Assignment 2

CS340: Theory of Computation

Abhishek Pardhi, Aayush Kumar, Sarthak Kohli

200026, 200008, 200886

UG Y20 - CSE

`apardhi20@iitk.ac.in, aayushk20@iitk.ac.in,`
`sarthakk20@iitk.ac.in`

Indian Institute of Technology
Kanpur

Department of Computer Science & Engineering

January 10, 2023

Contents

1 Question 1

1.1 1(a)

To Prove: Input x is accepted by the finite automata $\mathcal{F}$ iff there exists an accepting configuration sequence that starts with $< q_0, x >$

Solution:

We will prove both sides of the statement one by one.

Left to Right side: If input x is accepted by the finite automata $\mathcal{F}$, then there exists an accepting configuration sequence that starts with $< q_0, x >$

Let us consider $|x| = n$ where n is the length of the input string. We can write x as $x = x_1x_2x_3x_4........x_n$ where $x_i \in \sum$

Since x is accepted by the finite automata $\mathcal{F}$, there exists a sequence of transitions of states such that when the input ends, we land up in an accepting state say $q_n \in F$

We can write the transitions as follows:

1. $\delta(q_0, a_1) = q_1$
2. $\delta(q_1, a_2) = q_2$
3. $\delta(q_2, a_3) = q_3$.
   .
   .
4. $\delta(q_{m-1}, a_m) = q_m$ where $q_m \in F$

where $a_i \in \sum$ if $\mathcal{F}$ is a DFA and $a_i \in \{\sum \cup \{\epsilon\}\}$ if $\mathcal{F}$ is a NFA . Also $x = a_1a_2a_3......a_m$.

Observation: If $\mathcal{F}$ is a DFA then $m = n$. If $\mathcal{F}$ is a NFA then $m \geq n$ as some of the transitions may be ϵ transitions.

Construction of a Configuration Sequence:

Consider the following configuration sequence:

S=$< q_0, a_1a_2a_3....a_m >< q_1, a_2a_3a_4......a_m >< q_2, a_3a_4a_5....a_m >$ .... $< q_{m-1}, a_m >< q_m, >$

S **is accepting configuration sequence:** S follows the conditions mentioned in the problem as follows:

- Every configuration which is a part of S is a valid configuration since q_i is the state reached by $\mathcal{F}$ after reading z where z satisfies $x = za_{i+1}a_{i+2}.....a_m$ and $x = a_1a_2a_3....a_m$
- $\mathcal{F}$ moves from $< q_i, y_i >$ to $< q_{i+1}, y_{i+1} >$ in one step where $y_i = a_{i+1}a_{i+2}a_{i+3}....a_m$ because of the transitions as mentioned above
- $q_m \in F$ because of the transitions mentioned above
- $y_0 = a_1a_2a_3...a_m = x$

Right to Left Side:

If there exists an accepting configuration sequence that starts with $< q_0, x >$ then the input x is accepted by the finite automata $\mathcal{F}$

Proof

Let us denote the accepting configuration sequence that starts with $< q_0, x >$ as A

$A =< q_0, y_0 >< q_1, y_1 >< q_2, y_2 >$ .... $< q_m, >$ such that:

- $< q_i, y_i >$ is a valid configuration
- $\mathcal{F}$ moves from $< q_i, y_i >$ to $< q_{i+1}, y_{i+1} >$ in one step
- $q_m \in F$
- $y_0 = x$

To prove x is accepted by the finite automata $\mathcal{F}$ we need to find a valid sequence of transitions that takes place when input x is fed into the automata. If the last state of such transition sequence is an accepting state , then the input x is accepted by the finite automata.

Construction of a Valid Transition Sequence that ends in an Accepting State:

Consider the sequence of transitions of states:

$T = q_0 \rightarrow q_1 \rightarrow q_2..........q_m$

Proof of Validity:

- **The transition q_i to q_{i+1} is present in $\mathcal{F}$**: Since $\mathcal{F}$ moves from $< q_i, y_i >$ to $< q_{i+1}, y_{i+1} >$ in one step this implies, $y_i = a_{i+1}y_{i+1}$ where $a_{i+1} \in \sum$ if $\mathcal{F}$ is a DFA and $a_{i+1} \in \{\sum \cup \{\epsilon\}\}$ if $\mathcal{F}$ is a NFA
  Hence this implies that the finite automata $\mathcal{F}$ follows the transition $\delta(q_i, a_{i+1}) = q_{i+1}$
- **The transition T corresponds to the input x:**
  Input corresponding to the transition T is $a_1a_2a_3......a_m$
  Claim: $a_1a_2a_3.....a_m = x$
  Proof: $y_0 = a_1y_1 = a_1a_2y_2......a_1a_2a_3....a_m$ and $y_0 = x$ (given) so $a_1a_2a_3....a_m = x$
- **q_m is final state:**
  As per the definition of accepting configuration sequence

Hence T is a valid transition sequence that ends in an accepting state , hence $\mathcal{F}$ accepts x

Hence Proved

Since we proved both the sides of the statement, the bi-implication holds.

1.2 1(b)

To Prove: There exists a finite automata $\mathcal{F}$ such that the set of all accepting configuration sequences of $\mathcal{F}$ is computable but is not a CFL.

Solution:

Consider the DFA corresponding to the regular set:

$$L = \{1^n \text{ where } n > 0\}$$

Let us denote the DFA corresponding to the set L as $D = (Q, q_0, \sum, \delta, F)$ where $\sum = \{0, 1\}$

Proof 1: The set of all accepting configuration sequences of D is computable

We define a halting TM $M = (Q, q_0, \sum, B, \delta, F_{accept}, F_{reject})$

Here B refers to the blank spaces on the track. $F_{accept} = F$ and $F_{reject} = \phi$. All other parameters are same as that of D

Description of M:

1. Given an input s to M, find out the corresponding input to the DFA D as follows: s looks like $< p_0, y_0 >< p_1, y_1 >< p_2, y_2 > ..... < p_m, y_m >$. If p_0 is not equal to q_0 or y_m is not an empty string, then reject s, else simulate the input y_0 on the DFA D
2. If DFA D does not accept $y_0 = x$, then reject s directly, else go to the next step. Since we use a DFA, it always halt on finite input.
3. Let us consider the sequence of transitions look like $q_0 \to q_1 \to q_2 \to q_3 \to ....q_k$, where input is consumed at transition to q_k. If $k \neq m$ then reject s, else check for each $i \in [0, k]$ p_i should be equal to q_i, if not then reject s, else go to the next step
4. Also let $|x| = n$ where n is the length of the input string. Writing $x = x_1x_2x_3...x_n, x_i \in \sum$. Now for every i the following condition should hold $y_i = x_{i+1}x_{i+2}x_{i+3}...x_n$ since D is a DFA and we deterministically know the transition sequence of states when x is fed into D. If the input s violates this condition then reject it, else accept it.

