.
- q_7- Write 1 at the blank spot at the end of tape; Skipping back over to the second factor first blank symbol(first blank symbol from the right). Go back to the state q_2 and continue the entire loop again.
- q_3- Used all the 1s in the second factor; convert all the B in the second factor back to 1.
- q_4- Skip back over to the first Blank symbol(first blank symbol from the right) in first factor; And then loop back to q_0 state.
- $q_{accept}-$ Consumed the entire first factor; Machine Halts and ouput is written after $=$ sign on tape.

Formal Description of Turing Machine:

$$M = (Q, \sum, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_{accept}, q_{reject}\}$
- $\sum = \{1, \times, =\}$
- $\Gamma = \sum \cup B$ where B is the blank tape symbol.
- δ is described in the state Transition diagram below.
- The start state, the accepting state and rejecting state are q_0, q_{accept} and q_{reject}.

State Transition diagram of above TM

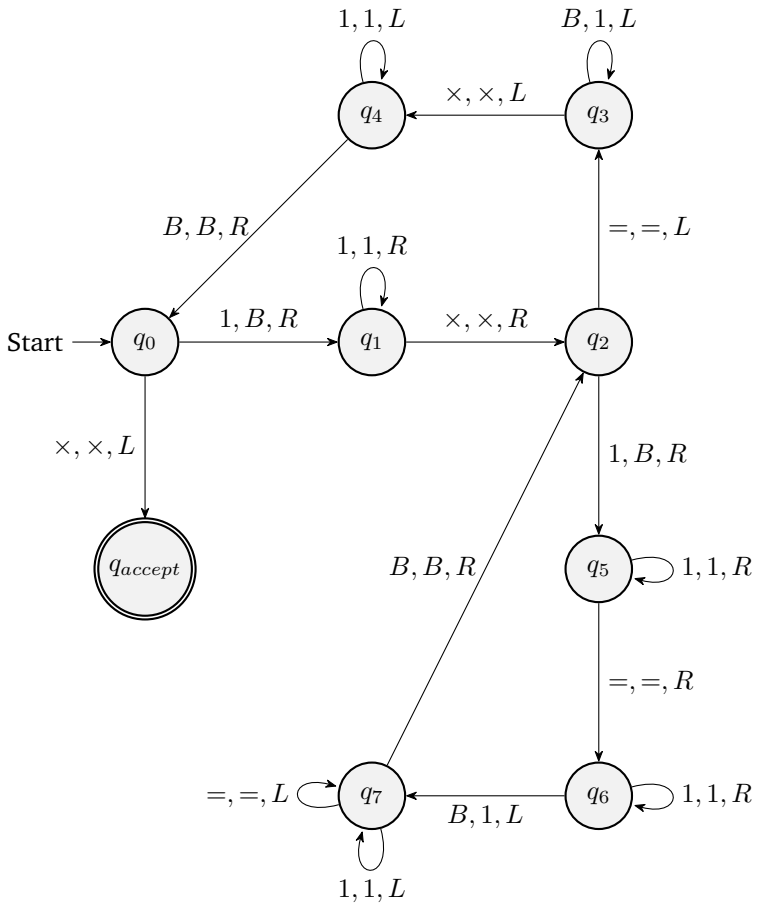


Figure: **Transition in Turing Machine**

Note: The transitions not shown for any state for any alphabet are assumed to go to q_{reject} state.
In brief our turing machine work as follows :- For each 1 in first factor cross off 1 in second factor and write that much 1 at the end. Now change the second factor again and repeat the above steps for next one 1 in first factor.

Notations:

