INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

THEORY OF COMPUTATION

CS340

Assignment 2

Authors:
Kajal Deep (200483)
Uttam Kumar (201071)
Nishant Roshan (200643)

November 10, 2022

Question 1

For a finite automata $\mathcal{F} = (Q, q_0, \sum, \delta, F)$ and input x, define a configuration of $\mathcal{F}$ to be the pair ⟨q, y⟩ where q ∈ Q, y is a suffix of x, and automata F on input x = zy, ends up in state q after reading z. Note that if F is an NFA, then there may be more than one state in which $\mathcal{F}$ ends in after reading z. An accepting configuration sequence is a string $<q_0, y_0><q_1, y_1><q_m, >$ where (1) each $\langle q_i, y_i\rangle$ is a configuration, (2) F moves from configuration $\langle q_i, y_i\rangle$ to $\langle q_{i+1}, y_{i+1}\rangle$ in one step, (3) $q_m \in \mathcal{F}$ and (4) $y_0 = x$.
Prove that
• Input x is accepted by $\mathcal{F}$ iff there exists an accepting configuration sequence that starts with $<q_0, x>$.
• The set of all accepting configuration sequences of $\mathcal{F}$ is computable but is not a CFL.

Solution:
Part 1:
To prove the iff condition, we must prove its correctness both ways.

To prove: **Input x is accepted by $\mathcal{F}$ $\implies$ ∃ an accepting configuration sequence starting with** $\langle q_0, x\rangle$
We need to construct an accepting configuration sequence for every input that is accepted by the automata. Following is the construction of such a sequence:

- Create an alphabet of the string sequence corresponding to each state in the path from the start state to accepting state.
- For the start state, the corresponding first letter of the string will be $<q_0, x>$
- If there is a state transition from state q_1 to state q_2 in $\mathcal{F}$ on input symbol α, and if the letter corresponding to state q_1 in the sequence is $<q_1, \alpha A>$, where α is an alphabet while A is the suffix string, then the letter corresponding to state q_2 would be $<q_2, A>$
- For the accept state, the last letter of the sequence will be $<q_m, >$, where $q_m \in F$

Therefore we defined an accepting configuration sequence inductively for each accepted input.

To prove : ∃ **accepting configuration sequence starting with** $<q_0, x>$ $\implies$ **input x is accepted by** $\mathcal{F}$
This part is simple, as the first letter in the sequence ($<q_0, x>$) would imply that x enters $\mathcal{F}$ with q_0 being the start state and the last letter in the sequence ($<q_m, >$, where $q_m \in F$) would mean that input x exhausts on a final state. Hence x is an accepted input.

Part 2:
The set of all accepting configuration of $\mathcal{F}$ is computable.
Proof:
Given the finite automata $\mathcal{F}$ and the input to be x.
We construct a Graph $G_{\mathcal{F},x}$ with the following properties:

- The configuration will represent the vertices of the graph.
- There will be a directed edge form configuration c1 to configuration c2 if it is possible to reach c2 form c1 in one step.
- Although by definition $G_{\mathcal{F},x}$ contains all possible configurations, but we will usually be interested in only those configurations in $G_{\mathcal{F},x}$ that are reachable from the start configuration.

Now our problem is to find all the paths that starts from the start configuration ($<q_0, x>$) and ends at one of the accepting configuration $<q_m, >$ where $q_m \in F$.
Solution:
The idea is to do Depth First Traversal given directed graph
For every $q_m \in F$ we make it the destination vertex and do the following.

- Start the DFS traversal from the source.
- Keep storing the visited vertices in an array say 'path[]'.
- If the destination vertex is reached, the contents of path[] is an accepting configuration sequence.
- The important thing is to mark current vertices in the path[] as visited also so that the traversal doesn't go in a cycle.

Following these steps we are able to figure out all accepting configuration sequences of $\mathcal{F}$.
Hence, **F** is computable.