Claim: M accepts the set of all accepting configuration sequences of D

Proof: A configuration sequence s accepted by our TM M satisfies the four conditions of accepting configuration sequence mentioned in the problem statement:

- p_0 should be q_0 and y_m should be empty
- By the first part of the question, we know that the DFA should accept y_0 for the configuration sequence to be accepting. Our Turing Machine also checks this.
- Next we need to check two more things:
  1. $< q_i, y_i >$ should be a valid configuration
  2. The move from $< q_i, y_i >$ to $< q_{i+1}, y_{i+1} >$ occurs in One step

  By imposing the conditions $y_i = x_{i+1}x_{i+2}...x_n$ and $q_i = p_i$ we ensure both the above requirements. Since x must be accepted by DFA D and $q_i = p_i$, it is easy to observe that $p_m \in F$. All sequences not of this form are rejected because of the conditions mentioned in description of M.

Proof 2: The set of all accepting configuration sequences of D is not a CFL
The set we are considering is $P = \{F \text{ where } F \text{ is an accepting configuration sequence } \}$
Proof by Contrapositive of Pumping Lemma for CFL's:

- Step 1: Opponent's Move: Consider any $p \geq 0$
- Step 2: Our Move: $\exists w \in P$ with $|w| \geq p$, w looks like $< q_0, y_0 >< q_1, y_1 >< q_2, y_2 > \ldots\ldots < q_m, >$ where w is an accepting configuration sequence
- Step 3: Opponent's Move: $\forall$ possible partitions of w as $w = uvxyz$, satisfying
  1. $|vxy| \leq p$
  2. $|vy| > 0$
- Step 4: Our Move: $\exists i \geq 0$ such that $uv^ixy^iz \notin P$
  Consider i to be a big positive number.
  Claim: $uv^ixy^iz \notin P$ for $i > 1$
  Proof:
  Claim (a): For the DFA D , $|y_{i+1}| < |y_i|$
  Proof (a): Since it is DFA, this means there is no epsilon transition involved , which means that the transition from configuration $< q_i, y_i >$ to $< q_{i+1}, y_{i+1} >$ utilizes one character of the input i.e $y_i = ay_{i+1}$ where $a \in \sum$. Hence $y_i = 1 + y_{i+1}$. Hence, we can say that $|y_i|$ is a monotonically decreasing sequence.
  Now, since $|vy| > 0$ we have:
  $|vy| = |v| + |y| > 0$ which implies atleast one of v, y is non-empty. Without any loss of generality, lets assume $|v| > 0$ and we pump v i times
  v looks like $< q_k, y_k >< q_{k+1}, y_{k+1} >< q_{k+2}, y_{k+2} > \ldots.\alpha$ times where α is length of v , $\alpha > 0$
  Now analyze $v^i = v^2v^{i-2}$:
  v^2 looks like $< q_k, y_k >< q_{k+1}, y_{k+2} > \ldots < q_{k+\alpha-1}, y_{k+\alpha-1} >< q_k, y_k >< q_{k+1}, y_{k+2} >$ ...
  By claim (a) $|y_k| > |y_{k+1}| > |y_{k+2}| \ldots\ldots > |y_{k+\alpha-1}| > |y_k|$
  $\rightarrow y_k > y_k$ which is a false
  $\rightarrow$ Hence $uv^ixy^iz \notin P$ for $i > 1$
  $\rightarrow$ Hence By Pumping Lemma, P is not a Context Free Language

2 Question 2

Strategy:
We will show that a 2-PDA is equivalent to a one-track one-head Turing Machine.
Let us consider any computable (that is, decidable) set A. Since A is decidable, we know that there is some halting Turing Machine that accepts A. As proven in class, any variant of Turing Machines can be converted to an equivalent one-track one-head Turing Machine. Thus, we can convert the Turing Machine that accepts A to this equivalent one-track one-head Turing Machine.
Hence, any computable set A can be accepted by a one-track one-head Turing Machine. Thus, showing that a 2-PDA is equivalent to a one-track one-head Turing Machine will be sufficient to prove that any computable set can be accepted by a 2-PDA. (As we can convert the one-track one-head Turing Machine that accepts the given computable set A to its equivalent 2-PDA).

Description:

Consider any one-track one-head Turing Machine, let it be M.
Let the leftmost non-blank letter in the tape of M be at index l_M, the rightmost letter be at index r_M and the head point to index h_M.
Then, the two stacks of the equivalent 2-PDA can be implemented in the following way: Stack 1 contains all the letters from indices l_M (at the bottom of the stack) to h_M (at the top of the stack), while Stack 2 contains all the letters from indices $h_M + 1$ (at the top of the stack) to r_M (at the bottom of the stack).
Thus, Stack 2 contains all the letters to the right of the head and Stack 1 contains all the other letters. The position of the head is represented by the top of Stack 1.
So, any movement of the head can be implemented as follows:

- Left movement: pop the element from the top of Stack 1, push the output to the top of Stack 2.
  In case Stack 1 has only the special bottom of the stack (Γ), we push the blank symbol (B) to the top of Stack 2 and do not pop from Stack 1.
- Right movement: pop the element from the top of Stack 2, push the output to the top of Stack 1.
  In case Stack 2 has only the special bottom of the stack (Γ), we push the blank symbol (B) to the top of Stack 1 and do not pop from Stack 2.

