Monash University
Faculty of Information Technology
2nd Semester 2023

FIT2014
Exercises 9
Undecidability and recursively enumerable languages

Although you may not need to do all the exercises in this Tutorial Sheet, it is still important that you attempt all the main questions and a selection of the Supplementary Exercises.

ASSESSED PREPARATION: Question 3.
You must provide a serious attempt at this entire question at the start of your class.

---

1. Prove that the following problem is undecidable, using a mapping reduction from the Diagonal Halting Problem.

HALT ON EVEN STRINGS
INPUT: Turing machine M.
QUESTION: Does M always halt when the input string has even length?

2.

Recall the languages TM ACCEPTANCE and FINITE AUTOMATON ACCEPTANCE from Exercise Sheet 8, Question 5.

Does there exist a mapping reduction from TM ACCEPTANCE to FINITE AUTOMATON ACCEPTANCE? Why or why not?

3.

A **two-way infinite tape** (TWIT) is a Turing machine tape that is infinite in both directions, i.e., infinite in both left and right directions. The tape cells may be numbered by the integers instead of just by the *positive* integers. When an input string x of n letters is placed on the tape, it is still placed on the first n tape cells in the positive direction, i.e., tape cells 1 to n. Initially, all other tape cells are blank, in both directions.

A **TWIT Turing machine** is just like a normal Turing machine except that it uses a TWIT. There is therefore no possibility of the tape head crashing off the left end of the tape. When the machine accepts, the output string is taken to be the string on the positive portion of the tape, up to but not including the first blank on the positive side, at the time the machine halts.

(a)

Given a normal Turing machine, how would you construct a TWIT Turing machine that simulates it and, in particular, has the same eventual outcome (Accept/Reject/Loop) in all cases and computes the same function?

Now consider the following problem.

TWIT Halting Problem
INPUT: A TWIT Turing machine M, a string x
QUESTION: When M is run with input x, does it eventually halt?

(b)

Give a mapping reduction from the Diagonal Halting Problem to the TWIT Halting Problem.

(c)

Is the TWIT Halting Problem decidable or undecidable? Justify your answer.

(d)

Is the TWIT Halting Problem recursively enumerable? Justify your answer.

4.

For each of the following decision problems, indicate whether or not it is decidable.

You may assume that, when Turing machines are encoded as strings, this is done using the Code-Word Language (CWL).

Decision Problem	your answer (tick **one** box in each row)	
Input: a Java program P (as source code). Question: Does P contain an infinite loop?	☐ Decidable	☐ Undecidable
Input: a Java program P (as source code). Question: Does P contain the infinite loop `for (int i=0; i>=0; );` ?	☐ Decidable	☐ Undecidable
Input: a Java program P (as source code). Question: Does P contain recursion?	☐ Decidable	☐ Undecidable
Input: a Java program P (as source code). Question: If P is run, will some method of P keep calling itself forever?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: is there some input string, of length **at most** 12, for which M halts in at most 144 steps?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: is there some input string, of length **at least** 12, for which M halts in at most 144 steps?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Does M halt on an even number of steps?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Is there a palindrome which, if given as input, causes M to eventually halt?	☐ Decidable	☐ Undecidable
Input: a Turing machine M. Question: Is M a palindrome?	☐ Decidable	☐ Undecidable

5. Define the function $f : \mathbb{N} \to \mathbb{N}$ by

$$f(x) = \begin{cases} 3x+1, & \text{if } x \text{ is odd,} \\ x/2, & \text{if } x \text{ is even,} \end{cases}$$

for all $x \in \mathbb{N}$.

We consider *iteration* of f. Given a positive integer x, form the sequence

$$x \longrightarrow f(x) \longrightarrow f(f(x)) \longrightarrow f(f(f(x))) \longrightarrow f(f(f(f(x)))) \longrightarrow \cdots$$

For example, starting with $x = 3$, we have

$$3 \longrightarrow 10 \longrightarrow 5 \longrightarrow 16 \longrightarrow 8 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1 \longrightarrow 4 \longrightarrow \cdots,$$

so it eventually cycles forever through the numbers 4,2,1.

Let COLLATZ be the language of all positive integers for which repeated application of f eventually produces 1. The above example shows that $3 \in$ COLLATZ, and in fact that $\{1, 2, 3, 4, 5, 8, 10, 16\} \subseteq$ COLLATZ.

