CS340 Assignment - 2

Church of Turing

Soham Samaddar
200990

Aditya Tanwar
200057

Samarth Arora
200849

September 2022

Q1. For a finite automata $\mathcal{F} = (Q, q_0, \Sigma, \delta, F)$ and input x, define a *configuration* of $\mathcal{F}$ to be the pair $\langle q, y\rangle$ where $q \in Q$, y is a suffix of x, and automata $\mathcal{F}$ on input $x = zy$ ends up in state q after reading z. Note that if $\mathcal{F}$ 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 $\langle q_0, y_0\rangle\langle q_1, y_1\rangle \ldots \langle q_m, \rangle$ where (1) each $\langle q_i, y_i\rangle$ is a configuration, (2) $\mathcal{F}$ moves from configuration $\langle q_i, y_i\rangle$ to $\langle q_{i+1}, y_{i+1}\rangle$ in one step, (3) $q_m \in F$, and (4) $y_0 = x$.

Prove that

- Input x is accepted by $\mathcal{F}$ iff there exists an accepting configuration sequence that starts with $\langle q_0, x\rangle$.
- The set of all accepting configuration sequences of $\mathcal{F}$ is computable but is not a CFL.

Solution: For the sake of clarity, call the first element of the configuration pair the "state", and the second element the "suffix string" of the configuration pair.

Definition. For two strings s_1, s_2 where s_2 is a suffix of s_1, let s_1/s_2 denote the string that is obtained by removing s_2 from the right end of s_1. More formally, $s_1 = cs_2$, where c is the additional prefix, then $s_1/s_2 = c$.

First we establish a lemma:

Lemma. In an accepting configuration sequence, $\langle q_0, y_0\rangle\langle q_1, y_1\rangle \ldots \langle q_m, \rangle$, for all i with $0 \le i < m$, we have $|y_{i+1}| \le |y_i| \le |y_{i+1}| + 1$ where $|y|$ denotes the length of the string y. Additionally, y_{i+1} is a suffix of y_i, with y_i/y_{i+1} being either ϵ or a character $c \in \Sigma$.

Proof. Consider two adjacent configurations $\langle q_i, y_i\rangle$ and $\langle q_{i+1}, y_{i+1}\rangle$. Since q_i and q_{i+1} are adjacent states, there can be only two possibilities:

- The transition occurred on reading a character c from the input. Hence, $y_i = c\, y_{i+1}$ whereby the inequality follows. Also, the suffix condition follows.
- The transition was an ϵ transition. Hence $y_i = \epsilon\, y_{i+1} = y_{i+1}$. Again, the inequality follows, as well as the suffix condition □

We show first that if x is accepted by $\mathcal{F}$ then there exists an accepting configuration sequence. Interpret the output of δ as a set, even if $\mathcal{F}$ is a DFA. The idea is to essentially backtrack the run of $\mathcal{F}$ on the input string x, and check if the starting state was reachable. Given a run of (possibly NFA) $\mathcal{F}$ on x, we give an algorithm, to generate an accepting configuration sequence:

Step 1. Initialize p to be q_m, where q_m is one of the accepting states that $\mathcal{F}$ reaches on completely reading x. Initialize y to be empty and ξ to be an empty configuration sequence.

Step 2. If $y = x$, we are done.

Step 3. Else, prepend $\langle p, y\rangle$ to ξ. Now, p must have been reachable from some other state in the run of $\mathcal{F}$, so there must exist a transition of the form $\delta(q, c) = p$, where q was reached in a step precisely before.

Step 4. Make the updates: $y \leftarrow c\,y$ and $p \leftarrow q$.

Step 5. Repeat from Step 2.

Proof of termination: The run of $\mathcal{F}$ must have been finite, and with each iteration of the algorithm, we move one step closer to the start state, as well as towards recovering x (by constructing y). By one step, we mean we move closer by adding one configuration to the accepting configuration sequence. Thus, after a finite number of iterations, the algorithm must terminate.

Now, we show that if an accepting configuration for a string x exists then $\mathcal{F}$ must accept it, thus completing the two sides of our proof. By the definition of an accepting state, $\mathcal{F}$ moves from the i^{th} state to the $(i+1)^{th}$ state in one step, this is to say that there exists a transition of the form $\delta(q_i, y_i/y_{i+1}) = q_{i+1}$ (by this *lemma*) for all $0 \leq i < m$. Using this fact, we use induction to show that there exists a run of $\mathcal{F}$ on input x, in which each of the states are reached exactly in the order prescribed by the sequence.

* **Inductive Hypothesis**: In an accepting configuration sequence, the configuration $\langle q_{i+1}, y_{i+1}\rangle$ is reached in some run of $\mathcal{F}$ on input x only if there exists a run in which $\langle q_i, y_i\rangle$ is reached.

* **Base Case**: Showing it for $i = 0$ is trivial, since $\mathcal{F}$ starts on q_0 and we can simply set $y_0 = x$.

* **Inductive Step**: Let $\mathcal{R}$ be the run on which configuration i is reached. We showed above that there must exist a transition of the form $\delta(q_i, y_i/y_{i+1}) = q_{i+1}$ (in an accepting configuration sequence). Thus, the state $\langle q_{i+1}, y_{i+1}\rangle$ is reachable from the state $\langle q_i, y_i\rangle$ (and consequently, $\langle q_0, y_0\rangle$).

With this inductive proof, it becomes apparent that given an accepting configuration sequence, it is possible for $\mathcal{F}$ to reach an accepting state (correspondingly $\langle q_m, \rangle$) on completely exhausting the initial input x (correspondingly $\langle q_0, y_0 = x\rangle$). Thus, $\mathcal{F}$ accepts x if there exists an accepting configuration sequence.
This completes our proof for the first part of the question.

To show that the set of accepting configuration sequences of $\mathcal{F}$ is computable, we construct a TM $M_{\mathcal{F}}$, with a description of $\mathcal{F}$ hardcoded in it. Let us look at the alphabet over which we could define the strings of accepting configuration sequences. Clearly, the following alphabet

$$\Sigma_{A_{\mathcal{F}}} = \{\langle\} \cup \{\rangle\} \cup \{,\} \cup \Sigma \cup Q$$

works.

The three characters "$\langle$", "$\rangle$" and "," are used to properly delimit and format the accepting configuration sequence. Each of the states are a distinct character to denote the state of the configuration. The alphabet Σ is used to denote the suffix string of the configuration. Now the TM does the following on receiving the configuration sequence $\langle q_0, y_0 \rangle \langle q_1, y_1 \rangle \ldots \langle q_m, \rangle$:

Step 1. Ensure that the input is in the correct format. The correct format of the string is ensured by checking whether each "$\langle$" is followed by a state, followed by a ",", followed by a suffix string followed by "$\rangle$". This pattern should repeat throughout.

Step 2. Keep a special cell somewhere on the tape (so that it does not intrude with the working) which will be used to store the transition character between states. Call this cell, the *transition character cell.*

Step 3. During the process, we cross off configurations. Move to the leftmost configuration which has not yet been crossed out (initially we start with $\langle q_0, y_0 \rangle$). Let this configuration be $\langle q_i, y_i \rangle$.

Step 4. Check if $\langle q_i, y_i \rangle$ is the final configuration in the sequence. If it is the final configuration, move to the hardcoded description of $\mathcal{F}$ and check if q_i is a final state (which can be done by a linear search over all the final states). If it is not a final state, reject. Otherwise check if y_i is an empty string. If it is an empty string, accept, otherwise reject.