Construction:

Let M be a one-track one-head Turing Machine, defined as $M = (Q, q_0, \Sigma, B, \delta_M, F_{accept}, F_{reject})$.
We define an equivalent 2-PDA $P_M = (Q, q_0, \Sigma, \gamma, \delta_P, \Gamma, F)$
Here, γ is the stack alphabet, and for this construction $\gamma = \Sigma$.
Γ represents the bottom of the stack symbol, which we define as $\Gamma = B$.
q_0, Σ, Q of both P_M and M are same.
$F = F_{accept}$.
Acceptance for the 2- PDA is defined as being in a final state when all the input has been read.
δ_P takes as input the tuple $(q, a, \{s_1, s_2\})$ which indicates current state, input, and the tops of stack 1 and stack 2 respectively.
We intialise the stacks as follows: Stack 1 contains only the leftmost bit of input, and Stack 2 contains the rest of the input.
Hence, the input for the PDA always comes from the top of Stack 1. We define δ_P as follows:

- $\forall q \in F_{accept}$, create the transitions: $\delta_P(q, \epsilon, \{a, \epsilon\}) = (q, \{\epsilon, \epsilon\}) \forall a \in \Sigma$
  We also remove all other transitions from these states.
  Thus, on reaching an accepting state, the only path we can take is that we stay in this state reading all inputs leaving the stack unaffected. Thus, if we can reach one of these states, the input will be accepted.
- $\forall q \in F_{reject}$, create the transitions: $\delta_P(q, \epsilon, \{a, \epsilon\}) = (q, \{\epsilon, \epsilon\}) \forall a \in \Sigma$
  We also remove all other transitions from these states.
  Thus, if we ever reach these states, the only path we can take is that we stay in this state reading all inputs leaving the stack unaffected. Thus, if we can reach one

of these states, the input will be rejected.
Hence, if we reach this state, we end up in this state, and thus reject the input.

- For all transitions in δ_M, $\delta_M(q_i, a) = (q_j, b, L)$, we define the corresponding transitions in $\delta_P : \delta_P(q_i, \epsilon, \{a, \epsilon\}) = (q_j, \{\epsilon, b\})$. Thus, we pop the top of the stack 1 (which corresponds to the position of the head, and thus the input) and then push the output to stack 2.
- For all transitions in δ_M, $\delta_M(q_i, a) = (q_j, b, R)$, we define the corresponding transitions in $\delta_P : \delta_P(q_i, \epsilon, \{a, s_2\}) = (q_j, \{bs_2, \epsilon\}) \forall s_2 \in \Sigma$. Thus, we pop the top of the stack 1 (which corresponds to the position of the head, and thus the input), replace it by b, then further, we append to the stack 1 the top of stack 2 (s_2) (which initially contained the symbol at position $h_M + 1$ which is now the symbol at position of head as we moved head to the right). Thus, s_2 is at the top of stack 1.
- In case any of the transitions in the last two points includes reading $B(= \Gamma)$, we make sure to restore Γ in the stack. Thus, we can always keep reading Γ, and the stack is never empty, so we can move to the left or right infinitely.

Proof by Induction:
We use induction on size of the input, claiming that for any input, the path we take is the same as both machines, and the top of stack 1 represents the head of the tape of the Turing Machine while both machines run. Since accepting states of the 2-PDA are same as those of the Turing Machine, the acceptance criterion of both machines is identical.

- **Base Case:** Size of input = 0, that is, input = ϕ.
  Clearly, in both the 2-PDA and the Turing Machine, execution will end at the start state, q_0. In case $q_0 \in F_{accept}$, the input is accepted by both, else it rejected by both. The top of stack 1 contains Γ, and the head of the tape points to B. Since $B = \Gamma$, both are the same.
- **Inductive Hypothesis:** Let both machines have the same behaviour, that is, both machines follow the same transitions and path and the top of stack 1 represents the head of the tape of the Turing Machine while both machines run, for all input of size $\leq k$. We prove that both machines have similar behaviour for input of size $\leq k + 1$.
- **Inductive Step:** Consider any input $u \in \Sigma^*$, $|u| = k + 1$. Split this input into two parts: $u = u_0 x$ where $|u_0| = k$ and $x \in \Sigma$.
  Based on the Inductive Hypothesis, we know that after the input u_0, both machines will be at the same state. Let this state be q. Now, we analyse the following exhaustive cases:
  - $q \in F_{accept}$: In this case, the input is accepted by the Turing Machine, and execution ends at this state. In case of the 2-PDA, our construction ensures that on reading x, the next state is q itself irrespective of x. Thus, we stay on state q and the input u is accepted by the 2-PDA. Since the Turing Machine stops its run, the position of head is irrelevant.
  - $q \in F_{reject}$: In this case, the input is rejected by the Turing Machine, and execution ends at this state. In case of the 2-PDA, our construction ensures that on reading x, the next state is q itself irrespective of x. Thus, we stay on state q and the input u is rejected by the 2-PDA. Since the Turing Machine stops its run, the position of head is irrelevant.

- $q \in Q \setminus F_{accept} \cup F_{reject}$: Let us have the transition $\delta_M(q, x) = (q_i, b, L)$.Clearly, x is at the position of the head by our inductive hypothesis. By construction, we pop the head of stack 1 in this case, thus the top of stack 1 now contains the symbol to the left of the head, which is the new head as we move the head to the left. We also push b to the top of stack 2. We also have a transition to q_i, hence both machines end up at the same state and top of stack 1 contains the head.
  In case we have the transition $\delta_M(q, x) = (q_i, b, R)$, we by construction change the symbol at the current top of stack 1 and then push the top of stack 2 (which contained the symbol at the right of the current head) to the top of stack 1. Thus, the top of stack 1 now contains the symbol to the right of the original head, which is the new head. We also define a transition to q_i, hence both machines end up at the same state and top of stack 1 represents the head of the tape of the Turing machine.

Hence, by induction, both machines are identical.
Thus, any computable set can be accepted by a 2-PDA.

3 Question 3

Strategy:
We will show that a 4-counter TM is equivalent to a one-track one-head Turing Machine. Let us consider any computable (that is, decidable) set A. Since A is decidable, we know that there is some Turing Machine that accepts A. As proven in class, any variant of Turing Machines can be converted to an equivalent one-track one-head Turing Machine. Thus, we can convert the Turing Machine that accepts A to this equivalent one-track one-head Turing Machine.
Hence, any computable set A can be accepted by a one-track one-head Turing Machine. Thus, showing that a 4-counter TM is equivalent to a one-track one-head Turing Machine will be sufficient to prove that any computable set can be accepted by a counter TM. (As we can convert the one-track one-head Turing Machine that accepts the given computable set A to its equivalent counter TM).