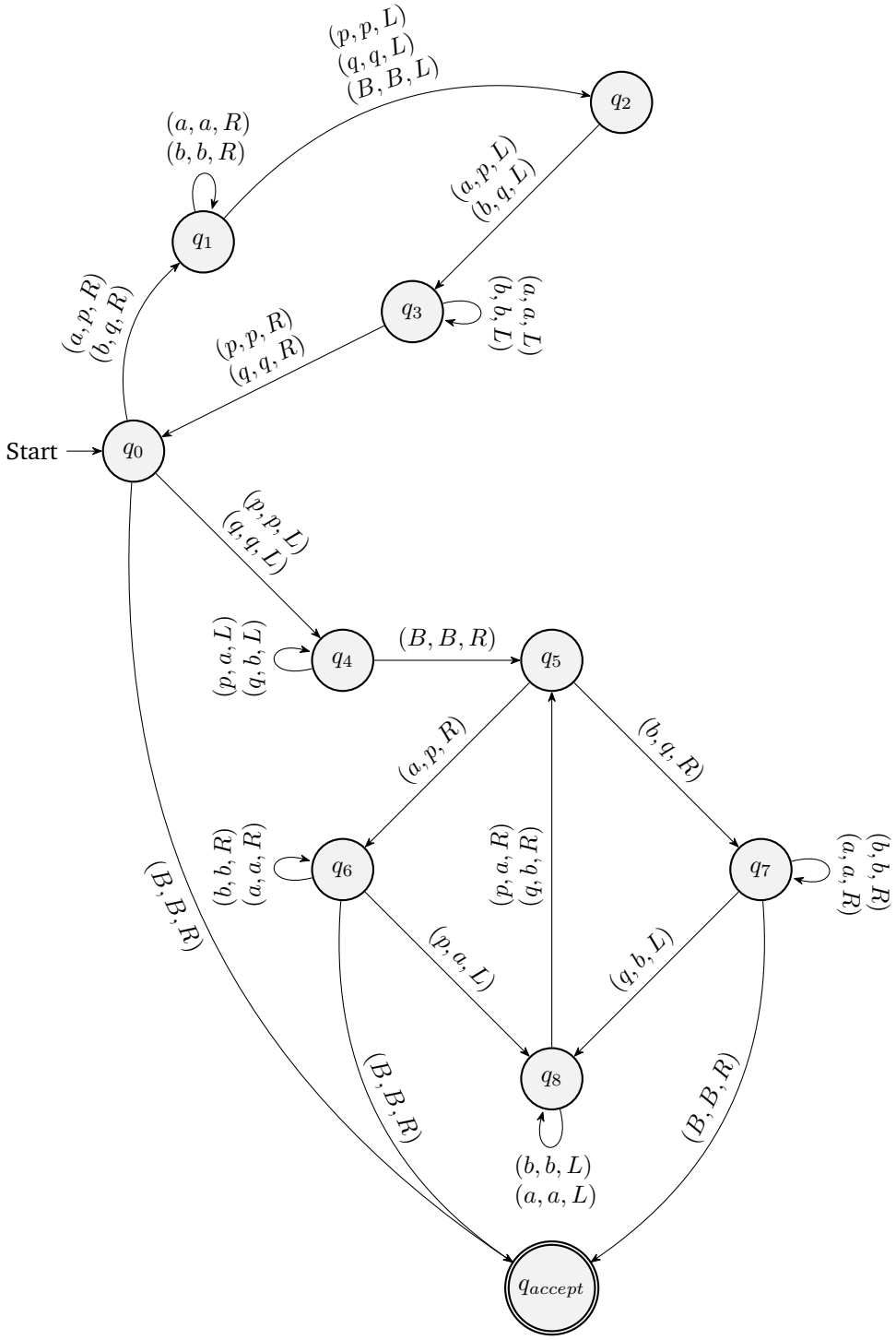
- Here q_0 is the start state and $q_1, q_2, ..., q_7$ are the transition states. q_{accept} is the accepting state.
- $\times, 1, =$ are the input symbol and B is the Blank Tape Symbol. R/L shows right or left.
- $\delta(q_i, a) = (q_j, b, R)$ is denoted by an arrow from state q_i to q_j and is labelled by (a, b, R).
- This means on reading a from the tape on current head position, the state machine moves from state q_i to q_j, writes b on the tape on the current head position and moves the head to the right.
- $\delta(q_i, a) = (q_j, b, L)$ is denoted by an arrow from state q_i to q_j and is labelled by (a, b, L).
- This means on reading a from the tape on current head position, the state machine moves from state q_i to q_j, writes b on the tape on the current head position and moves the head to the left.

Q5. Design TM

Let's construct a Turing Machine. The Turing machine described here has unbounded tape both the sides.
Formal Description of Turing Machine:

$$M = (Q, \sum, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_{accept}, q_{reject}\}$
- $\sum = \{a, b\}$
- $\Gamma = \sum \cup \{B, p, q\}$ where B is the blank tape symbol.
- δ is described in the state Transition diagram below.
- The start state, the accepting state and rejecting state are q_0, q_{accept} and q_{reject}.



