CS 332 – Spring 2021, Assignment 5

Colaborators: None

Answer 1

(a) Get the M_1, and replace the accept state with just a normal state, and keep the rest of M_1, then make a non-deterministic transition from this state, to the start state of M_2. Run the input string through M_1, and keep a copy of it in M_2. Then, run the string through M_1, if at any point it rejects, reject it, if it accepts, move on to M_2 non-deterministically. Once you reach M_2 while tracking in what square you ended in M_1 (the last used square might not be the square the machine is in on accepting you need to track the last used one), go to the second tape, and walk through deleting all the symbols until the last symbol you used for M_1, and run M_2 from there, if it rejects reject, if it accepts accept. If the string runs forever either on M_1 and M_2, then we know they are still Turing Recognizable, and rejected.

(b) Since we have already proven in class that all NTM's can be transformed into DTM's, by proving that we can simulate the concatenation of two TM recognizable languages using an NTM, then we can also do it in a DTM, thus the concatenation operator is closed for all Turing-Recognizable machines.

(c) Keep the original String in a secondary tape, and run the string via M with a single alteration that checks for the empty string and accepts. If the string isn't empty, and gets accepted by M, then you'll need to keep track of the last used section of the tape. Then with that in mind, copy the original string back to the first tape, clear everything before the last used square, and run the leftovers on M again, and keep doing this until there is no more string. If at any point M rejects, reject, if not, accept only once there is no more string. Should be noted that whenever you clear the part before the square, you should do the same on the secondary tape.

(d) Again, as seen above, by proving that we can use an NTM to obtain the * operator on a Turing-recognizable machine, then we can, as seen in class, transform this NTM into a DTM, thus proving the operator is closed for all Turing-Recognizable machines.

(e) Technically no changes would need to occur, since if we ever reach a reject state, we were already rejecting. The only change is that we would know that all inputs would result in a decision, so there is no risk of running forever, although technically if they did run forever it would still reject.

Answer 2

(a)

(1) Q, a set of states
(2) $\sum$, an alphabet not containing the blank symbol
(3) Γ, a tape alphabet, where $\bigsqcup \in \Gamma$ and $\sum \subseteq \Gamma$
(4) : $Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, F\}$
(5) $q_0 \in Q$
(6) $q_{accept} \in Q$
(7) $q_{reject} \in Q$

(b) This eco-friendly TM, or ETM, is basically the same as a multi-tape TM, since the backside of the tape, works just as another tape, since the R transition basically just moves to the other size on the same slot, an equal transition could be created to move to another tape, and go to the same indexed square than it was before.

With that said, we have already proven that a multi-tape TM can be represented as a regular TM, with some loss of efficiency, so, since using an ETM is basically the same thing as using a TM with two tapes, then we can extrapolate that using an eco-friendly TM is the same as using a regular TM, with some syntactic rules to allow for the same functionality (as seen in class).

Answer 3

(a)

```
# If you are copying your code directly from here, please be careful about spacings,
# latex might have altered some stuff and python might complain.

def simulate_DFA(self, encoded_DFA, input_string):
    (transitions, start_state, final_states) = self.encoding_to_triple(encoded_DFA)
    transcript = []
    currentState = start_state
    transcript.append(currentState)


    for symbol in input_string:
        currentState = transitions[(currentState, symbol)]
        transcript.append(currentState)

    finalString = "reject"
    if (currentState in final_states):
       finalString = "accept"

    transcript.append(finalString)

    return transcript
```

(b) As seen before, $x = 0011110$.

The inputs for this program are:

$DFA = (q0, 0, q1), (q0, 1, q0), (q1, 0, q1), (q1, 1, q2), (q2, 0, q3), (q2, 1, q2), (q3, 0, q3), (q1, 1, q0) \# q0 \# q0, q1$

$inputString = 0011110$.

The outputs were $q0, q1, q1, q0, q0, q0, q0, q1, accept$.

Answer 4

(a) $EQ_{DFA,REX} = \{\langle D, R\rangle \| D$ is a DFA, and R is a regular expression and $L(D) = L(R)\}$

(b) We know that, if the DFA and the regex recognize the same language, than if a string is recognized by one, it'll have to be recognized by the other. As such, to respond to the $EQ_{DFA,REX}$ above then we can design a Turing machine T that:

$T =$ "On input $\langle D, R\rangle$ where D is a DFA and R is a regular expression:

1. Convert R into an NFA N_R using Theorem 1.54 from the book.
2. Convert N_R into a DFA D_R using Theorem 1.39 from the book.
3. Run D and D_R on the TM from Theorem 4.5 from the book.
4. If the Theorem 4.5 accepts, then accept, if it rejects then reject."

As each of the individual theorems called here are decidable (as stated in the book), then $EQ_{DFA,REX}$ is decidable.