Step 5. Otherwise, we compare $\langle q_i, y_i \rangle$ and $\langle q_{i+1}, y_{i+1} \rangle$. First we compare y_i and y_{i+1}. We outline the process of comparison: start from the right end of y_i and compare with the right end of y_{i+1}. If they are equal, cross them off. Otherwise, reject the input. Continue this till one of two situations arise:

1. All the characters of both string y_i and y_{i+1} have been crossed off. In such a case, store ϵ in the *transition character cell.*
2. y_i has exactly one extra character compared to y_{i+1}. Then store this extra character in the *transition character cell.*

In all other situations, reject the input.

Step 6. Now, go to the hardcoded description of $\mathcal{F}$ and find whether there exists a transition from q_i to q_{i+1} (which can be done by a linear search across all the transitions) using the character stored in the *transition character cell.* If such a transition exists, cross off the configuration $\langle q_i, y_i \rangle$ and continue from **Step 3**. Otherwise reject the input.

The above construction works since analyzing two consecutive configurations in the accepting configuration sequence gives us the precise information about the transition that was taken. More formally, if $\langle q_i, y_i \rangle$ and $\langle q_{i+1}, y_{i+1} \rangle$ are two consecutive configurations in the accepting configuration sequence, then the transition that was taken is $\delta(q_i, y_i / y_{i+1}) = q_{i+1}$ (follows from this lemma).

As for showing that the set of all accepting configuration sequences is not a CFL, we give a counter-example to the opposite claim.

Consider the regular language $L = \{0^n : n \in \mathbb{N}\}$ over the alphabet $\Sigma = \{0, 1\}$. Construct a **DFA** $\mathcal{F}$ for this language (which can be done since it is easy to see that L is a regular language). Now, for the sake of contradiction, assume that the set of accepting configuration sequences of $\mathcal{F}$ (denoted by $A_{\mathcal{F}}$) is a CFL. The CFL will also use the same alphabet defined above $\Sigma_{A_{\mathcal{F}}}$.

Now, we state a lemma characterizing the suffix string of the configuration.

Lemma. In an accepting configuration sequence **for a DFA**, $\langle q_0, y_0\rangle\langle q_1, y_1\rangle \ldots \langle q_m, \rangle$, for all i with $0 \leq i < m$, we have $|y_{i+1}| + 1 = |y_i|$ where $|y|$ denotes the length of the string y.

Proof. Consider two adjacent configurations $\langle q_i, y_i\rangle$ and $\langle q_{i+1}, y_{i+1}\rangle$. Since q_i and q_{i+1} are adjacent states and the automata is a DFA, the transition necessarily occurred on reading a character c from the input. Hence, $y_i = c\, y_{i+1}$ whereby the equality follows. □

As a consequence of the above lemma, we get the following corollary.

Corollary. The accepting configuration sequence of 0^n contains exactly $(n + 1)$ configurations. Since the suffix string of the starting configuration has length n and the ending configuration has suffix string of length 0, and the length of the suffix string decreases by 1 after each step.

Now we use the contrapositive form of the pumping lemma for CFLs to prove that $A_{\mathcal{F}}$ is not a CFL.

Let the demon choose $n \in \mathbb{N}$. We choose the accepting configuration sequence of 0^n. By the above corollary, the sequence must have exactly $(n + 1)$ configurations. Since each configuration has at least length 3 (which simply comes from the delimiting characters), so the accepting configuration sequence string has length greater than $3n > n$.

Now the demon splits the string into $xuyvz$ with $uv \neq \epsilon$ and $uyv \leq n$. Now we split the analysis into various cases.

Case 1: *(Brackets balanced)* Both the substrings u and v are of the form $\langle s\rangle$ where $s \in \Sigma_{A_{\mathcal{F}}}$ or the substring is ϵ. Now both u and v cannot be ϵ by the choice made by the demon. So without loss of generality, let u be non empty. Then $u = \langle q_i, y_i\rangle\langle q_{i+1}, y_{i+1}\rangle \ldots \langle q_{i+l}, y_{i+l}\rangle$ for some $l \geq 0$. By continuously applying the lemma, we can get $|y_i| = |y_{i+l}| + l$ Now if we pump u, and if $u^2 = \langle q_i, y_i\rangle\langle q_{i+1}, y_{i+1}\rangle \ldots \langle q_{i+l}, y_{i+l}\rangle\langle q_i, y_i\rangle\langle q_{i+1}, y_{i+1}\rangle \ldots \langle q_{i+l}, y_{i+l}\rangle$. If $xu^2yv^2z \in A_{\mathcal{F}}$, then by the lemma, it must be that $|y_{i+l}| = |y_i| + 1$ since the two configurations appear consecutively. But then, $|y_i| = |y_i| + 2$, a contradiction.

Case 2: *(Brackets balanced)* One of the substrings (without loss of generality u) is of the form $0^{n_1}\rangle \ldots \langle 0^{n_2}$. Then pumping u twice violates *the corollary*. To see this, u^3 would look something like this:

$$0^{n_1}\rangle \ldots \langle q_i, \underline{0^{n_2+n_1}}\rangle \ldots \langle q_j, \underline{0^{n_2+n_1}}\rangle \ldots \langle 0^{n_2}$$

Observing the underlined terms, we can see that two substrings $y_i = 0^{n_1+n_2}$ and $y_j = 0^{n_1+n_2}$ have the same length, but the corollary dictated that the length should have decreased. We have reached a contradiction in this case.

Case 3: *(Brackets unbalanced)* One of the substrings (without loss of generality u) is of the form $\rangle \ldots$ (i.e., the string does not have the opening brackets balanced with the closing brackets). Pumping u once would violate the input convention altogether, since the opening brackets would be one more in count than the closing brackets. Since, the string would not even remain a configuration sequence, it could not possibly be an accepting string. So, pumping lemma does not hold in this case.

Case 4: *(Brackets unbalanced)* One of the substrings (without loss of generality u) is of the form $\ldots \langle$ (i.e., the string does not have the opening brackets balanced with the closing brackets). Pumping u once would violate the input convention altogether, since the opening brackets would be one less in count than the closing brackets. Since, the string would not even remain a configuration sequence, it could not possibly be an accepting string. So, pumping lemma does not hold in this case.

Since the pumping lemma for CFLs does not hold for the set of accepting configuration sequences of $\mathcal{F}$, the set is not a CFL. The set was, however, shown to be computable above.

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*.

Solution: Given any computable set A, let M denote the halting TM which computes A. The TM model we use a single tape, single read-write head, finite left tape end and infinite right tape end. We construct a 2-PDA (P) from M, which is able to simulate precisely M, showing that a 2-PDA is able to at least accept any computable set. The idea is to essentially split the tape of the TM into two parts. The first part of the tape is from the left end of the tape till the cell just before the read-write head, while the second part of the tape is the cell starting from the read-write head, and continues infinitely to the right. We allow one stack to store the first part of the tape and the other stack to store the other part of the tape.

Let ζ_1 and ζ_2 denote the two stacks. ζ_1 will store the contents of the cells to the left of the read-write head in the TM while ζ_2 stores the contents to the right. Initially, the input is written on the tape of the TM and the read-write head points to the start of the input. So we need to setup this configuration in P before simulating M. In informal terms, this can be done by reading the entire input and pushing whatever we read onto ζ_1. Once the entire input has been read, we push everything from ζ_1 to ζ_2 which reverses the input. Now this is precisely the setup that we want. After the setup, we simulate M. Let us write down the mathematical details now.