Note 1: There exists these transition :- (a,a,R)/(b,b,R)/(p,p,R)/(q,q,R)/(B,B,R) from every state to q_{reject}. We have not written them in the diagram to make it readable. Please note that it makes the TM non-deterministic.

Basic algorithm our TM follows:-

1. We first try to find the center, so we will mark the first character (p/q) depending on a or b, respectively, and go to the right until a B/p/q is found.
2. Comes back to the previous by going left once. Mark the character by (p/q) depending on a or b, respectively.
3. Goes to the left until a (p,q) is encountered. Go to the next character by moving right once. Repeat the steps from 1.
4. If at step 1 there is neither a nor b. Go to the left once. Now we are in the middle of input.
5. Mark the left string from the current position (including it) to (a/b) depending on (p/q) respectively.
6. At this state, the left part of the input is of form $(a, b)^*$ and the right half of form $(p, q)^*$
7. Now checks if the first character is (a), change it to p and move right, and go to the right until a p is encountered. Mark it black and come to the left until (p/q) is encountered.
8. If at 7 we encounter (b), do the same thing for q instead of p.
9. Go right once and repeat 7.
10. If at 7 there is neither a nor b. Accept it.
11. We have taken care of empty string separately.

Q6. Prove that the following set is not computable: $L = \{(p, q)$ — there exists a string x accepted by both TM M_p and TM $M_q\}$.

To begin with our proof for the above, let us prove that the following set is undecidable (or not computable).

$$E_0 = \{r| \text{ such that } M_r \text{ is a TM that accepts } 0\}$$

Proof by contradiction

M_q: Let it be the TM that decides E_0.

We will construct a TM that decides the Halting Set with the help of M_q.

M_s: Let it be a TM as follows:

- Input to M_s: (p,x)
- Working of M_s :
  - It constructs the description of the TM M_r which works as described below.
  - After generation of description, it runs r on M_q and accepts iff M_q accepts r.

M_r : Let it be a TM as follows:

- Input to M_r: y
- Working of M_r :
  - On input y, it ignores y and simulates M_p on x, where (p, x) are input to TM M_s.
  - It accepts iff M_p halts(accepts or rejects) on x.

M_r either accepts ϕ or Σ^*.
M_s is the TM that solves the Halting Problem.

Proof of Correctness

M_s accepts $(p, x) \Rightarrow M_q$ accepts $r \Rightarrow M_r$ accepts 0 $\Rightarrow M_r$ accepts $\sum^* \Rightarrow M_p$ halts on x.

M_s rejects $(p, x) \Rightarrow M_q$ rejects $r \Rightarrow M_r$ does not accept 0 $\Rightarrow M_r$ accept $\phi \Rightarrow M_p$ does not halt on x.

Hence M_s is the TM that decides/solved Halting Problem which is contradiction as Halting Problem is undecidable. So set E_0 is undecidable.

Now let us prove that L is undecidable using the fact that E_0 is undecidable.

Proof by contradiction:
Let us suppose the Turing machine M_l that decides set L.
Construct a TM M_e that decides E_0 as follows:

- Input to M_e : r where M_r is a TM.
- Working of M_e :
  - It generates description of two TM M_p and M_q,

$$M_p = M_r$$
$$M_q = \text{TM that accepts } 0 \text{ only.}$$

  - It then simulates M_l on input (p, q) and accepts if M_l accepts (p, q) else rejects.

Proof of correctness

$$
\begin{gathered}
M_e \text{ rejects } r \\
\Downarrow \\
M_l \text{ rejects } (p, q) \\
\Downarrow \\
\nexists \text{ any x accepted by both } M_p \text{ and } M_q \\
\Downarrow \\
M_p \text{ does not accept } 0 \\
\Downarrow \\
M_r \text{ does not accept } 0 (M_p <=> M_r)
\end{gathered}
$$

$$
\begin{gathered}
M_e \text{ accepts } r \\
\Downarrow \\
M_l \text{ accepts } (p, q) \\
\Downarrow \\
\exists \text{ a x accepted by both } M_p \text{ and } M_q \\
\Downarrow \\
M_p \text{ accepts } 0 \\
\Downarrow \\
M_r \text{ accepts } 0 (M_p <=> M_r)
\end{gathered}
$$

Hence M_e is a TM that decides E_0 which is a contradiction as E_0 is undecidable. Hence, set L is undecidable.