Computation Theory

Supervision 2

5. The Halting Problem and undecidability

1. Show that decidable sets are closed under union, intersection, and complementation. Do all of these closure properties hold for undecidable languages?
2. Suppose S_1 and S_2 are subsets of $\mathbb{N}$. Suppose $r \in \mathbb{N} \to \mathbb{N}$ is a register machine computable function satisfying: for all n in $\mathbb{N}$, n is an element of S_1 if and only if $r(n)$ is an element of S_2. Show that S_1 is register machine decidable if S_2 is. Is the converse, inverse, or contrapositive of this statement true?
3. Show that the set E of codes $\langle e, e' \rangle$ of pairs of numbers satisfying $\varphi_e = \varphi_{e'}$ is undecidable.
4. Show that there is a register machine computable partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ such that both sets $\{ n \in \mathbb{N} \mid f(n)\downarrow \}$ and $\{ y \in \mathbb{N} \mid \exists n \in \mathbb{N}.\, f(n) = y \}$ are register machine undecidable.

6. Turing machines

1. Compare and contrast register machines with Turing machines: how do they keep track of state, how are programs represented, what form do machine configurations and computations take?
2. Familiarise yourself with the Chomsky hierarchy and explain the connection between regular expressions and Turing machines.

7. Notions of computability

1. Before the formal development of the field of computation theory, mathematicians often used the term *effectively computable* to describe functions that can – in principle – be computed using mechanical, pen-and-paper methods.
   a) How was the notion of effective computability formalised by Church and Turing, and generalised to other models of computation?
   b) Suppose we invented a new model of computation. How can we establish that it is as "powerful" as mechanical methods? Make sure to formally explain what "power" means in this case.
   c) Can our new model be even more powerful?
2. Briefly describe of three Turing-complete models of computation not covered in the course.

8. Partial recursive functions

1. Show that the following functions are all primitive recursive. Make sure to give the final form of the function as a composition of primitive functions and projection.

a) Truncated subtraction function, $minus\colon \mathbb{N}^2 \to \mathbb{N}$, where

$$minus(x,y) \triangleq \begin{cases} x-y & \text{if } y \leq x \\ 0 & \text{if } y > x \end{cases}$$

b) Exponentiation, $exp\colon \mathbb{N}^2 \to \mathbb{N}$, where $exp(x,y) = x^y$.

c) Conditional branch on zero, $ifzero\colon \mathbb{N}^3 \to \mathbb{N}$, where

$$ifzero(x,y,z) \triangleq \begin{cases} y & \text{if } x = 0 \\ z & \text{if } x > 0 \end{cases}$$

d) Bounded summation: if $f : \mathbb{N}^{n+1} \to \mathbb{N}$ is primitive recursive, then so is $g\colon \mathbb{N}^{n+1} \to \mathbb{N}$ where where

$$g(\vec{x},x) \triangleq \begin{cases} 0 & \text{if } x = 0 \\ f(\vec{x},0) & \text{if } x = 1 \\ f(\vec{x},0) + \cdots + f(\vec{x},x-1) & \text{if } x > 1 \end{cases}$$

2. Explain the motivation and intuition behind *minimisation*. How does it extend the set of functions computable using primitive recursion? Give three examples of computable partial functions that are not definable using primitive recursion, justifying your answer in each case.

3. Use minimisation to show that the following functions are partial recursive:

   a) the binary maximum function $max\colon \mathbb{N}^2 \to \mathbb{N}$.

   b) the integer square root function $sqrt\colon \mathbb{N} \rightharpoonup \mathbb{N}$ which is only defined if its argument is a perfect square.

Optional exercises

1. Write a Turing machine simulator in a programming language you prefer (a functional language such as ML or Haskell is recommended). Implement the machine described on Slide 64.

2. For the example Turing machine given on Slide 64, give the register machine program implementing $(S,T,D) := \delta(S,T)$, as described on Slide 70.

3. Recall the definition of Ackermann's function ack. Sketch how to build a register machine M that computes $ack(x_1,x_2)$ in R_0 when started with x_1 in R_1 and x_2 in R_2 and all other registers zero. (E9)

   Hint: Call a finite list $L = [(x_1,y_1,z_1),(x_2,y_2,z_2),\ldots]$ of triples of numbers *suitable* if it satisfies

   a) if $(0,y,z) \in L$, then $z = y+1$

   b) if $(x+1,0,z) \in L$, then $(x,1,z) \in L$

   c) if $(x+1,y+1,z) \in L$, then there is some u with $(x+1,y,u) \in L$ and $(x,u,z) \in L$.

The idea is that if $(x, y, z) \in L$ and L is suitable then $z = ack(x, y)$ and L contains all the triples $(x', y', ack(x, y'))$ needed to calculate $ack(x, y)$. Show how to code lists of triples of numbers as numbers in such a way that we can (in principle, no need to do it explicitly!) build a register machine that recognises whether or not a number is the code for a suitable list of triples. Show how to use that machine to build a machine computing $ack(x, y)$ by searching for the code of a suitable list containing a triple with x and y in its first two components.