1

Strategy

Proof by constructing a TM.

Outline

We simply determine if $L(M)$ is the same as its reversal language. This works because the reverse of the implication in the definition of S must also be true for any machine in S by reversing twice.

Proof

Consider a TM which, on input $\langle M \rangle$, does the following:

1. Check if the input is a valid encoding of a DFA. If not, reject.
2. Compute the NFA N for the reversal language of $L(M)$ (by reversing all transitions, making the start state the only final state, and adding a new start state with ε-transitions to all of the old final states).
3. Convert N to a DFA D by its powerset.
4. Determine if the DFAs M and D are equivalent, using the decidability of EQ_{DFA}. Accept if they are and reject if they are not.

Obviously if M accepts, then for any w in $L(M)$, $w^{\mathcal{R}} \in L(M)^{\mathcal{R}} = L(D)$.

In the other direction, if M rejects then either some w is in $L(M)$ but not $L(D)$, or in $L(D)$ but not $L(M)$. In the first case, $w \in L(M)$ but $w^{\mathcal{R}} \notin L(M)$. In the second, $w^{\mathcal{R}} \in L(M)$ but $w = (w^{\mathcal{R}})^{\mathcal{R}} \notin L(M)$.

Thus this TM accepts exactly S.

2

Strategy

Proof by construction for each direction of the biconditional.'

Outline

Given a decider for L, we construct an enumerator simply by iterating through Σ^* in shortlex order, running each string through the decider and printing the ones it accepts. Given an enumerator for L, we decide whether a string is in L by running the enumerator until it prints the string, whence we accept, or one greater, whence we reject.

Proof

Let L be a language, and first assume it is decidable, say by a TM M. Then our enumerator E operates as follows. Iterate through the strings of Σ^* in shortlex order. For each one, run M on it, and, if M accepts, write it to the write tape and print (otherwise move on). Since E only prints strings M accepts and M decides L, E only prints strings in L. Further, for each string in L, E will eventually reach it while iterating through Σ^* since E never hangs (note that running M will never hang by the definition of a decider), at which point E will find that E accepts it and therefore print it. Thus E enumerates L. Finally, as L considers each string exactly once in shortlex order, it enumerates L in shortlex order, as required.

Now, suppose L is enumerated in shortlex order by some enumerator E. First, if L finite, then it is trivially decidable (e.g., since it is regular). Otherwise, we construct a decider M that behaves as follows. On input w, run E until it outputs a string v with $w \leq v$ (this is the shortlex ordering). If $v = w$, then accept; otherwise, reject.

First, note that E must eventaully output such a v, for there are only finitely many strings u with $u < w$, but L is infinite, so some such v is in L and therefore must be printed by E in finite time, since E enumerates L. Now, let $w \in \Sigma^*$ and let v be the first string E prints with $w \leq v$. Suppose $w \in L$. Since E enumerates L, E will eventually print w, and furthermore, since it does so in shortlex order and w is the least string less than or equal to itself, we must have $v = w$. Thus M will accept w. If $w \neq L$, then E will never print w, so we must have $v \neq w$, and thus M will reject w. Therefore, M accepts all $w \in L$ and rejects all $w \notin L$, so M decides L, concluding the proof.

3

Strategy

Construct a stay-put TM in one direction, and a DFA in the other.

Outline

A DFA can be converted to a stay-put TM by directly translating its transitions into rightward TM transitions. The converse is more complicated. Rightward transitions can be simulated straightforward on a DFA, but what about stay-put transitions? The answer is to simulate a series of stay-put transitions by transitioning directly to the state they eventually move right on – or if they never do, accepting or rejecting on the spot.

Proof

Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$, we define a TM $T = (Q \sqcup \{a, r\}, \Sigma, \Sigma \sqcup \{\square\}, \delta', q_0, a, r)$, with δ' defined as follows:

$$\delta'(q, c) = \begin{cases} (\delta(q, c), \square, R), & c \in \Sigma \\ (a, \square, R), & c = \square, q \in F \\ (r, \square, R), & c = \square, q \notin F. \end{cases}$$

For any input which takes M to state q, the same input will take T through the same states to q, since each transition is an R-transition which moves on to the next letter. Upon reaching the end of the input, it will then accept iff the last state reached is an accept state. Thus T recognizes exactly the inputs M does.

