Name: _____

Exercise A6/B0, 2/8/05

Part 1: Simulating a TM Using a TM

In Module A, we discussed our intuition that the halting problem is semi-decidable and infinite-loop detection is non-TM-recognizable. In either case, it would be impossible to create an algorithm. So, we would not attempt to write a program to solve these problems in the general case. However, by no means we "proved" our intuition. Is our intuition really correct? In Module B, we will explore a more convincing argument behind the intuition, so that we can discuss other (possibly similar or dissimilar) problems with more confidence.

In this exercise, you need to deal with a certain type of "circularity." As a computer science student, you have been exposed to the idea of "recursion." So, you must know that you need to face circularity in a principled manner. As a related example, you might consider the case of "bootstrapping" in compiler construction. Probably the most well-known example is the fact that the C language is "written" in C. Have you heard of it? Did you think about this carefully? If you are interested in, see the Appendix at the end of this exercise. Note: One of the C language developers, Brian Kernighan, will be giving a keynote speech on Sat., Apr. 16, 2005 during Trenton Computer Festival (http://www.tcf-nj.org/), here at TCNJ.

Hint: Try to use schematic diagrams!

Task 1: Suppose that you are given a program *P* written in, say, Java which is claimed to do the following (i.e., for this task, you must accept that this claim holds): given another program *M* in Java (e.g., the source code in ASCII) along with some input *I* to *M*, *P* can determine whether *M* terminates on *I*. First, can you **think of** how *P* might work? Since *M* can be any program, we can supply (the source code of) *P* itself as the input to *P* (i.e., *M* = *P*). What can you **conclude**?

Task 2: To discuss problems such as the halting problem using TMs, we need a way to represent a TM as a string so that it can be placed on the tape of a TM. Then, we can write the representation of a TM *M* along with some input *I* to *M* (e.g., separated by '#') on the tape of some TM *P*. First, **propose** a way to represent a TM as a string. As in **Task 1**, we can supply *P* itself and *I* to *P*. Since we have intuition that this problem is semi-decidable, we would expect that *P* may never terminate on some inputs. Try to **compare** this situation with your analysis in **Task 1**, applying the materials discussed so far in this course.

Part 2: Mini Research Project Ideas

At the beginning of the semester, you were asked to come up with and continue to entertain your own practical problems through the course. While your problems were good starting points, our experience in this course could still be enriched, if we discuss even more examples. So, we will explore some additional problems as mini research projects. This time, you will be asked to identify one or more research questions (from the list below) and write a concise essay about them. Later, you will be asked to do some research activities related to your problems. Note that some of the sample questions are not precisely stated, some of them are probably not solvable, and some of them more or less solved. The instructor will be able to help you find relevant reading for some problems, but not all.

Task: Identify one or more research questions (or some variants of those) from the list below. Write a concise, possibly speculative essay about the question(s), e.g., why you are interested in, what your guessed answer is.

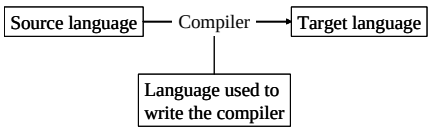
List of sample research questions

1. Can **organizational dynamics** be modeled as an algorithm?
2. Can **evolution** be modeled as an algorithm?
3. Can **ecology** be modeled as an algorithm?
4. Can **human development** be modeled as a computer?
5. Can our **minds** be modeled as a computer?
6. Can **vision** be modeled as an algorithm?

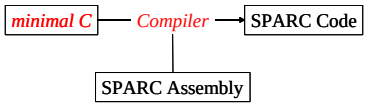
7. Can **learning** be modeled as an algorithm?
8. What would be the minimal mechanism to process **human language**?
9. Can the entire situation of an arbitrary **game** be modeled as an algorithm?
10. Would "perfect" **user modeling**, e.g., for web search, be possible?
11. Would "perfect" **computer security** be possible?
12. Can the entire process of **software engineering** be modeled computationally?
13. Can **computer networks** be modeled as a single computer?
14. Could **biology** be reduced to physics?
15. Could some computer generate real (not pseudo) **random numbers**?
16. Would it be possible to decide whether the given numbers are **random**?
17. Would **randomization** affect computability and/or complexity?
18. Would **parallelism** affect computability and/or complexity?
19. Would **artificial neural network** be more powerful than TMs?
20. Would **cellular automata** be more powerful than TMs?
21. Would the use of **analog** (or fuzzy) values affect computability?
22. Would relativistic, quantum, or some other **modern-physics**-based computation surpass TMs?
23. Can all the cases of **on-line algorithms** be simulated by off-line computation? [On-line algorithms would obtain inputs as the time progress. Off-line computation would provide all the possibilities as input at once. Cf. function-to-relation conversion used to fold the output within the input]
24. Would **oracle computing** affect computability? [Oracle computing: A TM with the capability to ask questions to another mechanism]
25. Would **persistent TM** be able to compute more than the standard TM? [Persistent TM: Multiple sessions of TM operation with some memory between them]
26. Would **accelerating** a TM give more power?
27. Would slight **error tolerance** affect any aspect of the Theory of Computation?
28. What would be the ability of a finite automaton with a **queue**?
29. What would be the effect of "**constant**" (as in complexity analysis) in practice?
30. Can any **mathematical function** be represented computationally?
31. What exactly are **power sets** doing to the Theory of Computation?
32. Is what you can do in **logic** the same as what you can do with computation?
33. If you have a research question of **your own**, please consult the instructor first.

Appendix: Bootstrapping in Compiler Construction

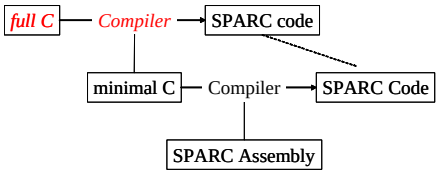
- Languages involved in compiler construction



- Write a compiler for the *minimal* C in SPARC assembly (cf. the "base" case in recursion)



- Write a compiler for *full* C in the minimal C (still generates SPARC code)



Special question (completely optional): Could we "bootstrap" to develop a programming language that is "verifiable" (cf. Ex A3 Part 1)? Those who respond to this question with an *interesting* idea (on the discussion board) will receive a free book (areas: discrete math, logic, theory, algorithms, etc.).

Survey: Time spent between classes: _____

// End