CS 81 (Spring 2018) **Assignment 7**
Computability and Logic **Due Monday April 2 at 6:00 PM**

Notice: When you submit this assignment, you are certifying therewith that you understand and accept the following policy, which applies to all assignments.

Collaboration Policy: The writeup that you submit must be your own work. You are encouraged to get help from the professor and grutors. You may discuss the problems with classmates, but if you do so, it should be in groups of no more than three. You are not allowed to copy or transcribe solutions from other sources, including the work of other students, the internet, previous solution sets, and images photographed from a whiteboard or blackboard. There is to be no "partnering" where two or more students submit the same writeup. If you get help on a problem, you should say who provided the help on a per-problem basis. Blanket statements such as "worked with John and Mary" are not allowed. Detected infractions may impact your academic career.

Formatting Policy All work must be typeset in electronic media and submitted as a **single pdf file**, one problem on each page as shown in the following pages. Retain this header page. **Handwritten and photographed or scanned work is not allowed.** Do not use inverse video (light typography on dark background). Do not rotate images. **You will not get credit for difficult-to-read submissions.**

Once your document is complete, make a pdf and submit to **Gradescope**.

Caution: Do not start on problems 1 and 2 the day of the deadline. For the first two problems, use the Turing machine simulator that I've provided on Piazza: tm.pro This is a simulator written in Prolog (**Pro**gramming in **Log**ic).

(You can develop in other simulators such as Jflap if you must, but your submitted code must still be in our model.)

There are two ways to run the simulator:

1. Install a Prolog interpreter/compiler on your computer. swipl is a good choice. I used it to develop this simulator. I prefer to run it inside a text editor (emacs) but it can be run bare, e.g. on the command line.
2. Use an online version in a browser. `http://rextester.com/l/prolog_online_compiler` seems like a good choice. It is compatible with swipl and is fast. (swish is not recommended as it is slow.)

[40 points] 1. The n^{th} triangular number satisfies the recurrence

$$t(0) = 0,$$
$$t(n) = n + t(n-1) \text{ for } n > 0.$$

Thus $t(0) = 0$, $t(1) = 1$, $t(2) = 3$, $t(3) = 6$, $t(4) = 10$, $t(5) = 15$, etc.

Design a Turing transducer **`tri1`** that behaves as follows:

Given an initial tape with n consecutive `1`s, **`tri1`** modifies the tape to halt with exactly $t(n)$ consecutive `1`s. Thus, if the initial tape is

```
11111
```

then the final tape is

```
111111111111111
```

since $t(5) = 15$.

Assume the blank symbol is `b`. You may use other symbols as needed. Assume that the head is intially at the left character of the input. In the case of $n = 0$, the entire tape will be blank.

Submit the code for **`tri1`** and a trace of **`tri1`** running on input `111`. Also show the outputs without tracing for inputs 0 to 6 `1`s.

Here is a test setup you can use (`resultsInRange` was added to `tm.pro` at 2:00 PM on Tuesday, March 27, 2018):

```
tri1(tm(... your definition of tri1 ...)).

test_tri1 :- tri1(Tm),
             run(Tm, [1, 1, 1]),
             resultsInRange(Tm, 0, 6).
```

[30 points] 2. Design a Turing decider **tri2** that decides the language T of strings in $\{1\}^\star$ that are triangular numbers. That is,

$$T = \{\varepsilon, 1, 111, 111111, 1111111111, 111111111111111\}$$

If an input is an elements of T then **tri2** accepts the input, otherwise it rejects the input.

Submit the code for **tri2**, a trace of **tri2** running on input and a trace of **tri2** running on input . Also show accepted inputs, without tracing for inputs with lengths in range 0 to 21.

Here is a testing setup you can use:

```
tri2(tm(... your definition of tri2 ...)).

test_tri2 :-
    tri2(Tm),
    run(Tm, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
    run(Tm, [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]),
    acceptsInRange(Tm, 0, 21).
```

[15 points] 3. Decidable languages, i.e. languages for which there is a Turing decider, are closed *under intersection*. This means that, if L_1 and L_2 are decidable, then $L_1 \cap L_2$ is also decidable. Describe in sufficient detail to be convincing how to construct a Turing decider for $L_1 \cap L_2$ from Turing deciders for L_1 and L_2. Focus on how the rules for the decider for $L_1 \cap L_2$ would be constructed.

[15 points] 4. Recognizable languages, i.e. languages for which there is a Turing recognizer, are closed *under union*. This means that, if L_1 and L_2 are recognizable, then $L_1 \cup L_2$ is also recognizable. Use the Church–Turing thesis to argue why this is true. Under this thesis, you do not have to give the construction details of a Turing machine. Instead you can just present a convincing informal algorithm for recognizing $L_1 \cup L_2$. (Hint: The key issue here is that either of the recognizers for L_1 or L_2 could diverge on an input x. So if you simply run the machines in one order or the other, then the first one run could diverge while the second one might not. So it might be concluded that an input string x is not in $L1 \cup L2$ when in fact it is. So you must describe how to protect against such divergence.)