Proof that accepting configurations of $\mathcal{F}$ is not a CFL:-
Now we will prove that the set of all the accepting configurations of $\mathcal{F}$ is not a CFL using Pumping Lemma.
$\forall$ p $\geq$ 0, we will find a w $\in$ (set of all accepting configuration sequences) with length of w $\geq$ p, such that $\forall$ possible partitions of w as w = uvxyz, satisfying

- $|vxy| \leq$ p
- $|vy| > 0$

such that there exist an i $\geq$ 0 such that $uv^ixy^iz \notin$ L the set of all accepting configuration sequences
Here w represent the string which is a sequence accepting configuration and an alphabet is represented by a configuration.
It can easily be verified that the pumping lemma cannot hold in this case.For the pumping to hold true it must be the case that there are two cycle in accepting configuration represented by the string v and y. But we can also achieve an accepting configuration with just one cycle repeated for arbitrary large p.
Hence pumping lemma is violated and the set of all accepting configuration is not a CFL.

Question 2

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.

Solution:
We know that any computable set is accepted by some turning machine. We will prove that the class of two stack machines (2-PDA), is equivalent to the class of turing machines in terms of computablity.
Proving this needs us to prove two things:

1. **A TM can simulate a 2-stack PDA**
   That is we need to prove that whatever input is accepted by a 2-stack PDA is accepted by a TM.
   Proof:
   Pushdown automata is non-deterministic. Therefore, we choose a non-deterministic Turing machine M to simulate the Two-Stack PDA. The construction of the TM will be as follows:

   - We split the tape of the Turing machine into two half.
   - We place a marker symbol in the tape to indicate the border between the two half.
   - The part of the tape left from the marker simulate one stack.
   - The part of the tape right from the marker simulate the second stack
   - The bottom of the stack is the marker

   This is equivalent to simulating the two stacks using two-tape Turing machine such that every tape simulates one stack.

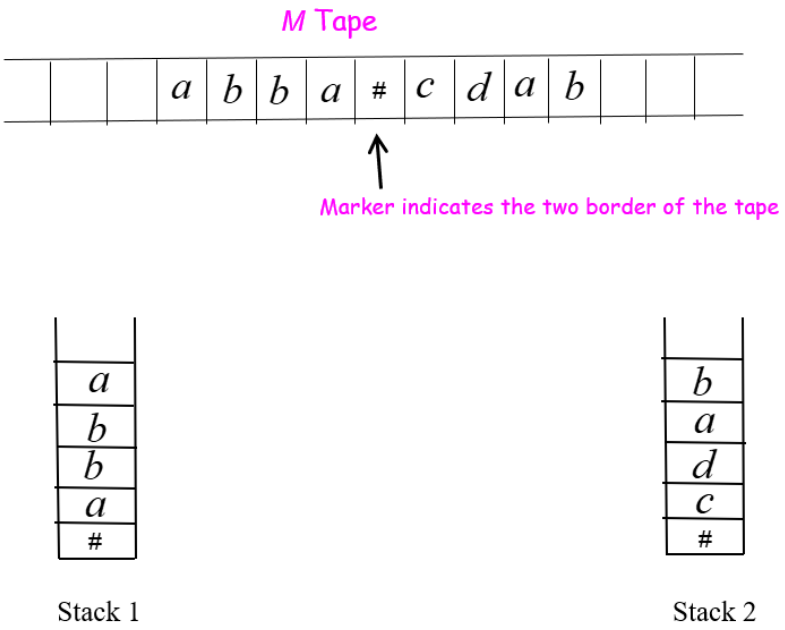


Figure 1

2. **A 2-stack PDA can simulate a TM**
   We basically need to prove that if a language L is accepted by a Turing machine, L is accepted by a Two-Stack machine.
   Proof:
   The idea is that the first stack can hold what is to the left of the head, while the second stack holds what

is to the right of the head, neglecting all the infinite blank symbols beyond the leftmost and rightmost of the head.
Let L be L(M) for some one tape TM M, two-stack machine S, simulating a one-tape TM M as the following:

- S begins with a bottom-of-stack marker on each stack, this marker considered the start symbol for the stacks, and must not appear elsewhere on the stacks. The marker indicates that the stack is empty.
- Suppose that w is on the input of S. S copies the input w onto its first stack, and stops to copy when reading the endmarker on the input.
- S pops each symbol in turn from its first stack and pushes it onto its second stack. The first stack of S is empty. The second stack holds w, with the left end of w at the top.
- S simulated the first state of M. The empty first stack indicates the fact that M has a blank to the left of cell scanned by the tape. S has a second stack holding w indicates the tape head point to the left most symbol in the string w (see the following diagram).

