

Due: By 2pm on Monday 15 October.

Worth: 10%

This assignment is to be completed in groups of no more than three students. Hand in a single paper for your group, with the information about each student filled in on the cover page.

1. [15 marks]

In “balanced ternary” notation, positive and negative integers are represented using the “digits” $-1, 0, +1$, as follows: the representation of integer x consists of the sequence of digits $a_k, a_{k-1}, \dots, a_1, a_0$ such that

$$x = a_k 3^k + a_{k-1} 3^{k-1} + \dots + a_1 3^1 + a_0 3^0.$$

For example, $-5 = -9 + 3 + 1$ so the representation of -5 is $-++$ (using “-” and “+” to stand for -1 or $+1$, respectively). For another example, the representation of 10 is $+0+$.

It is relatively straightforward to work out how to add two numbers written in balanced ternary notation, using a standard right-to-left sweep of the digits. For example, $-++ + +0+ = +--$ because

$$\begin{array}{r} 0++ \quad (\text{carry}) \\ -++ \\ + +0+ \\ \hline 0+-- \end{array}$$

For this question, your task is to implement addition of balanced ternary numbers on a Turing machine.

More specifically, write a 3-tape Turing machine with input alphabet $\Sigma = \{-, 0, +\}$ and with the following behaviour: when started with input strings x and y on its first and second tapes and its third tape blank, with all tape heads on the leftmost squares, your TM performs the addition of x and y and will eventually accept with the value of $x + y$ written in balanced ternary notation on its third tape (the final contents and head positions on the first two tapes don’t matter).

Give a high-level description (one short paragraph giving the main idea(s) of your algorithm), an implementation-level description (broken down into numbered stages that describe head movements and tape contents in more detail), and a formal-level description (the input and tape alphabets, list of states, and full transition table, broken down into sections with comments to explain the purpose of groups of states and relate them to the implementation-level stages).

To convince yourself (and us) that your TM is correct, trace the computation of your TM on sample inputs (including boundary cases such as x or y being empty).

2. [10 marks]

Prove that the halting problem for Python programs is undecidable. More precisely, show that there is no Python function `halt(P, x)` such that for all Python functions P and inputs x , `halt(P, x)` returns `True` if $P(x)$ returns a value; `halt(P, x)` returns `False` if $P(x)$ does not return (loops).

Note that you cannot use Church’s Thesis to answer this question, because an appeal to Church’s Thesis does not constitute a proof.

3. [15 marks]

State whether or not the following language is decidable, and whether or not it is recognizable, then prove your claims.

$$L_1 = \{ \langle P \rangle : P \text{ is a Python program with one input } x, \text{ and } P \text{ returns a value for some input} \}$$

4. [15 marks]

State whether or not the following language is decidable, and whether or not it is recognizable, then prove your claims.

$$L_2 = \{ \langle P \rangle : P \text{ is a Python program with one input } x, \text{ and } P \text{ returns a value for all inputs} \}$$

5. [10 marks]

A real number r is “constructible” if there is a two-tape Turing machine which will write the digits in the decimal expansion of r on its second tape, one after the other.

For example, all rational numbers are constructible (because we can simply carry out long division of the numerator by the denominator), as are the irrational numbers e , π , $\sqrt{2}$ (or any other rational power of a rational number). In fact, any number for which there is an algorithm to compute the value is constructible, by Church’s Thesis.

How many constructible real numbers are there? Prove your claim.