Let $P = (Q, \Sigma, \Gamma, \delta, s, \perp, f)$ where:

Q: Set of states

Σ: The input alphabet

Γ: The stack alphabet

δ: The transition function

s: The starting state

$\perp$: The bottom of both the stacks

f: The final state

The set of states $Q = \{q_1, q_2\} \bigcup Q_M$ where Q_M denotes the states of M and $q_1, q_2 \notin Q_M$. The stack alphabet will be equal to the tape alphabet of M and an additional bottom of stack symbol $\perp$. The starting state is nothing but q_1. The final state will be the accepting state of M.

The transition function will be:

$$\delta : Q \times \Sigma \times \Gamma \times \Gamma \to Q \times \Gamma^* \times \Gamma^*$$

where the transition $\delta(q, a, r_1, r_2) = (p, W_1, W_2)$ denotes that when the control is in state q, reads a character a from the input, pops a character r_1 from ζ_1 and pops a character r_2 from ζ_2, the control transitions to state p, pushes the string W_1 onto ζ_1 and pushes the string W_2 onto ζ_2. The strings W_1 and W_2 are pushed on the stack from **left-to-right**. Note that each of a, r_1, r_2, W_1 and W_2 can be ϵ, where we do not read, pop or push from the input/stacks as the case may be.

Setup: We perform this using three states q_1, q_2 and q_3. $q_1 = s$ will be the start state of P. q_3 will be the start state of M. The transitions associated with this setup are:

(a) $\delta(q_1, a, \epsilon, \epsilon) = (q_1, a, \epsilon)$ - Read the input and push it into ζ_1

(b) $\delta(q_1, \epsilon, \epsilon, \epsilon) = (q_2, \epsilon, \epsilon)$ - Non deterministically guess the end of the input

(c) $\delta(q_2, \epsilon, a, \epsilon) = (q_2, \epsilon, a)$ - Pop the top of ζ_1 and push it into ζ_2

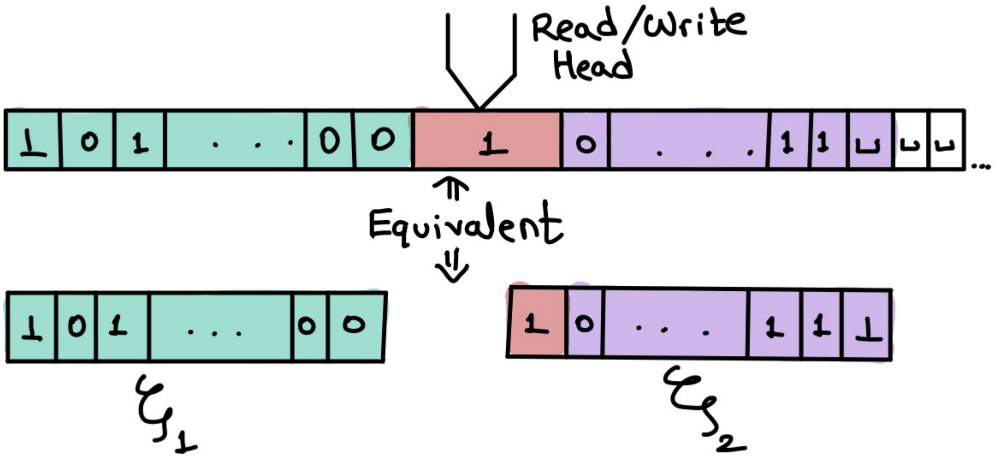
(d) $\delta(q_2, \epsilon, \perp, \epsilon) = (q_3, \perp, \epsilon)$ - Once the bottom of ζ_1 has been reached, move to q_3 and start simulating M

By non determinism, we might shift to q_3 and start simulating M before we even parse the entire input. This can lead to extra strings being accepted by P. But, once we move from state q_1, we will never read another symbol of the input again. So, even if we end up in f in P, P will only accept the input string if the entire input had been exhausted. So we do not need to worry about incomplete input simulations getting accepted by P.

Simulation: Currently, the control of P is on state q_3. This state is actually the start state of M. Now, for any left transition $\delta_M(p, a) = (q, b, L)$ (where δ_M is the transition function of M), add the transition $\delta(p, \epsilon, c, a) = (q, \epsilon, bc)$ to P. For any right transition $\delta_M(p, a) = (q, b, R)$, add the transition $\delta(p, \epsilon, \epsilon, a) = (q, b, \epsilon)$. Finally, the only final state of P is equal to the accepting state of M, that is $f = t$.

This construction works since we replicate the exact configuration of M for every transition. We provide the proof of this replication by inducting on the number of steps taken by M in any particular computation and showing that the configuration remains invariant.

Inductive Hypothesis: At the end of each step of M, ζ_1 stores the contents to the left of the head (with the leftmost cell's content at the tail of the stack), while ζ_2 stores the contents to the right, and the read-write head of M corresponds to the head of the stack ζ_2. The finite control of P is in the same state as the finite control of M. The following diagram denotes the equivalence we are setting up.



Base Case : We consider the base case as the situation when we start simulating M. At that point of time, the entire input is present in ζ_2 in a left-to-right manner, while ζ_1 only contains $\perp$. This is exactly the configuration of M at the start of computation by M. The entire input is written on the tape. The read-write head of M points to the first character of the input. Only the left-end marker is on the left side of the read-write head. M is in the start state while P is in the state q_3 which we have defined to be equal. Hence the base case is true.

Inductive Step : Suppose after k steps, the configuration is satisfied. Let the left stack contain the string S_1 and the right stack contain the string S_2. Hence, the tape of M currently stores $S_1S_2\sqcup^*$ and the read-write head points to the first character in S_2. Both P and M are in the same state p. Now, we split the next step into two cases:

Case 1: M moves to the left for the $(k+1)^{th}$ step. Let a and c denote the first character of S_2 and the last character of S_1 respectively. Rewrite S_1 and S_2 as $S_1 = S_1'c$ and $S_2 = aS_2'$. Hence, the transition looks like $\delta_M(p, a) = (q, b, L)$. After the transition, the tape points to c and the character a of S_2 was replaced by b. By construction, P has a transition of the form $\delta(p, \epsilon, c, a) = (q, \epsilon, bc)$ since the head of ζ_1 contains c, the head of ζ_2 contains a and the finite control is in state p. Now, it pops both c and a, moves to state q and pushes b and then c into ζ_2. So, ζ_1 now contains S_1' while ζ_2 contains cbS_2'. This is precisely what the current configuration of the tape is in M. Also, the finite controls of both P and M are in the same state. Hence proved.

Case 2: M moves to the right for the $(k+1)^{th}$ step. Let a, c denote the first and second characters of S_2. Rewrite $S_2 = acS_2'$. Hence, the transition looks like $\delta_M(p, a) = (q, b, R)$. After the transition, the tape points to c and the character a of S_2 was replaced by b. By construction, P has a transition of the form $\delta(p, \epsilon, \epsilon, a) = (q, b, \epsilon)$ since we do not care about the head of ζ_1, the head of ζ_2 contains a and the finite control is in state p. Now, it pops a, moves to state q and pushes b and into ζ_1. So, ζ_1 now contains $S_1 b$ while ζ_2 contains cS_2'. This is precisely what the current configuration of the tape is in M. Also, the finite controls of both P and M are in the same state. Hence proved.

We are done by induction. Hence P perfectly simulates M. If M accepts a string, it moves to state t and halts. P would also move to $t = f$ and since the entire input has been read, it accepts the string. If M rejects a string, P would also enter a non-final state and halt since no transitions are defined once M enters the rejecting state and hence P also has no transitions from that state. If M gets stuck in a loop, so does P. Hence proved.

