Homework 06

Aaron Wang

March 28 2025

1. **Doubly infinite tapes** [Problem 3.11]. A Turing machine with a doubly infinite tape is like a TM as defined in the book, but with a tape that extends infinitely in both directions (not just to the right). Initially, the head is at the first symbol of the input string, as usual, but there are infinitely many blanks to the left. Show how, given a TM with doubly infinite tape, to construct an equivalent standard TM. An **implementation description** in the style of Proof 3.13 is fine, and it's also fine to use any results proved in the book or in class.

   The intuition for this construction is that the whenever we need to go left, but have reached the left end of the tape, we add a space and move onto that space. This is done by shifting the rest of the tape over to the right by 1 unit and writing a space at the new extra space.

   Let M be a TM with a doubly infinite tape. Define M' to be a standard TM.

   M' = "On input string w"

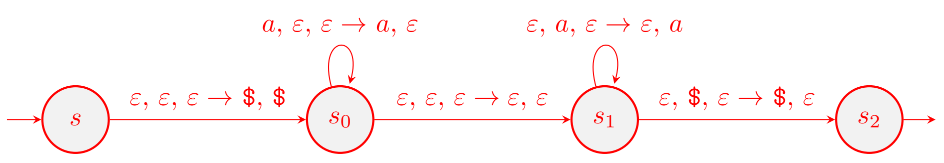
   1. Initialize tape by
      - Insert a # at the beginning.
   2. Do what M would do
      - The head starts at the w_1 (one after the #).
      - If the head ever ends up on #, insert a space right after and move the head there.
      - *accept*, *loop*, or *reject* based on how M would.

2. **Two-stack PDAs**. A *two-stack pushdown automaton* (2PDA) is a pushdown automaton with two stacks. Show that any Turing machine M can be converted into an equivalent 2PDA P.

   The intuition for the following conversion is this. Use the 2 stacks in the same way as the tape would be used where the first stack is used to hold everything preceding the head and the other stack holds everything following and where the head is currently looking at the top element on the second stack. The head moves right by moving one element from the second stack to the first stack, and moves left by moving an element from the first stack to the second. To consider the edge case of moving left when all the way at the left already, you check for a \$ and in that case, you move back right. To consider the edge case, where we extend pass the starting string (go to trailing spaces), we would Let M be a TM.

   Let $P = \{Q, \Sigma, \Gamma, \delta, s, F\}$ be the 2PDA such that

   1. The string (Σ) alphabet is the same as that of M.
   2. $\Gamma = \Sigma \cup \{␣, \$\}$. The stack alphabet is string alphabet with ␣ and \$.
   3. Have pre-processing states to read the input onto the stack and add \$ to demarcate the end and beginning. $\forall a \in \Sigma$.[1]



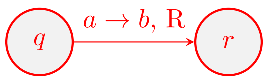
$a,\ \varepsilon,\ \varepsilon \to a,\ \varepsilon$
$\varepsilon,\ a,\ \varepsilon \to \varepsilon,\ a$
s
$\varepsilon,\ \varepsilon,\ \varepsilon \to \$,\ \$$
s_0
$\varepsilon,\ \varepsilon,\ \varepsilon \to \varepsilon,\ \varepsilon$
s_1
$\varepsilon,\ \$,\ \varepsilon \to \$,\ \varepsilon$
s_2

   4. For each state q of M, the same state q in P.
      i. $F = \{q_{accept}\}$.

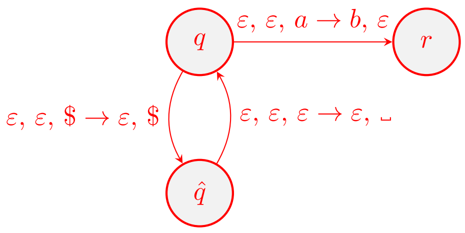
[1]The outgoing transition from s_2 will be $\varepsilon,\ \varepsilon,\ \varepsilon \to \varepsilon,\ \varepsilon$ into the original start state

   ii. Additionally, to ensure rejection, make sure that there are no outgoing transitions from q_{reject}.