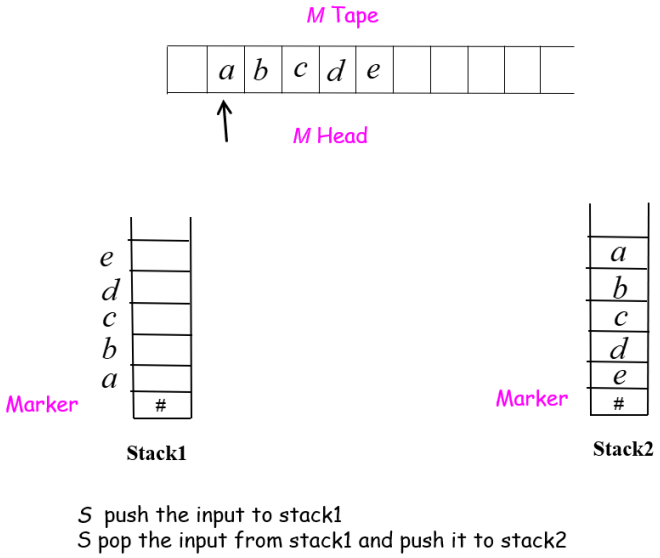


Figure 2

- In simulating a move of the TM M, S maintains the invariant that the symbol to which the head points in the TM is the top of the second stack.
- If M replaces X by Y and moves right, then S pushes Y onto its first stack and pops X from the second stack, representing the fact that Y is now at the left of M's head (see the following diagram).

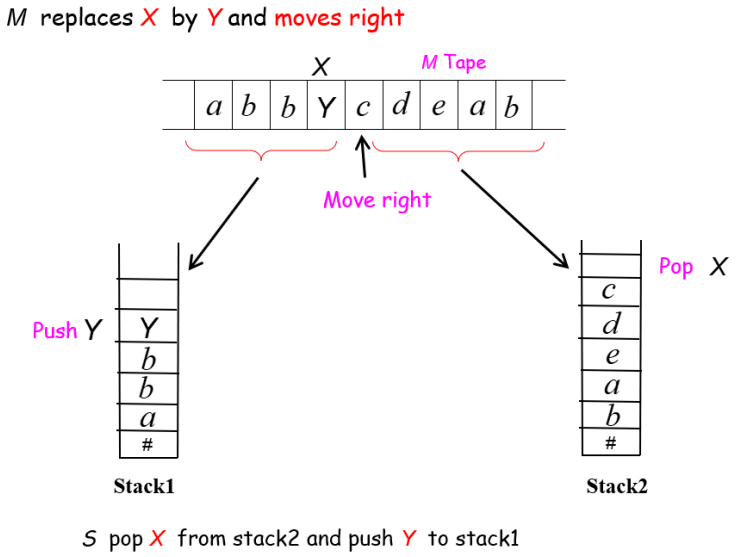


Figure 3

- If M replace X by Y and moves left, S 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).

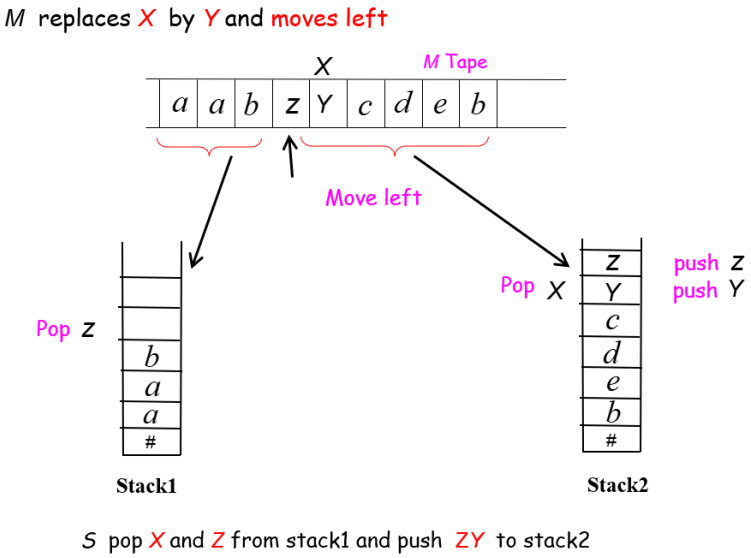


Figure 4

- Finally, S accepts if the new state of M is accepting. Otherwise, S simulates another move of M in the same manner

Question 3

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:
Now we know that a two-stack machine can simulate an arbitrary TM. Here we are going to show that two counters can simulate a stack. This will give us the conclusion that a counter TM having four counters can simulate an arbitrary TM.