(a) Using any computational tool of your choice, compute the sequence of iterates of f that begins with $x = 27$, until it starts repeating. (Explain your method briefly, in your submission.)

(b) Prove that COLLATZ is recursively enumerable.

(c) Is your algorithm for (b) a decider for COLLATZ? Why or why not?

(d) Can you prove anything more specific about the class of languages to which COLLATZ belongs? For example, is it decidable?[1]

[1] If you can answer (d), you may become famous! The *Collatz conjecture* (currently unproved) is that COLLATZ = $\mathbb{N}$. In other words, the conjecture asserts that, whatever number you start with, iterating f eventually reaches 1. The conjecture is attributed to Lothar Collatz in 1937 and has attracted a huge amount of human attention and computer time over the decades since then. See, e.g., Jeffrey C. Lagarias, The 3x + 1 problem and its generalizations, *American Mathematical Monthly* **92** (1) 3–23; `https://doi.org/10.1080/00029890.1985.11971528`. Some of the most significant progress in recent years has been made by the Australian mathematician Terence Tao. See: Kevin Hartnett, Mathematician Proves Huge Result on 'Dangerous' Problem, *Quanta Magazine*, 11 December 2019; `https://www.quantamagazine.org/mathematician-terence-tao-and-the-collatz-conjecture-20191211/`. Currently, the conjecture seems far out of reach: it is not even known if COLLATZ is *decidable*.

6.

Prove the following theorem (stated in Lecture 23, slide 19):
A language L is r.e. if and only if there is a decidable two-argument predicate P such that:

$$x \in L \text{ if and only if there exists } y \text{ such that } P(x, y). \quad [2]$$

7.

Give an example of a co-r.e., non-r.e. problem about each of the following:

(a) Turing machines

(b) Context-free grammars

(c) Polynomials and their roots

Supplementary exercises

8. If x is any string, we can *double* it by concatenating it with a copy of itself, forming the string xx. If x is any string of even length, then we can *halve* it by dividing it in half exactly, giving two strings x^L and x^R, being the left and right halves, respectively, of x. (So $x = x^L x^R$.)

A language is *closed under doubling* if every string formed by doubling a string in the language is also in the language. A language is *closed under halving* if halving a string in the language always gives two other strings that are also in the language.

Let L be any nonempty non-universal[3] language that is closed under doubling and halving. Let L^{even} be the set of all strings of even length in L.

(a) Prove that there exists a mapping reduction from L to L^{even}.

(b) Prove that there exists a mapping reduction from L^{even} to L.

(c) Prove that L is undecidable if and only if L^{even} is undecidable.

(d) Determine the time complexity, in big-O notation, of the mapping reductions you found in (a) & (b).

[2] A predicate is said to be *decidable* if its corresponding function — from the set of all its inputs to {True, False}, always correctly indicating whether or not the predicate is True for its input — is computable.

[3] The *universal* language is Σ^*, i.e., the set of all finite strings over the alphabet being used.

9.

A program is *self-reproducing* if it outputs a copy of itself and then stops. Self-reproducibility is a property of programs such as computer viruses and worms.

Prove that the language of self-reproducing programs is undecidable.

Assumptions you may make:

- All programs under discussion are written in some fixed programming language such as Python or Java.
- The Diagonal Halting Problem, for programs in this language, is undecidable.
- There exists a specific self-reproducing program S with the property that you can insert a code chunk C (of a certain type, to be explained shortly) into S, at some special point in S, so as to get another self-reproducing program S_C which executes C before reproducing itself. The inserted code chunk C is required to be "independent" of all the code in S, in that it has uses no variables, functions, methods, etc. that are used in S.

What does this mean for the possibility of an infallible virus-testing program?

10.

For any Turing machine U whose input is a pair (e, x) of strings, we say that U is *outwardly universal* if there is a computable function f which takes any Turing machine and produces a string (intended to be an encoding of the TM, which might be in the Code-Word Language, or might use some other scheme) such that, for all Turing machines M and all strings x,

- U halts for input $(f(M), x)$ if and only if M halts for input x, and
- when they halt, they produce the same output.

Observe that this definition is broader than the usual definition of UTMs! This is not only because we allow any computable encoding scheme, and not just the one used in lectures. It is mainly because we do not require step-by-step simulation, but only that the results of the computations be the same.

Prove that determining whether or not a Turing machine is outwardly universal is undecidable.