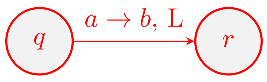
5. For each transition of M s.t. $a, b \in \Sigma \cup \{␣\}$ create transitions in P such that first if we are reading (trailing spaces), $\$ \to ␣$ and then R transitions move right and L transitions move left if possible.
   i. R transitions.
      For every transition of M that looks like this



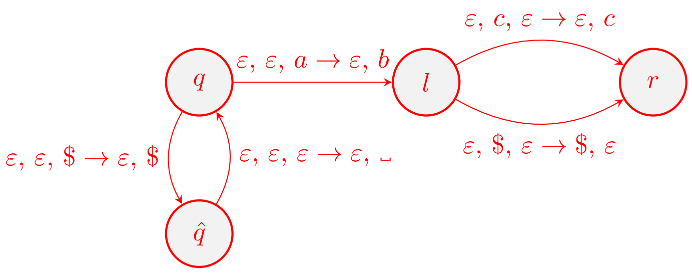
      Create transitions like this in P


q
$\varepsilon,\ \varepsilon,\ a \to b,\ \varepsilon$
r
$\varepsilon,\ \varepsilon,\ \$ \to \varepsilon,\ \$$
$\varepsilon,\ \varepsilon,\ \varepsilon \to \varepsilon,\ ␣$
$\hat{q}$

   ii. L transitions.
      For every transition of M that looks like this



      Create transitions like this in P $\forall c \in \Sigma \cup \{␣\}$


$\varepsilon,\ c,\ \varepsilon \to \varepsilon,\ c$
q
$\varepsilon,\ \varepsilon,\ a \to \varepsilon,\ b$
l
r
$\varepsilon,\ \varepsilon,\ \$ \to \varepsilon,\ \$$
$\varepsilon,\ \varepsilon,\ \varepsilon \to \varepsilon,\ ␣$
$\varepsilon,\ \$,\ \varepsilon \to \$,\ \varepsilon$
$\hat{q}$

6. Q is the set of all the states we created.
   i. The states from pre-processing s, s_0, s_1 and s_2.
   ii. The states copied from the M.
   iii. The new states $\hat{q}$ and l defined by the transitions (for every transition).

3. Brain fun. This problem is about a programming language known as P'' in polite company.

   Describe how to compile any $\mathcal{P}''$ program P into the formal description of a Turing machine M_P equivalent to P. The input to M_P would be a string $w \in \Sigma^*$, and it should accept iff P accepts w. It should be a standard single-tape TM, but you can use S ("stay") actions.

   Let P be a $\mathcal{P}''$ program. Let us use the follow construction to create a Turing Machine M that compiles P into a standard single-tape TM.

   i. Initialize an empty stack to keep track of open loops.[2]
   ii. Go through every character of the code and parse it in this way, keeping track of a counter i for every character.[3] In the following steps, every transition created is from q_i to q_{i+1}

      - `<` $\forall j \in [0, n-1]$ Create transitions $a_j \to a_j$, L
      - `>` $\forall j \in [0, n-1]$ Create transitions $a_j \to a_j$, R
      - `+` $\forall j \in [0, n-2]$ Create transitions $a_j \to a_{j+1}$, S
        Create transition $a_{n-1} \to a_0$, S
      - `-` $\forall j \in [1, n-1]$ Create transitions $a_j \to a_{j-1}$, S
        Create transition $a_0 \to a_{n-1}$, S
      - `[` Add i to stack of open bracket placement.
        $\forall j \in [1, n-1]$ Create transitions $a_j \to a_j$, S
      - `]` Let j be popped value from the open bracket placement stack.
        Let $q_i = q_j$ (create the looping state)
        Create transition $a_0 \to a_0$, S

   iii. $\forall q \in Q$ create a transition $\sqcup \to a_0, S$ from q to itself.
   iv. Let m be the length of the program P. q_m is the final state that we have created so far. From q_m...
      - $\forall j \in [1, n-1]$ create a transition $a_j \to a_j$, S to q_{accept}.
      - create a transition from $a_0 \to a_0$, S to q_{reject}.

   The formal description of $M = \{Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}\}$:

   - $Q = \{q_0, q_1 ... q_m, q_{accept}, q_{reject}\}$.
   - Σ is a subset of Γ as defined by the directions without a_0 and $\sqcup$.
   - $\Gamma = \{a_0, ..., a_{n-1}, \sqcup\}$
   - δ as defined above.
   - q_0 as defined above.
   - q_{accept} as defined above.
   - q_{reject} as defined above.

[2] At any point, if there is nothing to pop from this stack, P was not a proper program. Additionally, if there are extra elements on the stack at the end, P was not a proper program.

[3] i starts at 0, increments by 1 every time and $i, j \in \mathbb{Z}$