University of Virginia - cs3102: Theory of Computation Spring 2010

PS5

Due: 6 April 2010 (2:00pm)

This problem set covers material from Chapters 4 and 5. There are eight problems (of which Problem 6 is a challenging bonus question). For full credit, answers must be concise, clear, and convincing, not just correct. Note that this problem set is due one week later that it was scheduled on the original syllabus. This means you will not receive your graded PS5 back in time to review them before Exam 2 starts (take-home exam out on April 8). If you wish to turn in PS5 early (by class on Thursday, 1 April), you will receive your PS5 back by class on April 6.

You are strongly encouraged to use LaTeX to produce your submission, and we have provided a LaTeX template to assist with this. You may, however, handwrite your answers *so long as your writing is legible and easily interpreted.* For full credit, answers must be concise, clear, and convincing, not just correct.

Collaboration Policy. For this assignment, we will follow the same collaboration and resource policy as on Problem Sets 1-4. See the Problem Set 1 handout for a full description of the policy. Use collaboration to help you learn, but not in ways that prevent you from learning to solve problems on your own.

Problem 1: Countable and Uncountable Infinities.

a. Show that the set of all strings in $\{a, b, c\}^*$ is countable.

b. An *algebraic* number is a number that is a root of some equation of the form $c_0 + c_1x + c_2x^2 + \ldots + c_nx^n$ (that is, a value if x that makes the value of the equation 0). For example, ϕ (the golden ratio) is algebraic since it is a root of $x^2 - x - 1$ and $\sqrt{2}$ is algebraic since it is a root of $x^2 - 2$. But, π is not algebraic (proving this is difficult). Show that the set of all algebraic numbers is countable.

c. A *transcendental* number is a real number that is not algebraic. Are the transcendental numbers countable or uncountable? (Support your answer with a convincing argument.)

Problem 2: Language Sizes. Consider the language,

$$BIGGER_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } |L(A)| > |L(B)|\}$$

The notation $|L(M)|$ means the size of the language described by the machine M. The size of a language is the number of strings in the language. For purposes of this question, you should assume the definitions about set sizes from Definition 4.12.

Is $BIGGER_{DFA}$ decidable? Either prove that it is decidable (for example, by providing a high-level description of a Turing Machine that can decide it), or prove that it is undecidable (for example, but showing that a known undecidable problem can be reduced to it).

Problem 3: Closure Properties. For each part, provide a clear **yes** or **no** answer, and support your answer with a brief and convincing proof.

a. If A is a Turing-recognizable language, is the complement of A a Turing-recognizable language?

b. If A is a Turing-decidable language, is the complement of A a Turing-decidable language?

c. If A and B are Turing-recognizable languages, is $A \cap B$ a Turing-recognizable language?

d. If A and B are Turing-decidable language, is $A \cap B$ a Turing-decidable language?

Problem 4: Undecidability. Prove that each of the following languages is undecidable. (Hint: show that you can reduce a known undecidable problem to the problem of deciding the given language.)

a. $L_{INF} = \{< M > | M$ describes a TM that accepts infinitely many strings$\}$

b. $L_{HelloWorld} = \{J | J$ is a Java program that prints out "Hello World"$\}$

Problem 5: Unmodifiable-Input Turing Machine. (Based on a question by Ron Rivest.) Consider a one-tape Turing Machine that is identical to a regular Turing machine except the input may not be overwritten. That is, the symbol in any square that is non-blank in the initial configuration must never change. Otherwise, the machine may read and write to the rest of the tape with no constraints (beyond those that apply to a regular Turing Machine).

$$HALT_{UTM} = \{< M, w > | M \text{ is an unmodifiable-input TM and } M \text{ halts on input } w\}$$

a. What is the set of languages that can be recognized by an unmodifiable-input TM? (Support your answer with a convincing argument.)

b. Is $HALT_{UTM}$ decidable (by a regular TM)? (Support your answer with a convincing proof.)

Problem 6: Self-Rejecting DFAs. (Challenge Bonus) Define the function, $D(w)$, for DFAs similar to the function $M(w)$ we defined for Turing Machines in Class 16:

> $D(w)$ = If w is a valid encoding of a DFA, the DFA described by w. Otherwise, a DFA that rejects all inputs.

Is the language $\textsc{Self-Rejecting}_{DFA} = \{w \in \Sigma^* : w \notin \mathcal{L}(D(w))\}$ Turing-recognizable? Provide a convincing proof supporting your answer.

Problem 7: Random Access Memory. Random access memory (misnamed, since it is not at all random) allows a program to directly reference specified memory locations. Assume each memory location can store a single byte (8 bits) value. A random access machine (RAM) provides (at least) these two instructions:

1. **store *location value*** — write the value represented by the current square on the tape into location *location*. The *location* is any 32-bit integer, and *value* is any 8-bit value (represented by a single square on the tape).
2. **load *location*** — read the value in the location *location* and write it onto the current square on the tape. At the end of a load instruction, the square under the tape head should contain the read value. The read value should be the last value that was stored in *location* (using a **store** instruction), or 0 if no value has been stored in *location*.

a. Prove that a RAM is not more powerful than a Turing machine by showing how a TM could simulate a RAM. Your proof should include high-level descriptions of a Turing Machine that can simulate the **load** and **store** instructions. Your description should also explain how you represent memory on the tape.

b. Is the programming language consisting of just the **load** and **store** instructions above a *universal programming language*? (That is, is it possible to express all possible algorithms using this language.) If it is, prove it (by explaining how you could implement a universal Turing Machine using the **load/store** language. If it is not, explain convincingly why not, and describe the simplest modifications needed to make the language a universal programming language.

Problem 8: Minds and Machines. Many people find the suggestion that a human mind is no more powerful than a Turing Machine to be disturbing, but there appear to be strong arguments supporting this position. For example, consider this argument:

> The brain is a collection of 100 billion neurons. Each neuron is a cell that has inputs (known as *dendrites*) and outputs (synapses that emit neurotransmitter chemicals). The output depends on the inputs in a deterministic way that could be simulated by a Turing Machine. The connections between neurons could also be simulated by a Turing Machine. Since all components of the brain could be simulated by a Turing Machine, the brain itself could be simulated by a Turing Machine. Hence, a human mind is no more powerful than a Turing Machine.

Write a short essay that counters this argument (although many books have been written on this question, you should limit your response to no more than one page). If you reject the premise of this question either because you do not find it disturbing to think of your mind as a Turing Machine, or you feel that the only way to counter this argument is to resort to supernatural (e.g., religious) notions, you may replace this question with Sipser's Problem 5.13.