Description:
Consider any one-track one-head Turing Machine, let it be M.
We can assume the alphabet of M to be binary. If it is not, we represent each letter of M's alphabet using some m ($m = \lceil \log_2 |\text{alphabet of } M| \rceil bits$), and perform m bit operations for each one operation on a letter of M, where each letter of M is encoded as a distinct ordering of m bits.
Let the leftmost bit in the tape of M be at index l_M, the rightmost bit be at index r_M and the head point to index h_M.
Then, we can use four counters to simulate this tape: one counter (C_1) stores all the bits from index l_M to h_M and one counter (C_2) stores all bits from index $h_M + 1$ to l_M. Counters C_3 and C_4 are counters that store temporary values to help in the functioning of counters C_1 and C_2.

This storing is done as follows:
C_1 stores the decimal number that is formed with the most significant bit as 1, the second most significant bit as the bit at index l_M, the third most significant bit as the bit at index $l_M + 1$, and so on till the least significant bit which is the bit at index h_M.
C_2 stores the decimal number that is formed with the most significant bit as 1, the second most significant bit as the bit at index r_M, the third most significant bit as the bit at index $r_M - 2$, and so on till the least significant bit which is the bit at index $h_M + 1$.
We keep the most significant bits of these counters as 1 so that we can distinguish between tape where there are no bits and where there are a string of 0s.
We must be able to implement the following operations on these counters to sufficiently simulate a one-track one-head Turing Machine:

- Copying one counter to another: Let us say we want to copy the value of C_1 to C_2. For this, we will need another counter C_3 to store temporary values. We first set C_3 and C_2 to 0 by decrementing them till they are 0.
  We perform a series of operations, where each operation has three steps: decrement C_1, increment C_2, increment C_3. We do this till C_1 contains 0. Thus, now, C_2 and C_3 contain the value that was originally in C_1.
  Now, we restore value of C_1 through another series of operations, each operation having two steps: decrement C_3, increment C_1. We do this till C_3 contains 0. Thus, now, C_1 is restored.
- Multiplying a counter by two: Let us say we want to multiply C_1 by two. For this, we will need another counter C_3 to store temporary values. We first set C_3 to 0 by decrementing its value till it is 0.
  We perform a series of operations, where each operation has three steps: decrement C_1, increment C_3, increment C_3. We do this till C_1 contains 0. Thus, now, C_3 contains double the value that was originally in C_1.
  Now, we copy value of C_3 to C_1. Thus, C_1 now has twice its original value.
- Dividing a counter by two and storing remainder: Let us say we want to divide C_1 by two, and store remainder in C_4. For this, we will need another counter C_3 to store temporary values. We first set C_3 and C_4 to 0 by decrementing their value till they are 0.
  We perform a series of operations, where each operation has three steps: decrement C_1, decrement C_1, increment C_3. We do this till C_1 contains 0 or 1. Thus, now, C_3 contains the result of dividing the value that was originally in C_1 by two.
  We now copy the value of C_1 (which currently has the remainder) to C_4.
  Now, we copy value of C_3 to C_1. Thus, C_1 now has the result, and C_4 has the remainder;
- Reading value of head: We seek to check the bit at position h_M, thus, we wish to find the LSB (least significant bit) of C_1. For, this we must find the remainder at counter C_1 when we divide by two. So, we first copy the value of C_1 to C_3. Thus, we perform a series of operations, where each operation has two steps: at each step, we decrement C_3 by 1 (so, each operation decrements the counter by 2). At the start of every operation, we check if the counter has value 1 or 0. If it has any of these values, we stop the execution of the operations.

In case the counter has value 1, its LSB = 1 and we read 1.
In case the counter has value 0, its LSB = 0 and we read 0.

- Writing value of head: To do this, we must change the LSB of C_1. We first read the value of the head, as described in the last point.
  If the value to write is same as the current value, we do nothing.
  If the value to write is 1 and the current value is 0, we increment C_1 by 1.
  If the value to write is 0 and the current value is 1, we decrement C_1 by 1.
- Moving head to the left: For this, we must multiply C_2 by two and divide C_1 by two (storing the remainder value in C_4). In case the value in C_4 is 1, we increment C_2 (this is to add the value of the original head to C_2).
  Pictorially, the operation is as follows: (The underline represents position of head)

  Tape:0110..00$\underline{1}$01..0101

  $C1$: 10110..001 $C2$: 11010..10

  Now, after changing position:

  Tape:0110..0$\underline{0}$101..0101

  $C1$: 10110..00 $C2$: 11010..101
- Moving head to the right: For this, we must multiply C_1 by two and divide C_2 by two (storing the remainder value in C_4). In case the value in C_4 is 1, we increment C_1 (this is to add the value of the new head to C_1)..
  Pictorially, the operation is as follows: (The underline represents position of head)

  Tape:0110..00$\underline{1}$01..0101

  $C1$: 10110..001 $C2$: 11010..10

  Now, after changing position:

  Tape:0110..001$\underline{0}$01..0101

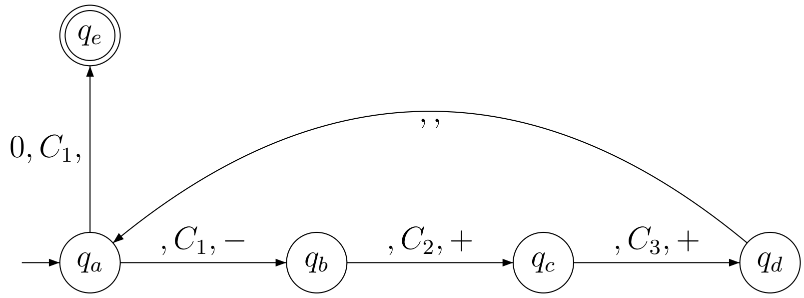
  $C1$: 10110..0010 $C2$: 11010..10

We always make sure that the minimum value contained in C_1 and C_2 at the end of an operation is 1 (that is, if the value of these are 0, increment and change to 1). This is as C_1 (C_2) takes the value 1 when the head moves to the left (right) till blank symbols of the tape. Since we restore the value, there are always more blank symbols on the extreme left and right simulating a tape of a Turing Machine.