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: We have already shown how a Turing Machine M can be simulated by two stacks in Q2. We build on the result and show how a single stack can be simulated by a finite number of counters. Consequentially, it shall follow that a Turing Machine can be simulated by a finite number of counters.
Preliminaries: We assume that the stack only has characters $\{0, 1\}$ (other than the bottom of stack symbol $\perp$). This assumption is justified since any arbitrary alphabet can be encoded in binary to realize an alphabet with characters only in $\{0, 1\}$ and then further, an insertion can be broken down into multiple smaller insertions, each catering to a single bit. Similarly, for popping.

With the preliminaries out of the way, we start off with interpreting the stack as a binary integer, with the *MSB* being the bottom-most digit, and the *LSB* being the top-most digit (where the push-pop action takes place).

Lemma. Pushing a 1 onto the stack is equivalent to multiplying the number by 2 and adding 1 to it.
Proof. Let x be the number represented by the stack prior to the push, and x' after the push.

$$x' = x1 = x \ll 1 + 1 = x \cdot 2 + 1$$

Similarly, when pushing a 0 onto the stack, it is equivalent to multiplying the number by 2.

Lemma. Popping a 1 from the stack is equivalent to dividing the number by 2.
Proof. Let x be the number represented by the stack prior to the pop, and x' after the pop.

$$x'1 = x = x' \ll 1 + 1 = x' \cdot 2 + 1$$
$$\Rightarrow x' = x/2 \qquad \text{(Integer division by 2)}$$

Similarly, when popping a 0 from the stack, it is equivalent to dividing the number by 2.

We are now in a position to describe the state of a stack with the help of counters. We use *3* counters to do so:

- The first two counters π_1 and π_2 are used to store the number in the stack. These two counters shall maintain the invariant that right before reading the next character, exactly one of the stacks stores the number (which maybe 0) represented by the stack and the other counter is 0. We now describe how different operations that are done on these two stacks-

Push: WLOG, assume the first counter π_1 stores the number in the stack and the second counter π_2 stores 0. If the character being pushed onto the stack is 1, then π_2 is incremented initially, otherwise it is not. After this initial operation, π_1 is decremented until it drops to zero, and for each decrement, π_2 is incremented twice. Essentially, twice of the number stored in π_1 is transferred to π_2 plus whatever was pushed onto the stack. The proof of correctness behind the multiplication and addition was provided in *this lemma*.

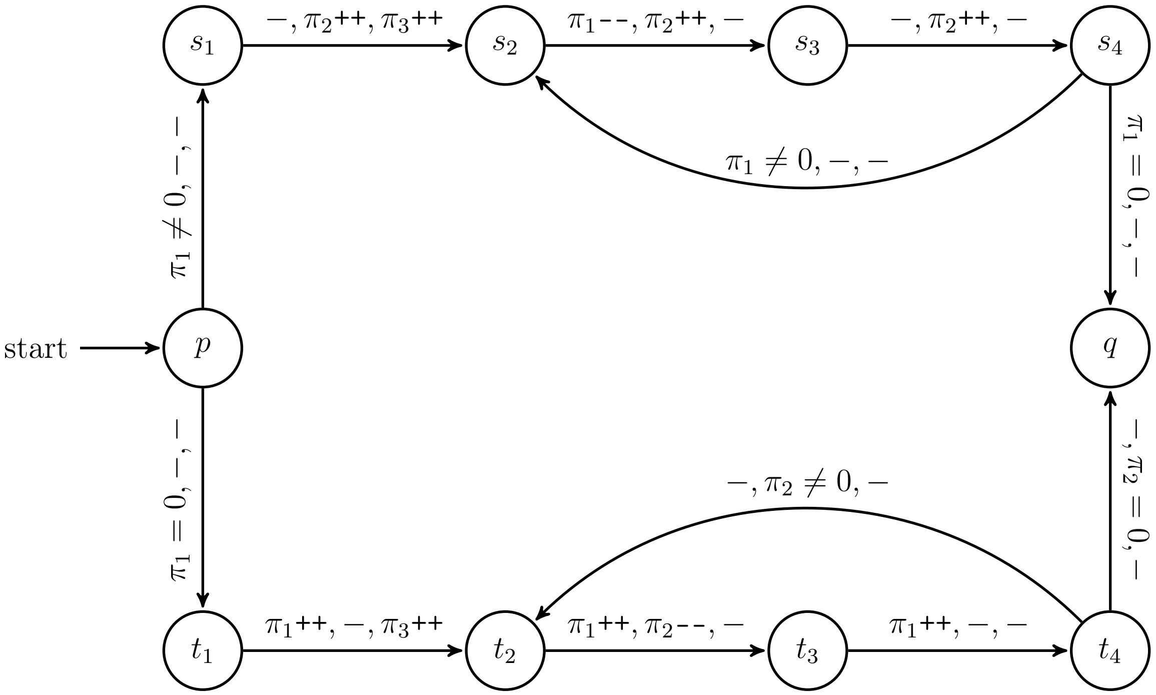
Pop: WLOG, assume π_1 stores the number in the stack and π_2 stores 0. To simulate the pop, π_1 is decremented until it becomes zero, and after every other decrement, π_2 is incremented once (i.e., π_2 is incremented only half of the times, π_1 is decremented). This essentially stores half of π_1 in π_2. The proof of correctness behind storing half was provided in *this lemma*.

Top: Assume again WLOG, that π_1 stores the number and π_2 stores 0. To check the topmost element of the stack, we decrement π_1 in pairs of two and see if the number became zero in the first decrement of the pair, or the second decrement. Since the topmost element contributes to the LSB of the number, checking for the topmost element (0 or 1) is equivalent to checking the number's parity.
We also increment π_2 with each decrement of π_1. This is done to retain the stack number in π_2 at the end of the operation.

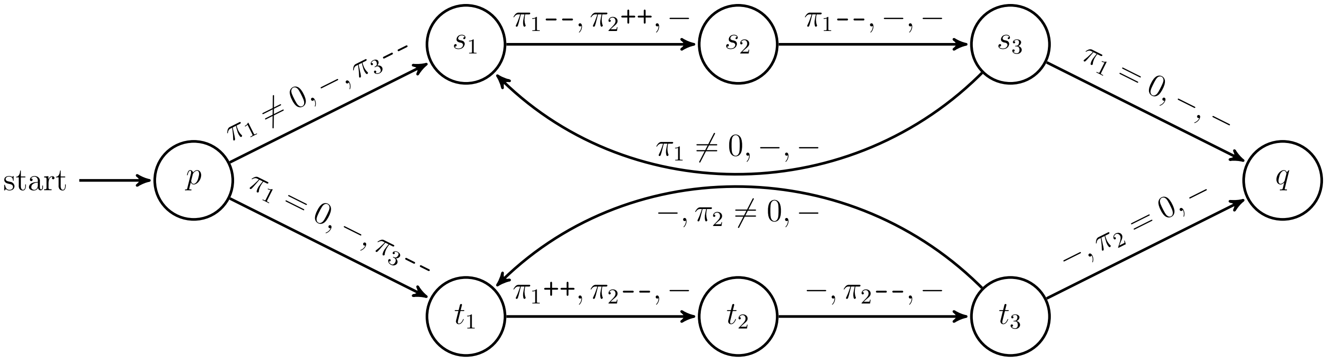
- The third counter π_3 just stores the number of elements (other than $\perp$) currently in the stack. This is to differentiate an empty stack from a stack storing only 0's. This counter is thus used to check if the stack is *Empty* or not. Updation of this counter is fairly trivial: Increment on push, decrement on pop, and do nothing when checking the top element/ checking for emptiness of the stack.
Thus, reading the bottom of the stack symbol ($\perp$) is equivalent to the condition $\pi_3 = 0$.

