CSCI 1010 - Fall'26 Theory of Computation Lorenzo De Stefani

HW5

Due: n/a

Reminder: Submit your assignment on Gradescope by the due date. Submissions must be typeset. Each page should include work for only one problem (i.e., make a new page/new pages for each problem). See the course syllabus for the late policy.

While collaboration is encouraged, please remember not to take away notes from any labs or collaboration sessions. Your work should be your own. Use of other third-party resources is strictly forbidden.

Please monitor Ed discussion, as we will post clarifications of questions there.

Problem 1

A ***Turing machine with left reset*** is similar to an ordinary Turing machine, but the transition function has the form:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, RESET\}$$

.

If $\delta(q, a) = (r, b, RESET)$, when the machine is in state q reading an a, the machine's head jumps to the left-hand end of the tape after it writes b on the tape and enters state r. Note that these machines do not have the usual ability to move the head one symbol left. Show that Turing machines with left reset recognize the class of Turing-recognizable languages.

Solution: To do this, we must show that Turing Machines with left reset are equivalent to an ordinary Turning Machines. We will provide high level argument, though the proof should be a little more fleshed out for full points.

First, an ordinary Turing Machine can simulate a Turing Machine with left reset. A Turing Machine with left reset has equivalent actions to the Turing Machine except for that it can reset to the left hand end of the tape at any point. Then, what we must show is that an ordinary Turing Machine can simulate this functionality. The Turing Machine can do this by moving left until it reaches the left hand side of the tape. Therefore, an ordinary Turing Machine can simulate a Turing Machine with left reset.

Next, a Turing Machine with left reset can simulate an ordinary Turing Machine. Once again, the two machine's actions are equivalent except for the fact that the Turing Machine can move left. In order to simulate this, the Turing Machine with left reset can do the following (note that marking can be done by introducing a new character for every character in the alphabet, with a specific subscript to indicate what the marker is):

1. Mark the current cell with a unique subscript.
2. Perform the following:
   (a) Reset and go right until you reach the last marked character (defined with a unique indicator) or an unmarked character (while you go right, mark every character with a not last character indicator).
   (b) If the character is unmarked, mark it with the last marked character indicator. Move right. If the right character, is the original character, break from this loop.
   (c) If the character is marked, replace it with a not last character indicator and move to the next character. Replace that character with a last character indicator. Move right. If the next character is the original character, break. Otherwise, repeat.
3. We are at the original character and know we have a is last character marker to the left. Replace the original character with its old self (each character has its own specific indicator character it corresponds to).
4. Reset. Replace all characters with their old, unmarked character. When you reach the character with the is last character indicator, replace it with its old character and stop.

Problem 2

Given an arbitrary Turing decidable language L', consider the following language

$$L = \{w | \exists w' \in L' \text{ such that } w \text{ is a substring of } w'\}$$

1. Prove that L is Turing recognizable. That is, give a TM recognizer M for L. Prove that it is correct (i.e., $L = L(M)$) and that it halts and accepts any member of L

2. Without providing rigorous proof, give an intuitive argument for whether you believe L to be decidable.

3. Suggest a property that, if L' were to have it, would guarantee L to be decidable. Motivate your answer.

Solution:

1. Turing-recognizability

   By definition of Turing decidability, there exists some TM M' that decides over language L'. Given M', we can construct a TM M that recognizes (but not necessarily decides) over L as follows: M simulates M' it by treating what's the original input string as the input string to M'. If the simulation reveals that M' accepts the string, M also accepts the string. If the simulation reveals that M' rejects the string, M appends a character from the alphabet onto the string, and simulates M' running on this new string. If the simulation reveals that M' accepts the string M also accepts the string. If the simulation reveals that M' rejects the string, M replaces the previously appended character with a different character and re-simulates M' on this new string. This process is repeated until all characters in the alphabet have been tried.

   If all characters have been used, M reverts to the original string, and tries prepending each character this time. If nothing is accepted, it goes back to appending, this time appending two characters. Then, it prepends one character and appends two characters, and lastly prepends two characters and appends two characters. M repeats this process until a string is eventually accepted.

2. Turing-decidability

   It is not decidable because the language L' may be infinite. If it is infinite, for a string that is not a substring, M is bound to run forever since its termination condition is when a superstring is found.

3. Restricting L'

   There are multiple possible answers.

   - Limit the length of strings in L'
   - Make L' regular

Problem 3

A *very sparse language* is a language L over an alphabet Σ that contains exactly one string of each length. If L is a very sparse language, it contains the empty string, exactly one string of length one, exactly one string of length two, and so on. Prove that if L is a very sparse language and is Turing-recognizable, then L is decidable.

Solution: Let L be a recognizable, very sparse language. Suppose R is a recognizer of L, and we would like to be able to decide L. We can construct such a decider D as follows:

On input x, measure the length n of x. Over a finite alphabet Σ, there are a finite number of strings with length n. So, D runs R on all of these strings simultaneously. Since we know that there is exactly one string in L of each length, eventually one of these strings must be recognized; and because we are running each simultaneously (by doing them one step at a time), even if some loop forever it will not block finding the string in L.

Thus, D's simulations of R will accept exactly once on one string of length n, which must be the corresponding string in L. If the accepted string is x, D accepts because x must be in L. Otherwise D rejects, because the only string in L with the same length as x must not be x. Hence, D is a decider for L.