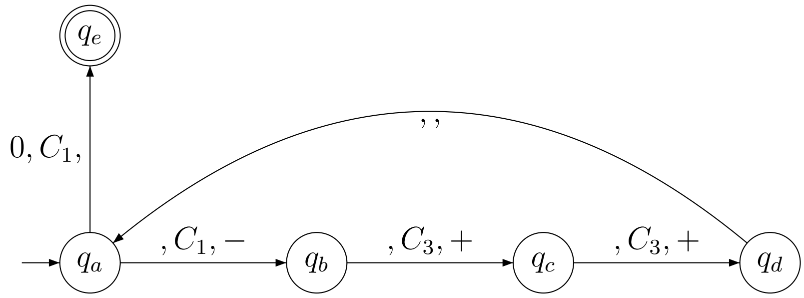
Construction:
Let M be a one-track one-head Turing Machine, defined as $M = (Q, q_0, \Sigma, B, \delta_M, F_{accept}, F_{reject})$.
We define an equivalent counter TM $M_C = (Q_C, q_{0C1}, \Sigma, B, \delta_C, F_{accept}, F_{reject})$.
$\Sigma, B, F_{accept}, F_{reject}$ all remain same for both Turing Machines. Since we need to define many new transitions, we require some more states ($Q \subset Q_C$), and use different transitions.
Define transitions using the following tuple: $(a, C_x, +/-)$: (value at counter, counter

number, increment/decrement).
If we are not allowed to read counters and can only read the head, we can always replace the head with the value to read temporarily using another counter to store the initial head and then restoring it.
We describe the implementation of each of the required operations, with arrows in showing the start of the operation and the final state showing completion:

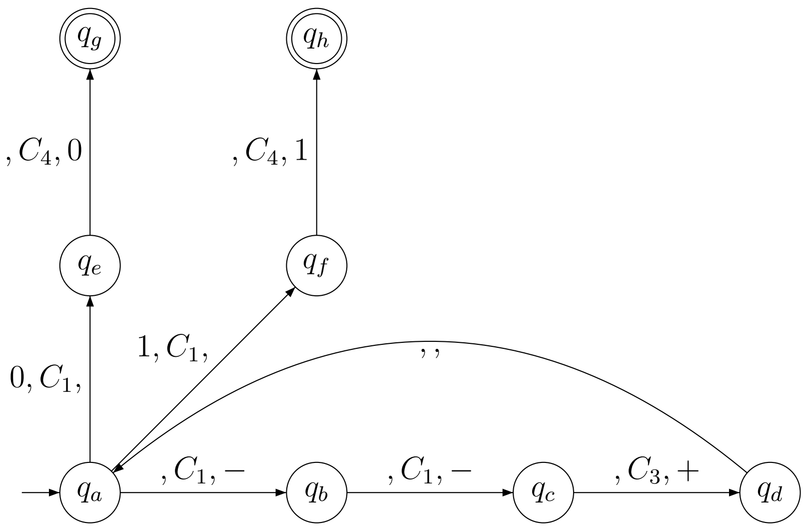
- Copying one counter to another: Copying from C_1 to C_2, assume C_3 and C_2 set to 0.



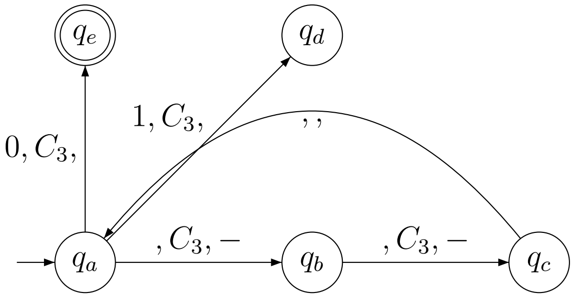
- Multiplying a counter by 2: Multiplying C_1, assume C_3 set to 0.



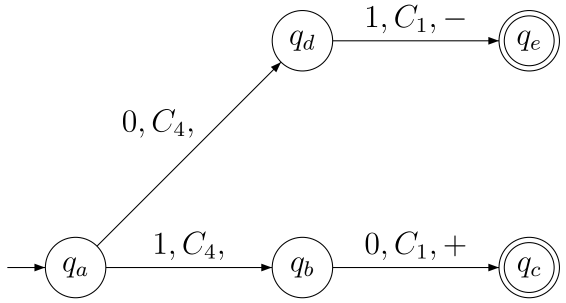
  Now, copy C_3 to C_1.
- Dividing one counter by two: Dividing C_1, assume C_3, C_4 set to 0, store remainder in C_4.



  Now, copy C_3 to C_1.
- Reading Head: Copy C_1 to C_3.



- Writing Head: Let the value of head be stored in C_4.



- Moving head to left and right can be done as mentioned in the description using the above implementations.
- If there is a transition $\delta_M(q, a) = (q_F, b, R)$: We perform two transitions in our 4-Counter TM: writing to the head and then shifting head to the right. In addition, we move to state q_F as we copy the state transitions from the one-track one-head TM. Thus, we end up at q_F with the modified head.
- If there is a transition $\delta_M(q, a) = (q_F, b, L)$: We perform two transitions in our 4-Counter TM: writing to the head and then shifting head to the left. In addition, we move to state q_F as we copy the state transitions from the one-track one-head TM. Thus, we end up at q_F with the modified head.
- Before reaching q_0, we first read all input and write it to the head at state q_{0C1}. Then, we move the head to the leftmost bit of the input at state q_{0C2}. After this, we move to state q_0. Thus, by the time we are at q_0, we have put all input in C_2 except for the head which is at C_1. (we define new states q_{0C1}, q_{0C2})

Proof by Induction:
We use induction on size of the input, claiming that for any input, we can achieve the same final states in the 4-Counter TM and the one-track one-head Turing Machine. Thus, both machines will have same acceptance criterion.

- Base Case: Size of input = 0, that is, input = ϕ.
  Clearly, in both the 4-Counter TM and the Turing Machine, execution will end at the start state, q_0. In case $q_0 \in F_{accept}$, the input is accepted by both, else it rejected by both.
- Inductive Hypothesis: Let both machines have the same behaviour, that is, both machines follow the same transitions and final state, for all input of size $\leq k$. We prove that both machines have similar behaviour for input of size $\leq k+1$.
- Inductive Step: Consider any input $u \in \Sigma^*$, $|u| = k+1$. Split this input into two parts: $u = u_0x$ where $|u_0| = k$ and $x \in \Sigma$. We only consider the cases where execution does not terminate after input u_0.
  Based on the Inductive Hypothesis, we know that after the input u_0, both machines

will be at the same (non-final) state. We let this state be q, thus, $q \in Q \backslash F_{accept} \cup F_{reject}$. If there is a transition $\delta_M(q,x) = (q_F, b, R)$, or $\delta_M(q,x) = (q_F, b, L)$, we move to q_F with the updated head as shown in the implementation. If $q_F \in F_{accept} \cup F_{reject}$, we will accept or reject the input based on the state, and terminate execution of the 4-Counter TM, identically as the one-track one-head TM. Thus, we end up at the same state with updated position of head for the input of size $k+1$.