Lastly, we provide state diagrams to simulate these updations of counters for the sake of completeness.

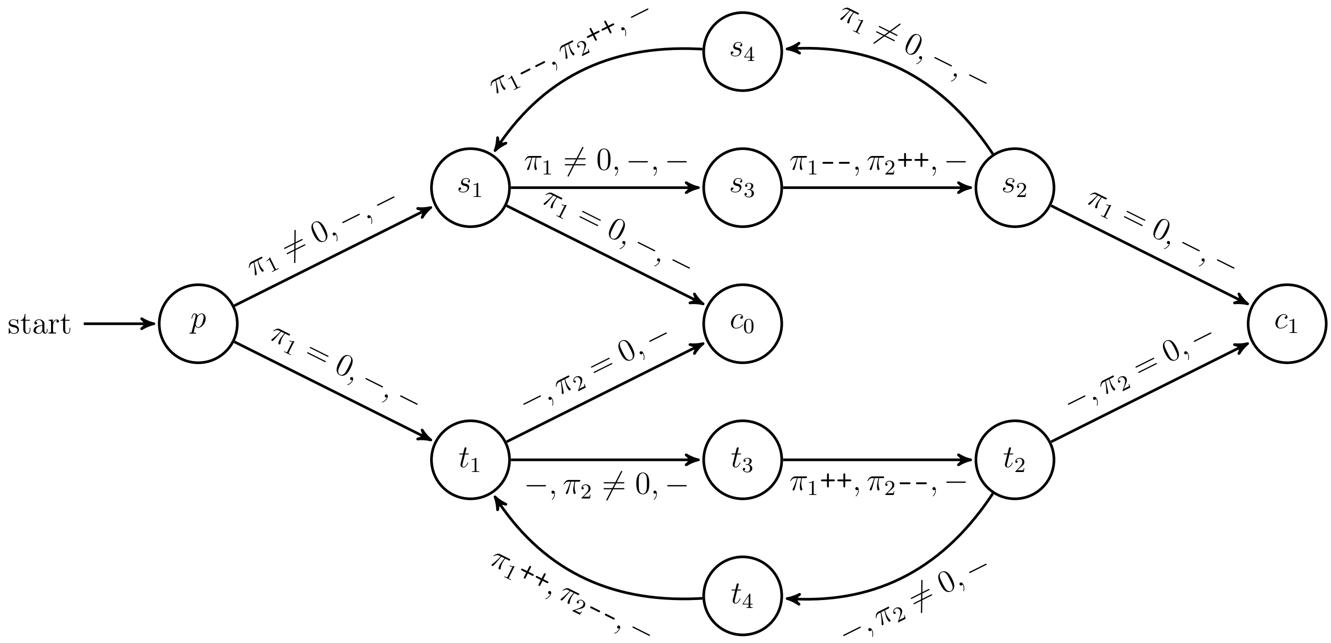
Push: We show the equivalent state diagram for pushing 1 onto the stack using counters. The construction for pushing 0 is similar except for incrementing π_2 between s_1 and s_2 when $\pi_1 \neq 0$ initially/ incrementing π_1 between t_1 and t_2 when $\pi_1 = 0$ initially.



Pop: We show the equivalent state diagram for popping from the stack using counters.



Top: Suppose that we are currently on the state p, and we wish to transition to state c_0 if the symbol on the top of the stack is 0, and to state c_1 if the symbol on the top of the stack is 1. We show the equivalent state diagram for checking the top symbol of the stack using counters.



This completes our construction of "simulating a stack with a finite number of counters".

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 $\Sigma = \{1, \times, =\}$. 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: We make the assumption that it is possible for one of the numbers to be equal to 0, which is represented by the empty string. We also make the assumption that if one of the numbers are 0, then the machine goes to an accept state immediately, since the output will be 0.
If the machine ever finds out a violation of the input then it immediately goes to a reject state. Also, it is assumed that the TM can be destructive to its input.

We first define a high level functioning of the Turing machine before providing the rigorous mathematical details.

- The TM first does a sanity check on the given input for exactly one × and exactly one $=$. $(q_0 - q_5, q_a)$
- The TM picks a 1 in the first number and erases it. If there is no 1 left, the TM comes to a halt. $(q_6 - q_8, q_{14}, q_a)$
- The TM iterates over all the 1's in the second number and writes a corresponding 1 in the tape. $(q_9 - q_{14})$
- After all the 1's in the second number are covered, it goes back to the second step of finding a 1 in the first number.

Essentially, the TM simulates two nested for loops, the first loop decrementing the first number, and the second decrementing the second number, and in each iteration of the innermost loop, the product is incremented by appending a 1 to it.

Now, the Turing machine $M = \langle Q, q_0, \Sigma, b, \delta, q_a, q_r \rangle$ has the definitions as follows:

Q : $\{q_0, q_1, \ldots, q_{14}, q_a, q_r\}$

q_0 : The starting state

Σ : $\{1, \times, =\}$

b : Blank symbol

δ : The transition function is as shown as below:

State	b	1	$\times$	$=$
q_0	$\langle q_1, b, R\rangle$	$\langle q_r, -, -\rangle$	$\langle q_r, -, -\rangle$	$\langle q_r, -, -\rangle$
q_1	$\langle q_r, -, -\rangle$	$\langle q_3, 1, R\rangle$	$\langle q_2, \times, R\rangle$	$\langle q_r, -, -\rangle$
q_2	$\langle q_r, -, -\rangle$	$\langle q_2, 1, R\rangle$	$\langle q_r, -, -\rangle$	$\langle q_a, =, R\rangle$
q_3	$\langle q_r, -, -\rangle$	$\langle q_3, 1, R\rangle$	$\langle q_4, \times, R\rangle$	$\langle q_r, -, -\rangle$
q_4	$\langle q_r, -, -\rangle$	$\langle q_5, 1, R\rangle$	$\langle q_r, -, -\rangle$	$\langle q_a, =, R\rangle$
q_5	$\langle q_r, -, -\rangle$	$\langle q_5, 1, R\rangle$	$\langle q_r, -, -\rangle$	$\langle q_6, =, L\rangle$
q_6	$\langle q_7, b, R\rangle$	$\langle q_6, 1, L\rangle$	$\langle q_6, \times, L\rangle$	$\langle q_r, -, -\rangle$
q_7	$\langle q_r, -, -\rangle$	$\langle q_8, b, R\rangle$	$\langle q_r, -, -\rangle$	$\langle q_r, -, -\rangle$
q_8	$\langle q_r, -, -\rangle$	$\langle q_8, 1, R\rangle$	$\langle q_9, \times, R\rangle$	$\langle q_r, -, -\rangle$
q_9	$\langle q_r, -, -\rangle$	$\langle q_{10}, b, R\rangle$	$\langle q_r, -, -\rangle$	$\langle q_r, -, -\rangle$
q_{10}	$\langle q_{11}, 1, L\rangle$	$\langle q_{10}, 1, R\rangle$	$\langle q_r, -, -\rangle$	$\langle q_{10}, =, R\rangle$
q_{11}	$\langle q_{12}, b, R\rangle$	$\langle q_{11}, 1, L\rangle$	$\langle q_r, -, -\rangle$	$\langle q_{11}, =, L\rangle$
q_{12}	$\langle q_r, -, -\rangle$	$\langle q_{10}, b, R\rangle$	$\langle q_r, -, -\rangle$	$\langle q_{13}, =, L\rangle$
q_{13}	$\langle q_{13}, 1, L\rangle$	$\langle q_r, -, -\rangle$	$\langle q_{14}, \times, L\rangle$	$\langle q_r, -, -\rangle$
q_{14}	$\langle q_a, b, L\rangle$	$\langle q_6, 1, L\rangle$	$\langle q_r, -, -\rangle$	$\langle q_r, -, -\rangle$
q_a	$\langle q_a, -, -\rangle$	$\langle q_a, -, -\rangle$	$\langle q_a, -, -\rangle$	$\langle q_a, -, -\rangle$
q_r	$\langle q_r, -, -\rangle$	$\langle q_r, -, -\rangle$	$\langle q_r, -, -\rangle$	$\langle q_r, -, -\rangle$

q_a : The accept state

q_r : The reject state

The significance of each of the states has been elaborated below:

q_0 : Initial state

q_1 : If the first number is zero, jump to q_2, else jump to q_3.

