CSE 547T Automata Theory Spring Semester 2017

Homework 3: Turing Machines and Computability

Assigned: 3/20/2017 *Due Date: 4/5/2017*

This homework must be completed and submitted electronically. Formatting standards, submission procedures, and (optional) document templates for homeworks may be found at

`http://classes.engineering.wustl.edu/cse547/ehomework/ehomework-guide.html`

Advice on how to compose homeworks electronically, with links to relevant documentation for several different composition tools, may be found at

`http://classes.engineering.wustl.edu/cse547/ehomework/composing-tips.html`

Please remember to

- **create a separate PDF file (typeset or scanned) for each problem;**
- **include a header with your name, WUSTL key, and the homework number at the top of each page of each solution;**
- **include any figures (typeset or hand-drawn) inline or as floats;**
- **upload and submit your PDFs to Blackboard before class time on the due date.**

Always show your work and prove your constructions correct.

1. Prove or disprove the decidability of each of the following properties of Turing machines. **Do not use Rice's Theorem**. If you want to prove undecidability, you may reduce from any of SA, ACC, HALT, EMPTY, FINITE, or from their complements under the set of valid input encodings.

   (a) Given a TM M, is $L(M)$ regular?

   (b) Given a TM M and a constant k, does M move its head past the kth cell on its tape (counting from the left) on some input?

   (c) Given two TMs M_1 and M_2, do they accept the same language?

2. Say that a (single-tape) TM is *rightward-bound* on input w if its head never moves left at any point during its computation on w.

   (a) Is the language $L_{rb} = \{(M, w) \mid M \text{ is rightward-bound on } w\}$ decidable? Prove or disprove.

   (b) Consider the set of all TMs that are rightward-bound on all inputs; that is, their head never moves left while computing any input. What is the set of languages accepted by such TMs? Justify your answer.

3. Show that every infinite RE language contains an infinite recursive language as a subset. (*Hint*: enumerate.)

4. Consider the restriction of the Post Correspondence Problem in which the alphabet used for tile strings has only a single symbol, say 0. For example, one possible tile might be [00 | 000]. Show that this *unary PCP* problem is decidable.

5. Some of you might not believe that a TM is as powerful as a real computer. To convince any skeptics, let's consider the following abstract version of an ordinary computer with unbounded resources.

   A *random access machine* (RAM) has an unbounded number of addressable memory cells, each of which can hold a signed integer of arbitrary length. A RAM executes a (fixed) program consisting of a sequence of instructions, each of which is of one of the following four types:

   - "`STR` n a": store the number n at address a
   - "`ADD` a_1 a_2 a_3": add the values stored at memory addresses a_1 and a_2, storing the result at a_3
   - "`BR` a j": if the value at address a is greater than 0, continue execution at instruction j; otherwise, continue with the next instruction.
   - "`HLT`": stop execution.

   Initially, a RAM starts execution from the 0th instruction of its program, and every memory address holds the value 0 except for address 0, which holds an input value. When a RAM halts, its output is the value left at address 0.

   Describe how to simulate a RAM using a Turing machine. More precisely, given a RAM R with a fixed program, describe how to construct a TM M that, on any input, computes the same result as the RAM. M should take R's input on its first tape; when (if) M halts, its first tape should contain only R's output.

   Be sure to describe how the various parts of the RAM's memory are stored on tape(s), and how each of the four instructions is implemented. Give enough detail to show that your implementation is feasible, but not so much as to overwhelm. You may use any “subroutines” we discussed as part of our TM constructions in class.

Note: althrough a RAM seems limited in its instruction set, it can do anything a real computer can do, given the right program. It can add signed numbers and hence can subtract, multiply, and divide (as well as simply move values between memory addresses by adding 0). From these rudiments, one can derive logical bit operations. a RAM can also implement branches with arbitrary arithmetic comparisons (e.g., $x > y$ if $x - y > 0$, and $x = y$ if $x - y + 1 > 0$ but $x - y$ is not > 0).