Hence, by induction, both machines are identical.
Our machine works for a Turing Machine where we can write to the head, this includes the case where we have a read-only tape.
Thus, any computable set can be accepted by a 4-counter TM.

4 Question 4

We assume that the input in tape is formatted correctly as mentioned in the question (sequence of 1s followed by '$\times$' symbol followed by sequence of 1s followed by '=' symbol).
Let the Turing Machine that solves the problem be defined as $M = (Q, q_0, \Sigma, B, \delta_M, F_{accept}, F_{reject})$
The alphabet for this Turing Machine $\Sigma = \{0, 1, P, Q, \times, =\}$.
$F_{accept} = \{q_8\}$ and $F_{reject} = \phi$.

Let $X = 1^n$ and $Y = 1^m$, and the input be of the form '$X \times Y =$'.

Strategy:
The essence of the execution of the Turing Machine is as follows: for each bit of the first string (X) that we read, we append $|Y| = m$ number of 1s to the output.
Hence, the number of 1s finally will be $|X| \cdot |Y| = n \cdot m$. Thus, the output will be 1^{nm}.

Description:
We define some special symbols to aid in this execution. Their description is as follows:

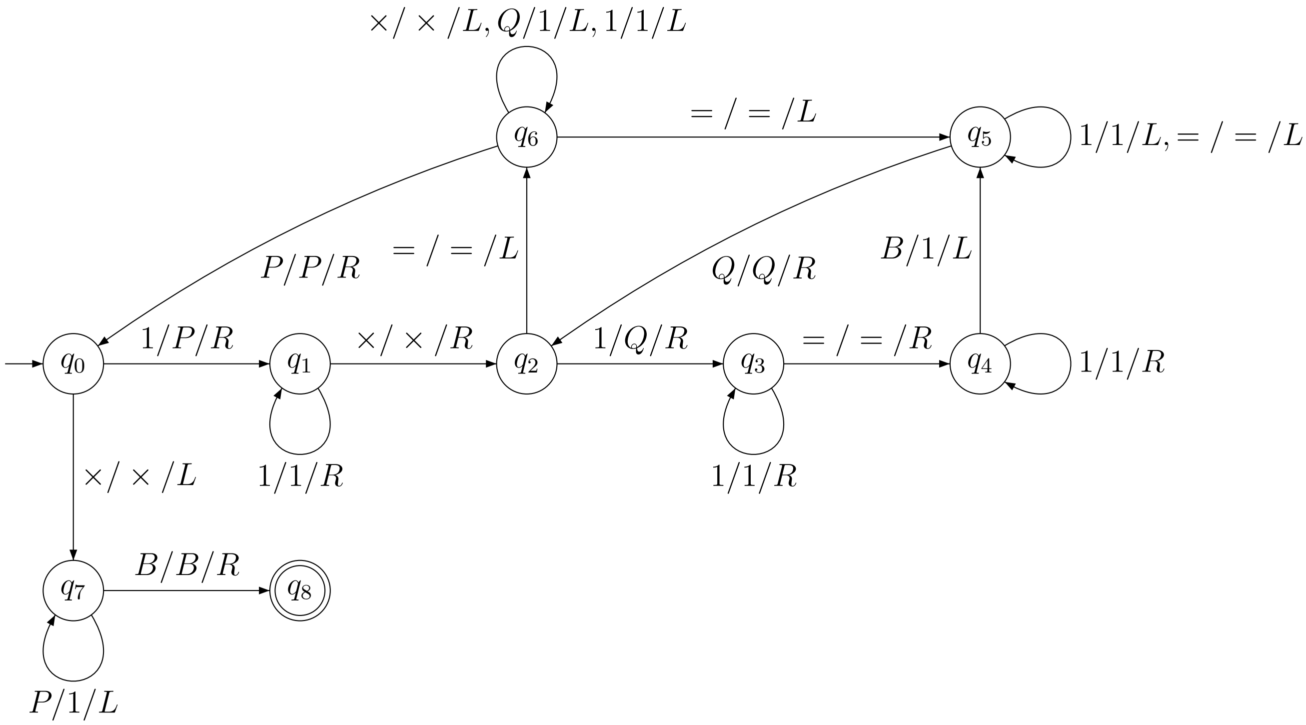
- P: This symbol replaces a bit in X, indicating we have read this bit.
- Q: This symbol replaces a bit in Y, indicating we have read this bit. Since we read all bits of Y once for every read of one bit in X, we reset these symbols to 1 after we have read a bit of X and have read Y entirely once.

The execution of the Turing Machine proceeds as follows:

1. The head is initially placed at the leftmost bit of the input, which will be the first 1 bit of X.
2. We change this bit to symbol P. Following this, we move our head to the right till we encounter the symbol '$\times$'.
3. We continue to move the head towards the right till we find the first 1 bit (of Y).
4. Then, we change this bit to Q and move the head towards the right till the rightmost non-blank symbol (that is, the rightmost bit of the current output). (Initially, this will be the '=' symbol.)
5. We move the head one position towards the right and write 1 at this position (thus appending 1 to the output).

6. Then, we move the heads leftwards till we encounter the leftmost 1 in Y.
7. Now, we repeat steps 4-6 til Y contains any 1s.
8. This continues till all the 1s in Y have been converted to Qs, in which case we do not encounter any 1 in Y.
9. If there are no 1s left in Y, our head will currently be at the '=' symbol. Now, we move leftwards till the leftmost Q of Y, changing all Qs of Y to 1s at every left step of the head.
10. Now, we move the head leftwards till we encounter the leftmost 1 of X.
11. Now, we repeat steps 2 - 10 till there are any 1s in X.
12. If there are no 1s left in X, our head will currently be at the '×' symbol. Now, we move leftwards till the leftmost P of X, changing all Ps of X to 1s at every left step of the head.
13. Thus, the contents of X and Y have been restored and we have printed the required output. Now, we transition to the accepting final state.