q_2 : Keep reading the remaining input and jump to q_a afterwards, because the result is 0 in this case.

q_3 : Keep reading the 1's until a × is read, then jump to q_4.

q_4 : If the second number is 0, jump to q_a, else jump to q_5.

q_5 : Read the rest of the second number and transition to q_6 after that.

q_6 : The machine starts the multiplication here. Keep going to the left, until the first 1 in the first number is found. Go to q_7 after that.

q_7 : Erase the first 1 in the first number and go to q_8.

q_8 : Keep reading the remaining 1's in the first number and go to q_9 after reading ×

q_9 : Erase the first 1 of the second number and go to q_{10}.

q_{10} : Keep reading the 1's of the second number and the product (thus far), and the = sign. After the first blank is found, write a 1 and give control to state q_{11}.

q_{11} : Go all the way to the left of the (remaining) second number.

q_{12} : If the second number is still remaining then erase the first 1 and return control to $q_1 0$ to append a 1 to the product. Otherwise, if the second number is empty, give control up to state q_{13}.

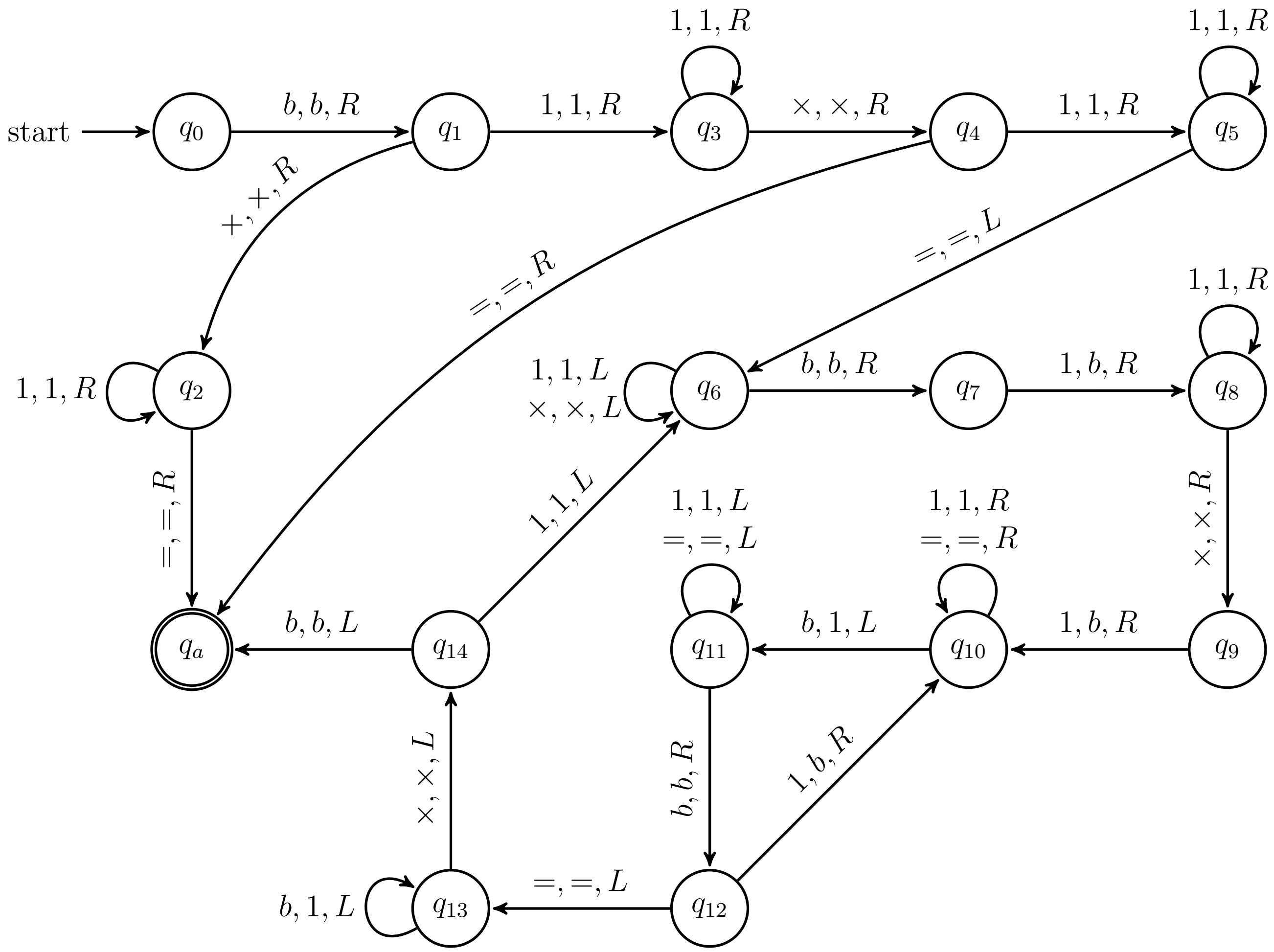
q_{13} : Re-write the second number onto the tape in preparation for the next iteration. Then give the control to state q_{14}.

q_{14} : Check if the first number has any 1's left. If there are any left, return to state q_6, otherwise declare the multiplication to be complete by going to state q_a.

q_a : The multiplication was completed successfully and the machine did not find any violations in the input.

q_r : The multiplication was not completed successfully and the machine stopped abruptly.

The state diagram of the TM is drawn below. The reject state q_r is made implicit, although it is assumed that one of the input numbers being 0 (represented by the null string) implies that the product is 0 irrespective of correctness of input.



Q5. Design a TM (draw the state diagram) that accepts following set of strings:

$$L = \{ww \mid w \in \{a, b\}^*\}$$

Solution: We first define a high level functioning of the Turing machine before providing the rigorous mathematical details.

1. First we determine the parity of the length of the string. Scan the entire input till a blank symbol (right end) is found, while maintaining the length of the string modulo 2 in the finite control. If the length is odd, immediately reject.
2. Then we split the input into two strings. We do so by marking the first half of the string with a tilde superscript, and the second half with a hat superscript. The head of the Turing machine is currently at the right end of the string. Mark a character with a hat, then move to the left end. Mark the left end with a tilde. Then move to the rightmost unmarked character and mark it with a hat. Continue this process till the no element remains unmarked.
3. Now we perform a character by character check of the two strings. Move to the first character of the tilde string (which is the leftmost character). Remember this character

and compare it with the first character of the hat string. If they are unequal, reject the input. Otherwise, cross off both these characters. Continue this process, shuttling between the tilde and the hat strings. If all characters have been crossed off, accept the input.