We can simulate a stack with two counters as follows:
Without loss of generality assume that the stack alphabet of the stack to be simulated contains only two symbols 0 and 1. We can achieve this by encoding finitely many stack symbols as binary numbers of some fixed length say k.
Pushing or popping one stack symbol is simulated by pushing or popping k binary digits. Then the contents of the stack can be regarded as a binary number whose least significant bit is on the top of stack.

Now we are going to use two counters Counter1 and Counter2 to simulate a stack.

1. Counter1 will store the contents of the stack
2. Counter2 will we used as scratch space (we are going to see).

- How can we push a 0.
  - Multiply the counter by 2
- How can we push a 1.
  - Multiply the counter by 2,and then add 1 to it

Now we can only increment or decrement a counter by 1 so how can be multiply a counter by 2?
This can be done as follows:
Suppose we want to multiply counter1 by 2.

- Set counter2 to 0
- While counter1 is not 0:
  - Decrement counter1
  - Increment counter2 twice
- Swap Counter1 and Counter2

So, by following above steps we can simulate a push a 0 or 1.
Pop operation:

- While Counter1 $\neq$ 0.
  - Decrement Counter1
  - If Counter1 becomes 0 our popped result is 1
  - Decrement Counter1
  - If Counter1 becomes 0 our popped result is 0
  - Increment Counter2
- Swap Counter1 and Counter2

Swap Operation:
In swap operation we want the contents of Counter2 to be copied to Counter1 as Counter2 stores the end result. Note that Counter1 is 0 at beginning of swap operation. This can be done as follows:
Increment Counter1 by 1 and decrement Counter2 by 1 until Counter2 becomes 0.
We also need to have a method to check if the stack is empty.
To achieve this we use 1 to represent an empty stack. Stack is empty if (counter1 - 1) = 0).
So whenever we start we will have Counter1 as 1 (represents empty stack) and Counter2 starts out as 0.

Conclusion: Two counters can simulate a stack.Two stacks can simulate a Turing machine. Therefore, four counters can simulate a Turing Machine.Therefore, a Turing machine with an input tape that is read only and a finite number (4 here) of counters where in one move of the TM, a counter either remains unchanged, or is incremented by one, or is decremented by one, can accept every computable set.

Question 4

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.

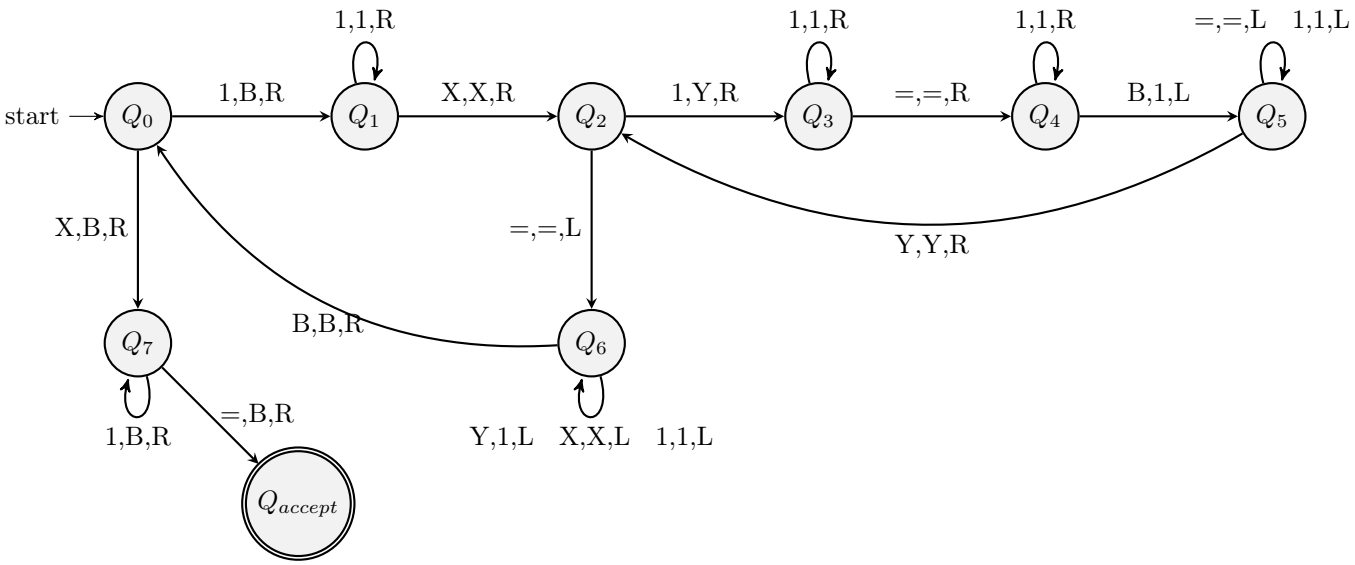
Solution:
Approach Used: The approach is to write the multiplier multiplicand number of times (ex- 2*3 = 3+3 i.e, 11 × 111 = "111" + "111" = 111111). To do this, we need to write 1s in the output as many times as there are 1s in the multiplier. This is done as:

copy-step - Replace 1 in the multiplier by Y and write 1 (i.e replace B) in the output (after =). Repeat this for all the 1s in the multiplier. After this all the Ys are again replaced with 1s.

Thus, replace all the 1s of the multiplicand by Bs one by one, repeating the copy-step each time after replacing 1 by B. Once all the 1s in the multiplicand is replaced with Bs, the multiplication is done.

Steps:

- Step-1: Start from state Q0.
  If symbol is 1, replace it by B and move right. Go to state Q1 and Step-2.
  Else if symbol is ×, replace it by B, move right and go to state Q7. Again move right until = occurs replacing all 1 by B, remaining on the same state. Then replace = by B, move right and go to the accept state.
- Step-2: Replace all 1 by 1, move right and remain on the same state until × occurs. Go to state Q2 and Step-3.
- Step-3: At state Q2,
  If the symbol is 1, replace it by Y, move right and go to state Q3 and Step-4.
  Else if symbol is =, replace it by =, move left and go to state Q6. Replace all Y by 1, 1 by 1 and × by ×, moving left and remaining on the same state until B occurs. Then replace B by B, move right and go to state Q0 and Step-1.
- Step-4: Move right replacing all 1 by 1, while remaining on the same state. When = occurs, replace it by =, move right and go to state Q4 and Step-5.
- Step-5: Replace all 1 by 1, moving right and remaining on the same state. When B occurs, replace it by 1, move left and go to state Q5 and Step-6.
- Step-6: Replace all 1 by 1 and = by = while moving left and remaining on the same state. Then if Y occurs move right and go to state Q2 and Step-3.


start
Q_0
Q_1
Q_2
Q_3
Q_4
Q_5
Q_6
Q_7
Q_{accept}
1,1,R
1,B,R
X,X,R
1,Y,R
1,1,R
=,=,R
1,1,R
B,1,L
=,=,L
1,1,L
X,B,R
=,=,L
Y,Y,R
B,B,R
1,B,R
=,B,R
Y,1,L
X,X,L
1,1,L

