This exam begins for you when you open (or peek inside) this packet. It ends at 9:50 AM on Monday 2014 May 19. Between those two times, you may work on the exam as much as you like. Although I do not intend the exam to require more than a few hours, you should get started early, in case you want to spend more time. The exam is open-book and open-note:

- You may use all of this course's materials: the Sipser textbook, your class notes, your old homework, and the materials on our course web site. If you missed a class and want to get some other student's notes, then do so before either of you begins the exam. You may not share any materials with any other person while you are taking the exam.
- You may cite theorems and examples from class, and from the assigned sections of our textbook. You do not have to reprove them. On the other hand, you may not cite results that we have not studied.
- You may not consult any other books, papers, Internet sites, etc. You may use a computer for viewing the course web site, running Python programs of your own creation, typing up your answers, and e-mailing with me. If you want to use a computer for other purposes, then check with me first.
- You may not discuss the exam in any way — spoken, written, etc. — with anyone but me, until everyone has handed in the exam. During the exam period you will inevitably see your classmates around campus. Refrain from asking even seemingly innocuous questions such as "Have you started the exam yet?" If a statement or question conveys any information about the exam, then it is not allowed. If it conveys no information, then you have no reason to make it.

Feel free to ask clarifying questions in person or over e-mail. You should certainly ask for clarification if you believe that a problem is mis-stated. Check your e-mail occasionally, in case I send out a correction.

Your solutions should be thorough, self-explanatory, neat, concise, and polished. You might want to work on scratch paper, and then recopy your solutions. Alternatively, you might want to type your solutions. Always show enough work and justification so that a typical classmate could understand your solutions. If you cannot solve a problem, then write a brief summary of the approaches you've tried. Partial credit is often awarded. Present your solutions in the order assigned, in a single stapled packet.

Good luck.

Recall that our course web site offers a Python regular expression tutorial that I wrote for CS 254. Recently I have enhanced it, so that it now explains how to use parentheses to influence precedence, without declaring a group to be captured. I use this non-capturing parenthesis feature in Problem A.

In our Interpreter assignment, string literals were delimited by the single quotation mark `'`, and were not allowed to contain any `'`s. But sometimes we want to put `'`s into strings, right? So let's improve the language a bit, by allowing the user to enter a `'` into a string by escaping it with a backslash. For example:

```
:> (set y '\'You are hearing me talk,\' I said.')
''You are hearing me talk,' I said.'
:> (* 2 y)
''You are hearing me talk,' I said.'You are hearing me talk,' I said.'
```

[During the exam, one student pointed out that string literals are now unable to end in a backslash, because that backslash would escape the `'` at the end of the string literal. This is a small error in the specification. The specification should probably add a mechanism for escaping backslashes.]

A1. Describe how to alter your Python regular expression for scanning, so that it correctly captures this more sophisticated notion of string literal. To avoid complicated answers, let's agree that you need no longer worry about checking for mismatched `'` in the scanner; assume that the user delimits string literals with `'`s competently.
A2. Now your scanner is capable of scanning string literals that contain `\'`. But, where that character combination occurs, we don't actually want the backslash to live on in the string literal. We want just the quotation mark to live on. Give Python code that uses another regular expression to convert the `\'`s to `'`s in your string literal tokens.

A *SARF machine* is a tuple $N = (S, A, R, F)$, where $S, A, R \in \{0,1\}^*$ and $F : \{0,1\}^* \to \{0,1\}^*$. The SARF machine takes a bit string w as input. It immediately prepends the string S to w, forming the string Sw. Then it applies F to Sw, obtaining a new string $F(Sw)$. Then it applies F to that string $F(Sw)$, to obtain yet another string $F(F(Sw))$. It keeps applying F to generate new strings from old, until it generates A and accepts or generates R and rejects.

[During the exam, some students asked exactly when the checks for A and R occur. This is a small deficiency in the specification. The string w is not checked against A and R, but every string generated after that is checked: Sw, $F(Sw)$, $F(F(Sw))$, etc.]

B. Prove that for any Turing machine M with input alphabet $\Sigma = \{0,1\}$, there exists an equivalent SARF machine N. Here, "equivalent" means that for every bit string w, N accepts

$\Leftrightarrow M$ accepts, N rejects $\Leftrightarrow M$ rejects, and N runs forever $\Leftrightarrow M$ runs forever.

C. In this problem, all Turing machines are over $\Sigma = \{0, 1\}$ and $\Gamma = \{0, 1, \text{␣}\}$. A Turing machine is said to be *small* if it has fewer than 100 states. Let $SMALL_{TM}$ be the set of all Turing machine encodings $\langle M \rangle$ such that there exists a small Turing machine N with the same language as M. Is $SMALL_{TM}$ decidable? Give a detailed proof.

D. How many hours have you spent on this exam? (Your answer does not affect your score.)