Now, the Turing machine $M = (Q, \Sigma, \Gamma, \perp, \sqcup, \delta, s, t, r)$ has the definitions as follows:

Q: $\{s, q_1, q_2, \ldots, q_{10}, t, r\}$

Σ: $\{a, b\}$

Γ: $\{a, b, \perp, \sqcup, \tilde{a}, \tilde{b}, \hat{a}, \hat{b}, \not\tilde{a}, \not\tilde{b}, \not\hat{a}, \not\hat{b}\}$

$\perp$: The left end marker

$\sqcup$: The blank symbol on tape

δ: The transition function (details given later)

s: The starting state

t: The accept state

r: The reject state

Now, let us define the transition function. We partition the transitions depending on the high level functionality it is associated with.

(a) If the input string is empty (edge case), accept:

i. $\delta(s, \sqcup) = (t, \sqcup, R)$

(b) Maintaining parity of the length of the string:

i. $\delta(s, a) = (q_1, a, R)$

ii. $\delta(s, b) = (q_1, b, R)$

iii. $\delta(q_1, a) = (q_2, a, R)$

iv. $\delta(q_1, b) = (q_2, b, R)$

v. $\delta(q_2, a) = (q_1, a, R)$

vi. $\delta(q_2, b) = (q_1, b, R)$

vii. $\delta(q_1, \sqcup) = (r, \sqcup, R)$

viii. $\delta(q_2, \sqcup) = (q_3, \sqcup, L)$

(c) Splitting the input into the tilde string and the hat string:

i. $\delta(q_3, a) = (q_4, \hat{a}, L)$

ii. $\delta(q_3, b) = (q_4, \hat{b}, L)$

iii. $\delta(q_4, a) = (q_4, a, L)$

iv. $\delta(q_4, b) = (q_4, b, L)$

v. $\delta(q_4, \perp) = (q_5, \perp, R)$

vi. $\delta(q_4, \tilde{a}) = (q_5, \tilde{a}, R)$

vii. $\delta(q_4, \tilde{b}) = (q_5, \tilde{b}, R)$

viii. $\delta(q_5, a) = (q_6, \tilde{a}, R)$

ix. $\delta(q_5, b) = (q_6, \tilde{b}, R)$

x. $\delta(q_6, a) = (q_6, a, R)$

xi. $\delta(q_6, b) = (q_6, b, R)$

xii. $\delta(q_6, \sqcup) = (q_3, \sqcup, L)$

xiii. $\delta(q_6, \sqcup) = (q_3, \sqcup, L)$

xiv. $\delta(q_6, \hat{a}) = (q_3, \hat{a}, L)$

xv. $\delta(q_6, \hat{b}) = (q_3, \hat{b}, L)$

xvi. $\delta(q_3, \tilde{a}) = (q_7, \tilde{a}, L)$

xvii. $\delta(q_3, \tilde{b}) = (q_7, \tilde{b}, L)$

(d) Crossing off the characters in the two strings:

i. $\delta(q_7, \tilde{a}) = (q_7, \tilde{a}, L)$

ii. $\delta(q_7, \tilde{b}) = (q_7, \tilde{b}, L)$

iii. $\delta(q_7, \hat{a}) = (q_7, \hat{a}, L)$

iv. $\delta(q_7, \hat{b}) = (q_7, \hat{b}, L)$

v. $\delta(q_7, \hat{\cancel{a}}) = (q_7, \hat{\cancel{a}}, L)$

vi. $\delta(q_7, \hat{\cancel{b}}) = (q_7, \hat{\cancel{b}}, L)$

vii. $\delta(q_7, \perp) = (q_8, \perp, R)$

viii. $\delta(q_7, \tilde{\cancel{a}}) = (q_8, \tilde{\cancel{a}}, R)$

ix. $\delta(q_7, \tilde{\cancel{b}}) = (q_8, \tilde{\cancel{b}}, R)$

x. $\delta(q_8, \tilde{a}) = (q_9, \tilde{\cancel{a}}, R)$

xi. $\delta(q_8, \tilde{b}) = (q_{10}, \tilde{\cancel{b}}, R)$

xii. $\delta(q_9, \tilde{a}) = (q_9, \tilde{a}, R)$

xiii. $\delta(q_9, \tilde{b}) = (q_9, \tilde{b}, R)$

xiv. $\delta(q_9, \hat{\cancel{a}}) = (q_9, \hat{\cancel{a}}, R)$

xv. $\delta(q_9, \hat{\cancel{b}}) = (q_9, \hat{\cancel{b}}, R)$

xvi. $\delta(q_9, \hat{b}) = (r, \hat{b}, L)$

xvii. $\delta(q_9, \hat{a}) = (q_7, \hat{\cancel{a}}, L)$

xviii. $\delta(q_{10}, \tilde{a}) = (q_{10}, \tilde{a}, R)$

xix. $\delta(q_{10}, \tilde{b}) = (q_{10}, \tilde{b}, R)$

xx. $\delta(q_{10}, \hat{\cancel{a}}) = (q_{10}, \hat{\cancel{a}}, R)$

xxi. $\delta(q_{10}, \hat{\cancel{b}}) = (q_{10}, \hat{\cancel{b}}, R)$

xxii. $\delta(q_{10}, \hat{a}) = (r, \hat{a}, L)$

xxiii. $\delta(q_{10}, \hat{b}) = (q_7, \hat{\cancel{b}}, L)$

xxiv. $\delta(q_9, \sqcup) = (t, \sqcup, L)$

xxv. $\delta(q_{10}, \sqcup) = (t, \sqcup, L)$

The definitions of the various states are given below:

q_1: The length of the string seen so far is odd

q_2: The length of the string seen so far is even

q_3: State in which a character is marked with a hat

q_4: State to shuttle to the left end during marking of strings

q_5: State in which a character is marked with a tilde

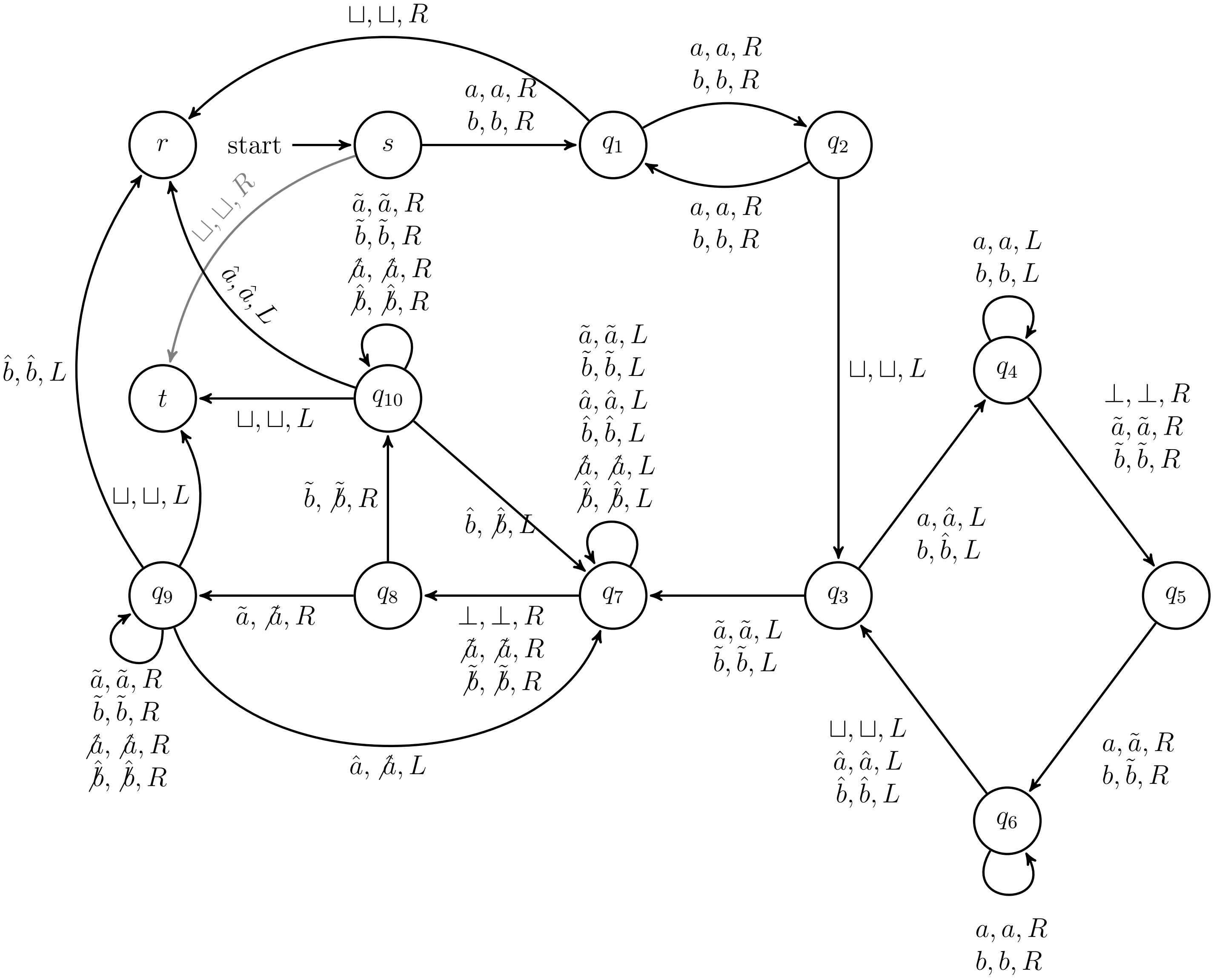
q_6: State to shuttle to the right end during marking of strings