State Diagram:



5 Question 5

Let the Turing Machine that solves the problem be defined as $M = (Q, q_0, \Sigma, B, \delta_M, F_{accept}, F_{reject})$
The alphabet for this Turing Machine $\Sigma = \{a, b, \underline{a}, \underline{b}, \overline{a}, \overline{b}, p, q, c\}$.
$F_{accept} = \{q_{15}\}$ and $F_{reject} = \{q_{13}, q_{14}\}$ as shown in the state diagram.
State q_{13} will reject strings of odd length and q_{14} will reject strings of even length but not of the type ww.

Let the input be some string $u \in \{a, b\}^*$, of length n.
In case u is of the form ww for some $w \in \{a, b\}^*$, we can say that w is of length $\frac{n}{2}$. Thus, for u to be of this form, the symbols at indices i and $\frac{n}{2} + i$ must be the same.

Strategy:
The essence of the execution of the Turing Machine is as follows: We initially check for even length, as inputs of odd length cannot be of the given form.
If the length of the string is even, mark the left and right halves of the input separately using special symbols ($\underline{a}$,$\underline{b}$) in the left half, ($\overline{a}$,$\overline{b}$) in the right half) to provide the machine a way to distinguish between halves, and then check if the corresponding symbols in the left and right halves match (that is, if the symbol at index i matches the symbol at index $\frac{n}{2} + i$). At any point, if we find that the corresponding symbols don't match, we reject the input.

Description:
Description of the new symbols defined is as follows:

- $\underline{a}$: marked symbol a in the left half.
- $\underline{b}$: marked symbol b in the left half.
- $\overline{a}$: marked symbol a in the right half.
- $\overline{b}$: marked symbol b in the right half.
- p represents any symbol in the set $\{\underline{a}, \overline{a}, \underline{b}, \overline{b}\}$
- q represents any symbol in the set $\{a, \underline{a}, \overline{a}, b, \underline{b}, \overline{b}\}$
- r represent any symbol in the set $\{a, \underline{a}, b, \underline{b}\}$
- c represents a another symbol which helps in the execution of the above strategy

The steps of execution of the Turing Machine are as follows:

1. Firstly, we check if the length of the input is even, by traversing through the input (moving the head rightwards) and transitioning between two states and every step, one of the states indicating if the length of the input is even and the other indicating if the length is odd.
2. If on reading all the input we end up at the state indicating odd length of input, we reject the input.
3. Else, we first move the head leftwards to the leftmost symbol of the input. Following this, we start to replace the original unmarked symbols with their marked counterparts. To do this, we start at the first unmarked symbol, which will be at the leftmost position in the input.
4. We change this symbol to include an underline ($a \to \underline{a}, b \to \underline{b},$) and then move the head rightwards to the rightmost unmarked symbol.
5. Then, we change this symbol to include an overline ($a \to \overline{a}, b \to \overline{b},$).
6. Thus, we have marked the leftmost and rightmost unmarked symbols.
7. Now, we repeat steps 4 - 6 till there are no more unmarked symbols left.
8. When all symbols have been marked, clearly all the symbols of the left half are underline-marked and symbols of the right half are overline-marked, as we mark an

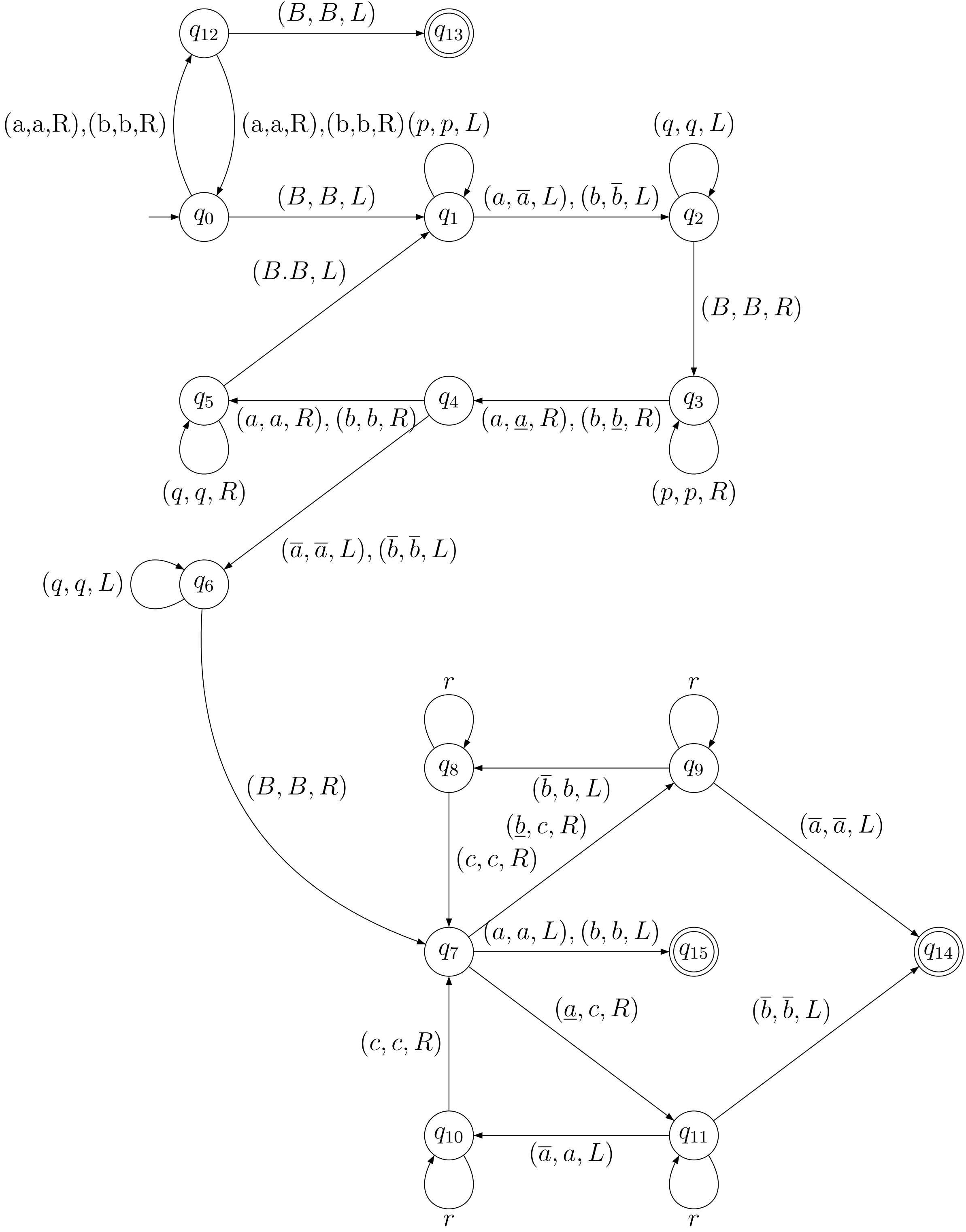
equal number of symbols with an overline and underline, marking underlines from the left and overlines from the right.

