1. Show from first principles that there is no finite state machine that
accepts the language of strings of the form $a^ib^jc^k$ such that $i + j = k$. That is strings of a’s, followed by b’s, followed by c’s, where the
number of c’s is the number of a’s plus the number of b’s. So intuitively
this shows that finite state machines can not ”add” in one obvious
interpretation of ”add”.

Hint: An argument almost identical to the argument that I did in class
that $0^n1^n$ can not be accepted by a finite state machine will work.
Assume that there is a finite automata $M$ with $k$ states that accepts
this language. Consider arbitrarily large strings of some particular
type, and argue it has to mess up on some string of length greater than
$k$. (Do not use the Pumping Lemma, that is not solving the problem
from first principles.)

Due Friday January 8 (You need not use LaTeX, your solutions may
be handwritten.)

2. Show from first principles that there is no finite state machine that
accepts the language of formulas of the form $x + y = z$, where natural
numbers $x$, $y$ and $z$ encoded in binary in the standard way, and where
$x$ plus $y$ is indeed equal to $z$. So for example the input 10100 + 111 =
11011 is in the language. So intuitively this shows that finite state
machines can not ”add” in the most obvious interpretation of ”add”.

Hint: Same basic idea as the previous problem.

Due Friday January 8 (You need not use LaTeX, your solutions may
be handwritten.)

3. Show that there is a finite state machine that can accept the language
over a string of the following 8 symbols (so each line is a different
symbol)

[000]
[001]
[010]
[011]
[100]
[101]
[110]
where the first column plus the second column is equal to the third column. So to reuse our example from problem 1, the 5 symbol input

\[101][001][110][011][011]\]

is in the language because \(10100 + 111 = 11011\).

(a) First show that there is a finite state machine that accepts the language where the low order bits appear first in the input string.

HINT: You do not need to be overly formal in your answer. You just need to make a convincing argument that this language can be accepted using only finite memory.

(b) Then show that there is a finite state machine that accepts the language where the high order bits appear first in the input string.

HINT: It is sufficient to show that if language \(L\) can be accepted by a finite state machine, then the language consisting of the strings in \(L\) reversed can be accepted by a finite state machine. But a simpler argument can be made for this particular language.

4. Problem 1.1 from the text. That is, construct a complete formal description of a Turing machine (using the formalization given in section 1.2) that adds two binary numbers (you need not do multiplication). Assume that input is of the form NUMBER1#NUMBER2, e.g. 1011#110. You should write your output on a second tape.

This will be a painful exercise, but you need to do something like this once to understand what a Turing machine is.

Due Monday January 9.

5. Problem 1.7 from the text. Basically you want to show how one could simulate a Turing Machine with a 2 dimensional tape by a Turing machine with a one dimensional tape.

Giving all details would be extremely painful. So concentrate on giving a convincing argument that such a simulation is possible. You should identify that main issues, and explain how they can be handled.
One purpose of this assignment is to give you more practice thinking about Turing machines. Another purpose is to gain some familiarity with simulation arguments.

Due Monday January 9.

6. Consider a programming language mini-Java that only has one type of loop, and the number of iterations of the loop must be determined when the loop is first encountered. So a loop statement might look like "Repeat n times", and the variable n is evaluated when the statement is first reached. You can assume that the program has a variable Size that is instantiated to the input size when the program starts running (otherwise, you couldn’t even read the input). Note that all mini-Java programs must halt on all inputs. Show by diagonalization that there is a language accepted by a Java program that is not accepted by any mini-Java program. Use Diagonalization. Give a concrete example of a language that is not acceptable by any mini-Java program, but that is accepted by a Java program.

The purpose of assigning this problem is to gain some familiarity with diagonalization arguments.

Hint: You can think of any string as a mini-Java program. If the string is not a syntactically correct Java program, then think of it as a mini-Java program that rejects all inputs. Similarly if the string is a Java program with a loop that isn’t of the right form. First diagonalize over all mini-Java programs to get a language D not accepted by any mini-Java program. Then show that D can be accepted by a Java program.

Due Wednesday January 11.

7. (a) Give a formal/complete description of a one-tape Turing machine $M$ that halts on exactly those strings of 0’s and 1’s where the number of 1’s is odd. Further $M$ should have the property that it has a unique halting configuration where the tape is empty. Keep your Turing machine as simple as possible.

(b) Consider the term rewriting problem that we considered in class and in the notes: The input consists of a start string, a target string, and a collection of transformation rules between strings. The problem is to determine whether the target string can be obtained from the start string by application of the transformation
rules. We showed that this term rewriting problem has no algorithm by reduction from the Halting Problem. Show the term rewriting instance that this reduction would produce given the Turing machine for the previous subproblem and the input 100101.

Due Friday January 13