q_7: State to shuttle to the left end to check if characters are equal at corresponding indices

q_8: State to cross off a tilde character

q_9: State to shuttle to the right end when the crossed off tilde character was an a. It either rejects the string if the corresponding hat character is not a, or crosses off the hat character.

q_{10}: State to shuttle to the right end when the crossed off tilde character was a b. It either rejects the string if the corresponding hat character is not b, or crosses off the hat character.


$\sqcup, \sqcup, R$
a, a, R
b, b, R
a, a, R
b, b, R
r
start
s
q_1
q_2
a, a, R
b, b, R
$\sqcup, \sqcup, R$
$\tilde{a}, \tilde{a}, R$
$\tilde{b}, \tilde{b}, R$
$\not{\hat{a}}, \not{\hat{a}}, R$
$\not{\hat{b}}, \not{\hat{b}}, R$
$\hat{a}, \hat{a}, L$
a, a, L
b, b, L
$\tilde{a}, \tilde{a}, L$
$\tilde{b}, \tilde{b}, L$
$\hat{a}, \hat{a}, L$
$\hat{b}, \hat{b}, L$
$\not{\hat{a}}, \not{\hat{a}}, L$
$\not{\hat{b}}, \not{\hat{b}}, L$
$\sqcup, \sqcup, L$
q_4
$\hat{b}, \hat{b}, L$
t
q_{10}
$\sqcup, \sqcup, L$
$\perp, \perp, R$
$\tilde{a}, \tilde{a}, R$
$\tilde{b}, \tilde{b}, R$
$\sqcup, \sqcup, L$
$\tilde{b}, \not{\tilde{b}}, R$
$\hat{b}, \not{\hat{b}}, L$
$a, \hat{a}, L$
$b, \hat{b}, L$
q_9
q_8
q_7
q_3
q_5
$\tilde{a}, \not{\tilde{a}}, R$
$\perp, \perp, R$
$\not{\tilde{a}}, \not{\tilde{a}}, R$
$\not{\tilde{b}}, \not{\tilde{b}}, R$
$\tilde{a}, \tilde{a}, L$
$\tilde{b}, \tilde{b}, L$
$\tilde{a}, \tilde{a}, R$
$\tilde{b}, \tilde{b}, R$
$\not{\hat{a}}, \not{\hat{a}}, R$
$\not{\hat{b}}, \not{\hat{b}}, R$
$\sqcup, \sqcup, L$
$\hat{a}, \hat{a}, L$
$\hat{b}, \hat{b}, L$
$a, \tilde{a}, R$
$b, \tilde{b}, R$
$\hat{a}, \not{\hat{a}}, L$
q_6
a, a, R
b, b, R

Q6. Prove that the following set is not computable:

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

Solution: Let us define the languages

$$\begin{aligned} A_{TM} &= \{\langle M, w\rangle \mid \text{M is a Turing Machine such that } M \text{ accepts } w\} \\ E_{TM} &= \{\langle M\rangle \mid \text{M is a Turing Machine such that } L(M) = \phi\} \\ L' &= \{(p, q) \mid \text{there exists no string } x \text{ accepted by both TM } M_p \text{ and TM } M_q\} \end{aligned}$$

Clearly the turing machine that recognizes L' is the complement of the corresponding turing machine that recognizes L.

Claim: The language A_{TM} is not computable

Proof: We borrow the proof from Professor Raghunath Tewari's note. Suppose A_{TM} is computable. Then there exists a halting TM H that on input $\langle M, w\rangle$, *accepts* if M accepts w and *rejects* if M does not accept w.
Using H, construct another machine N as follows.

Description of Turing Machine N

Input: $\langle M\rangle$ where M is a TM

(a) Simulate H on $\langle M, \langle M\rangle\rangle$.

(b) If H rejects then *accept* else *reject*.

The machine H is a halting TM, hence N is also a halting TM. Now we provide input $\langle N\rangle$ to N.

$$N \text{ accepts } \langle N\rangle \iff H \text{ rejects } \langle N, \langle N\rangle\rangle \iff N \text{ does not accept } \langle N\rangle$$

This is clearly a contradiction, therefore A_{TM} is not computable.

$\implies \overline{A_{TM}}$ is not computable.

Claim: The language $\overline{A_{TM}}$ reduces to E_{TM}

Proof: We borrow the proof from Professor Raghunath Tewari's note. We will construct a computable function f that takes input $\langle M, w\rangle$ and produces output $\langle N\rangle$ such that, M does not accept $w \iff L(N) = \phi$

The reduction function f:

Input: $\langle M, w\rangle$

Construct a TM N such that on input x it does the following:

(a) if $x \neq w$ then reject

(b) else simulate M on w. If M accepts then accept else reject

Output: $\langle N\rangle$

Proof of Correctness:

$$M \text{ does not accept } w \implies N \text{ rejects every input } \implies L(N) = \phi$$

$$M \text{ accepts } w \implies N \text{ accept } w \implies L(N) \neq \phi$$

Therefore, M does not accept $w \iff L(N) = \phi$, thus $\overline{A_{TM}}$ reduces to E_{TM}
$\implies E_{TM}$ **is not computable**

Claim: The language E_{TM} reduces to L'

Proof: We will construct a computable function f that takes input $\langle M \rangle$ and produces an output (p, q) such that $L(M) = \phi \iff L(M_p) \cap L(M_q) = \phi$

The reduction function f:
Input: $\langle M \rangle$

(a) Set $M_p := M$

(b) Construct a TM M_q that accepts all inputs (that is $L(\overline{M_q}) = \phi$)

Output: (p, q)

Proof of Correctness:
Now,

$$L(M) = \phi \iff L(M_p) = \phi \iff L(M_p) \cap L(M_q) = \phi$$

Therefore, E_{TM} reduces to L'
Since E_{TM} is not computable, therefore L' is not computable

$\implies$ **L is not computable**