Conversely, given a TM with stay put instead of left $T = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$, we construct a DFA $M = (Q, \Sigma, \delta', q_0, \{q_a\})$, with $\delta'(q, c)$ defined as follows for any $q \in Q, c \in \Sigma$:

- If $q = q_a$ or $q = q_r$, then $\delta(q, c) = q$.
- If there is a number $n \geq 0$ and a sequence of states $q = q_0, q_1, \ldots, q_{n+1}$ and characters $c = c_0, c_1, \ldots, c_{n+1}$ with $\delta(q_i, c_i) = (q_{i+1}, c_{i+1}, S)$ for $i < n$ and $\delta(q_n, c_n) = (q_{n+1}, c_{n+1}, R)$, then $\delta'(q, c) = q_{n+1}$.
- Similarly, if there is an $n \geq 0$ and a sequence of states $q = q_0, q_1, \ldots, q_{n+1} \in \{q_a, q_r\}$ and $c = c_0, c_1, \ldots, c_{n+1}$ with $\delta(q_i, c_i) = (q_{i+1}, c_{i+1}, S)$ for $i < n$ and $\delta(q_n, c_n) = (q_{n+1}, c_{n+1}, R)$ or (q_{n+1}, c_{n+1}, S), then $\delta'(q, c) = q_{n+1}$.
- Otherwise, $\delta'(q, c) = q_r$.

If T reads a character from the input and enters state q and never moves right again, it must accept, reject, or loop with only S-transitions. In these cases, M will move to the appropriate final state and stay there until the end of computation.

Alternatively, if T reads a character from the input and eventually moves on to the next character, M will eventually move on to the next character as well, and read it in the same state that T eventually reads it in.

Thus M accepts iff T accepts.

4

Strategy

In one direction, simulate all the branches using configuration histories. In the other, store the tape before and after the head in the two stacks.

Outline

To simulate a 2-PDA on a Turing machine, we construct a Turing machine that stores the configuration of the PDA. It then performs a BFS on the possible configurations of the PDA, and accepts iff it finds a valid path to an accepting configuration.

To simulate a Turing machine on a 2-PDA, we first ensure that the 2-PDA is deterministic. We use the right stack to represent the tape after (or at) the Turing machine's head, and the left stack to represent the tape before the Turing machine's head. We thus must first move the input into the right stack. Once that is done, we can determine all our actions by reading from the right stack, and perform moves by moving a character from the right stack to the left stack or vice versa.

Proof

We simulate a 2-PDA on a Turing machine by using the tape as a "queue" of possible configuration of the 2-PDA (say by simply keeping them in sequence with delimiters, popping by reading the first configuration and then moving the others left, and pushing by adding configurations at the end). Each configuration in the "queue" encodes a state of the 2-PDA, the contents of both stacks, and the remaining input. Our TM will then pop a configuration, iterate through the possible 2-PDA transitions from this configuration, and for each transition pushing the resulting configuration onto the stack. If the transition popped has no more input and is in an accept state, the TM accepts. If the "queue" is ever empty, the TM rejects.

The 2-PDA accepts iff there is some sequence of valid transitions which brings it to an accept state with no input left. If such a sequence exists the TM will eventually follow it and accept; conversely, if no such sequence exists the TM will not accept. Thus this TM accepts iff the 2-PDA did.

Conversely, we simulate a TM $T = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ with a 2-PDA $P = (Q \sqcup (Q'\Sigma, \Gamma \sqcup \{L\}, \delta', q_0', \{q_a\})$ with

$$Q' = Q \sqcup \{r(q,g), l(q,g) \mid q \in Q, g \in \Gamma\} \sqcup \{q_0', q_L, q_R\}$$

and δ' as follows:

$$q'(q, c, g_l, g_r) = \begin{cases} \{(q_R, L, \varepsilon)\}, & q = q_0', c = g_l = g_r = \varepsilon \\ \{(q_R, \varepsilon, c)\}, & q = q_R, c \neq \square, g_l = g_r = \varepsilon \\ \{(q_L, \varepsilon, \varepsilon)\}, & q = q_R, c = \square, g_l = g_r = \varepsilon \\ \{(q_L, \varepsilon, g_r)\}, & q = q_L, c \neq L, c = g_r = \varepsilon \\ \{(q_0, \varepsilon, \varepsilon)\}, & q = q_L, g_l = L, c = g_r = varepsilon \\ \{(l(q', c'), g_l, \varepsilon)\}, & \delta(q, g_r) = (q', c', R), g_l = c = \varepsilon \\ \{(r(q', g_l), \varepsilon, c')\}, & \delta(q, g_r) = (q', c', L), g_l = c = \varepsilon \\ \{(q', c', \varepsilon)\}, & q = l(g', c'), g_l = g_r = \varepsilon \\ \{(q', \varepsilon, c')\}, & q = r(g', c'), g_l = g_r = \varepsilon \\ \{\}, & \text{otherwise.} \end{cases}$$

All outputs of δ' are size 1 or 0, so P is deterministic. It first adds the entire input to the first stack, then reverses it so it is in the second stack with the first character at the top. At any point, it will modify the tape to the left and right of the head exactly as T would, ensuring that it will ever reach an accept state iff T will ever reach q_a. Thus it accepts exactly the strings T does.