9. At this point, the head is at the leftmost symbol of the right half of the input. From here, we move the head leftwards till the leftmost symbol of the input.
10. Now, we start checking if the input is of the desired form. We start at the leftmost underline-marked symbol, which will be at the leftmost position in the input.
11. On reading this symbol, we execute one of two cases depending on whether the symbol is $\underline{a}$ or $\underline{b}$. In both cases, we change the read symbol with the symbol c and move the head till the leftmost overline-marked (right half) symbol.
12. In case we had read the symbol $\underline{a}$, we check if the overline-marked symbol is $\overline{a}$, and we check for $\overline{b}$ in case we have read $\underline{b}$. If we do not pass this check, we immediately reject the input. If we pass the check, we unmark the read symbol.
13. On passing the check, we can say that the symbol in the left half matches its corresponding symbol in the right half. This is as in the first execution of this check we check the symbol at the leftmost position (index 1) and the symbol at the leftmost position of the right half (index $\frac{n}{2} + 1$). After every execution, we move one symbol ahead in both halves, and thus we check matching of symbols at indices i and $\frac{n}{2} + i$ for some i. Since we only move one symbol ahead in both halves in one execution, our check is exhaustive, that is, we check for every symbol in both halves.
14. Repeat steps 11 - 13 this till there are no more marked symbols (or the input is rejected, in which case execution ends).
15. If all the symbols are unmarked, then we have matched every underline-marked (left half) symbol with a overline-marked (right half) symbol. Hence, the input is of the desired form. Thus, we move to a final accepting state and end execution.

For example, the tape at an intermediate step (during marking) could look like:

$$\ldots \underline{ab}baab\overline{b}\overline{a} \ldots$$

State Diagram:


(B, B, L)
q_{12}
q_{13}
(a,a,R),(b,b,R)
(a,a,R),(b,b,R)
(p, p, L)
(q, q, L)
q_0
(B, B, L)
q_1
$(a, \overline{a}, L), (b, \overline{b}, L)$
q_2
$(B.B, L)$
(B, B, R)
q_5
q_4
q_3
$(a, a, R), (b, b, R)$
$(a, \underline{a}, R), (b, \underline{b}, R)$
(q, q, R)
(p, p, R)
$(\overline{a}, \overline{a}, L), (\overline{b}, \overline{b}, L)$
(q, q, L)
q_6
r
r
q_8
q_9
(B, B, R)
$(\overline{b}, b, L)$
$(\underline{b}, c, R)$
$(\overline{a}, \overline{a}, L)$
(c, c, R)
$(a, a, L), (b, b, L)$
q_7
q_{15}
q_{14}
$(\underline{a}, c, R)$
$(\overline{b}, \overline{b}, L)$
(c, c, R)
q_{10}
q_{11}
$(\overline{a}, a, L)$
r
r

6 Question 6

Question: Prove that the following set is not computable:

$$L = \{(p,q)| \text{ there exists a string } x \text{ accepted by both TM } M_p \text{ and TM } M_q \}$$

Strategy:

We will prove the given result by contradiction. We will assume that the set L is computable which means there exists a Turing Machine M_{decide} that accepts L. Using M_{decide} we will try to solve the Halting Problem, i.e, we will construct a Turing Machine that accepts the Halting Set. Since Halting set is undecidable, this will lead to a contradiction proving that L is undecidable.

Solution:

Let us assume the set L is computable. This implies that there exists a halting Turing Machine M_{decide} that accepts the set L.

M_{decide}: In words, given an input (α, β) to the TM M_{decide}, M_{decide} accepts the given input if there exists a string z which is accepted by both TM M_α and TM M_β. M_{decide} rejects the given input if there is no string accepted by both TM M_α and TM M_β where α and β are descriptions of Turing Machines M_α and M_β respectively.

Construction of a halting Turing Machine M_s that aims to solve the Halting Problem:

Aim of M_s: Given an input (p, x) where p is the description of the Turing Machine M_p, M_s aims to decide that whether M_p halts on x or not.

Description of M_s:

1. Given an input (p, x), construct description of a Turing machine, call it as r which works as follows:
   On an input y to M_r, ignore y and simulate M_p on x. If M_p halts on x then accept y, else reject y.
2. Run M_{decide} on the input (r, r) and accept (p, x) iff M_{decide} accepts (r, r).

Set Accepted by M_r: The set accepted by the TM M_r is independent of the input given to it because of the description of M_r as mentioned above. There are only two possibilities:

1. M_p halts on x: In this case M_r will accept all the inputs hence set accepted by M_r is $\sum^*$ where $\sum$ denotes the alphabet taken into consideration.
2. M_p does not halt on x: In this case M_r won't accept any input and hence set accepted by M_r is ϕ.

Thus, if M_r accepts any string, it accepts all strings $\in \sum^*$.

M_s **is Halting**:

Since M_s only simulates M_{decide}, and M_{decide} is halting by our assumption, we can say that M_s is halting.

Explanation that M_s solves the Halting Problem, i.e, accepts the Halting set:

- M_s accepts $(p, x) \iff M_{decide}$ accepts (r, r)
- M_{decide} accepts $(r, r) \iff$ there exists a string z which is accepted by the TM M_r and the TM $M_r \iff$ there exists a string z which is accepted by the TM $M_r \iff$ set accepted by M_r is $\sum^*$ (as proved above) where $\sum$ denotes the alphabet taken into consideration $\iff M_p$ halts on x.
- Hence, M_s accepts $(p, x) \iff M_p$ halts on x.
- Since M_s is halting, M_s rejects $(p, x) \iff M_p$ does not halt on x.
- M_s solves the Halting Problem which is **not Possible**.

Hence, by contradiction, there does not exist any halting TM that accepts the set L and therefore L is not computable.

$\rightarrow$ Hence Proved.