Question 5

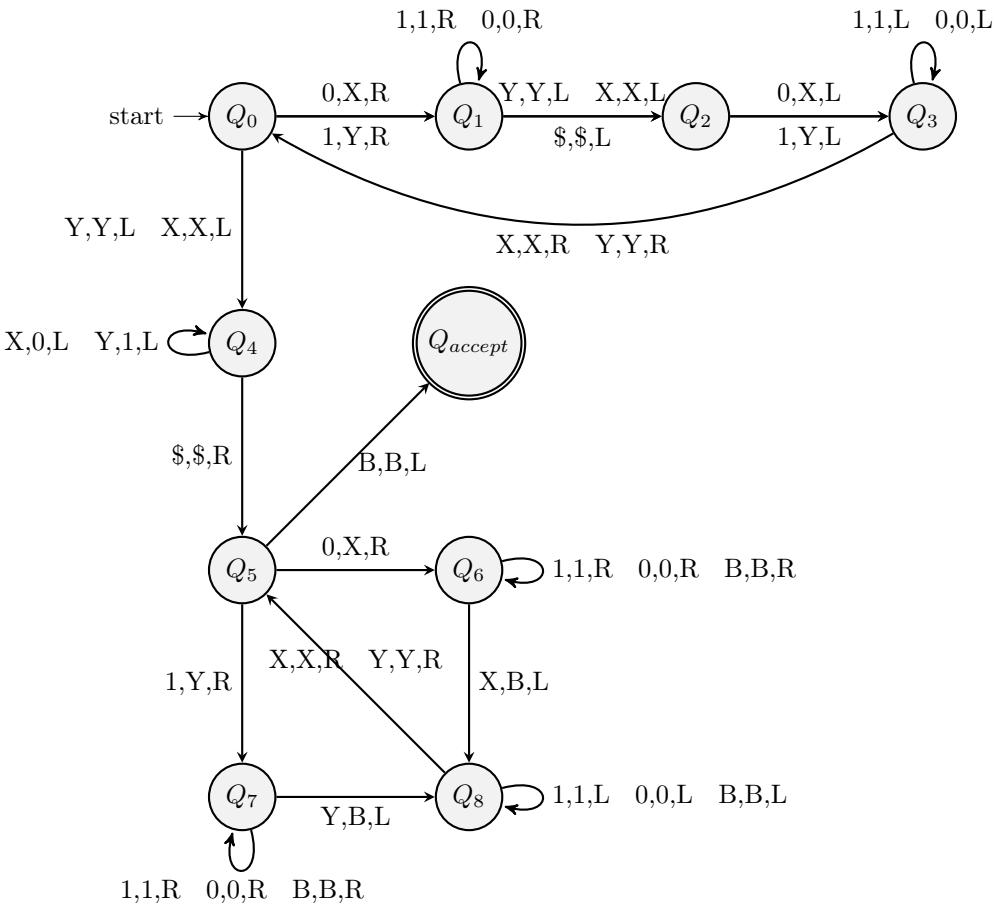
Design a TM (draw the state diagram) that accepts following set of strings:
$$L = \{ww \mid w \in \{a, b\}^*\}$$

Solution:
The Turing machine which solves this problem follows the following algorithm to decide which string to accept and which to reject. It majorly consists of the following three steps:

- **Finding the mid point of the string**
  For this if the first symbol is 1, convert it into Y, while if it is 0, convert it into X. Now move till the right end of our string and convert corresponding 0 or 1 into X or Y respectively. Till now we have our string in which the first and last symbol are X or Y and rest are 0 or 1.
  Now move the head left till, we find a X or Y. When we do so, convert the 0 or 1 right of it to X or Y respectively and then do the same on the right end. Keep repeating this step till there are no 0's or 1's to the left of our head (all the 0's and 1's have been converted to X and Y respectively). Our objective of finding the midpoint is satisfied.

- **After we have found the mid point we change half of the symbols**
  At present our head points to the first symbol of the right half.Convert all the X and Y on the left of the midpoint (left of present head) into 0 and 1 respectively.
  Now our head is at the first symbol of the string with the left half consisting of 0's and 1's while the right half consisting of X and Y.

- **Now we match the symbols**
  Convert the first 1 or 0 on the left half into Y or X respectively and match it with the first Y or X on the right half. If both matches, replace the Y or X in the right half by the blank symbol (B) and take the head to the first 1 or 0 on left half to repeat the process. If both don't match, reject the string.
  Keep doing this till all the symbols on the left part of the string are converted into X and Y and all symbols on the right of string into blanks.
  If any one part is completely converted but still some symbols in the other half are left unchanged then the string will not be accepted. Else accept the string.


1,1,R 0,0,R
1,1,L 0,0,L
start
Q_0
0,X,R
1,Y,R
Q_1
Y,Y,L X,X,L
$,$,L
Q_2
0,X,L
1,Y,L
Q_3
Y,Y,L X,X,L
X,X,R Y,Y,R
X,0,L Y,1,L
Q_4
Q_{accept}
$,$,R
B,B,L
Q_5
0,X,R
Q_6
1,1,R 0,0,R B,B,R
X,X,R Y,Y,R
1,Y,R
X,B,L
Q_7
Y,B,L
Q_8
1,1,L 0,0,L B,B,L
1,1,R 0,0,R B,B,R

Question 6

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}.

Solution:
Let B_{TM} be the turing machine for L.
B_{TM} is undecidable.
Descriptions of TMs used in Proof:
A_{TM} = {(M,w)| M is a TM and M accepts w}
E_{TM} = {<M>| M is a TM and L(M) = ϕ}

Proof Idea:
We will show that if B_{TM} is decidable, E_{TM} also would be decidable by giving a reduction from E_{TM} to B_{TM}.The idea is simple. E_{TM} is the problem of determining whether the TM M which when provided as input to E_{TM} accepts if M accepts no string x (ie. M is empty) and rejects if M accept at-least one string x. B_{TM} is the problem of determining whether there exist a sting x accepted by both the input turing machines(M_p and M_q). If we set one of the one of the turing machines say M_p to accept every string we end up with the problem of determining E_{TM} problem. So in a sense , the E_{TM} problem is a special case of the B_{TM} problem wherein one of the machines is fixed to recognize the every string. E_{TM} accepts if B_{TM} rejects and E_{TM} rejects if B_{TM} accepts. This idea makes giving the reduction easy.

PROOF We let TM R decide B_{TM} and construct TM S to decide E_{TM} as follows.
S = "On input <M>, where M is a TM:
1. Run R on input $< M, M_1 >$, where M_1 is a TM that accepts all inputs.
2. If R accepts, reject; if R rejects, accept."
If R decides B_{TM}, S decides E_{TM}. But E_{TM} is undecidable so B_{TM} is also undecidable.
Hence set L is not computable.

E_{TM} is undecidable
Proof Let us assume E_{TM} is decidable then we will show that A_{TM} (A_{TM} is TM which when provided with input <M,w>, accepts if M accepts w and rejects if M does not accept w) is also decidable - a contradiction. Let R be a TM that decides E_{TM}. We use R to construct TM S that decides A_{TM}. How will S work when it receives input $< M, w >$?
One ides is for S to run R on input $< M >$ and see whether it accepts. If it does, we know that L(M) is empty and therefore M does not accept w. But if R rejects $< M >$, then we know that L(M) is not empty and therefore that M accepts some string (But we don't know whether M accepts the particular string w).
So, we use the following tactics. Instead of running R on $< M >$, we run R on a modification of $< M >$. Modify <M> to guarantee that M rejects all strings except w, but on input w it works same as usual. Then we use R to determine whether the modified machine recognizes the empty language. The only string the machine can now accept is w, so its language will be nonempty iff it accepts w. If R accepts when it is fed a description of the modified machine, we know that the modified machine doesn't accept anything and that M doesn't accept w.
PROOF
Let M1 be the modified machine described above.
Input: <M1> where M1 is a TM
M1 = "On input x:
1. if x $\neq$ w , reject.
2. if x = w , run M on input w and accept if M does."
We assume that TM R decides E_{TM} and contruct TM S tat decides A_{TM} as follows:
S = "On input $< M, w >$, an encoding of a TM M and a string w:
1. Use the description of M and w to construct the TM M1 described.
2. Run R on input $< M1 >$.
3. If R accepts, reject; if R rejects, accept."
S must be able to compute a description of M1 from a description of M and w because it only needs to add extra states to M that perform the x = w test.
If R were a decider for E_{TM}, S would be a decider for A_{TM}. A decider for A_{TM} cannot exist, so we know that

E_{TM} must be undecidable.

A_{TM} is undecidable
Proof Let us assume A_{TM} is decidable. Then there must be a halting TM H that on input <M,w>, accepts if M accepts w and rejects if M does not accept w.
Using H,we construct TM N as follows:
Input: <M> where M is a TM
1. Simulate H on $< M, < M >>$. (description of H is hardcoded into the machine N.)
2. If H rejects then accept else reject. Note that the machine H is a halting TM, hence N is also a halting TM.
Now consider what happens when the machine N is provided with the input <N>. N accepts <N> $\Leftrightarrow$ H rejects $< N, < N >> \Leftrightarrow$ N does not accept <N> This is a contradiction. Hence A_{TM} is undecidable.

Thank You