Exercises and Tripos Questions

A course on *Computation Theory* has been offered for many years. Since 2009 the course has incorporated some material from a Part IB course on *Foundations of Functional Programming* that is no longer offered. A guide to which Tripos questions from the last five years are relevant to the current course can be found on the course web page (follow links from `www.cl.cam.ac.uk/teaching/`). Here are suggestions for which of the older ones to try, together with some other exercises.

1. Exercises in register machine programming:
   (a) Produce register machine programs for the functions mentioned on slides 36 and 37.
   (b) Try Tripos question 1999.3.9.

2. Undecidability of the halting problem:
   (a) Try Tripos question 1995.3.9.
   (b) Try Tripos question 2000.3.9.
   (c) Learn by heart the poem about the undecidability of the halting problem to be found at the course web page and recite it to your non-compsci friends.

3. Let ϕ_e denote the unary partial function from numbers to numbers (i.e. an element of $\mathbb{N} \rightharpoonup \mathbb{N}$—cf. slide 30) computed by the register machine with code e (cf. slide 63). Show that for any given register machine computable unary partial function f, there are infinitely many numbers e such that $\phi_e = f$. (Equality of partial functions means that they are equal as sets of ordered pairs; which is equivalent to saying that for all numbers x, $\phi_e(x)$ is defined if and only if $f(x)$ is, and in that case they are equal numbers.)

4. Suppose S_1 and S_2 are subsets of the set $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$ of natural numbers. Suppose $f \in \mathbb{N} \to \mathbb{N}$ is register machine computable and satisfies: for all x in $\mathbb{N}$, x is an element of S_1 if and only if $f(x)$ is an element of S_2. Show that S_1 is register machine decidable (cf. slide 66) if S_2 is.

5. Show that the set of codes $\langle e, e' \rangle$ of pairs of numbers e and e' satisfying $\phi_e = \phi_{e'}$ is undecidable.

6. For the example Turing machine given on slide 75, give the register machine program implementing
$$(S, T, D) := \delta(S, T)$$
as described on slide 83. [Tedious!—maybe just do a bit.]

7. Try Tripos question 2001.3.9. [This is the Turing machine version of 2000.3.9.]

8. Try Tripos question 1996.3.9.

9. Show that the following functions are all primitive recursive.
   (a) *Exponentiation*, $exp(x, y) \triangleq x^y$.

(b) *Truncated subtraction*, $minus(x,y) \triangleq \begin{cases} x - y & \text{if } x \geq y \\ 0 & \text{if } x < y \end{cases}$

(c) *Conditional branch on zero*, $ifzero(x,y,z) \triangleq \begin{cases} y & \text{if } x = 0 \\ z & \text{if } x > 0 \end{cases}$

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

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

10. Recall the definition of Ackermann's function *ack* from slide 122. Sketch how to build a register machine M that computes $ack(x_1, x_2)$ in $R0$ when started with x_1 in $R1$ and x_2 in $R2$ and all other registers zero. [Hint: here's one way; the next question steers you another way to the computability of *ack*. 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

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

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

    (iii) 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 recognizes 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 it's first two components.]

11. If you are not already fed up with Ackermann's function, try Tripos question 2001.4.8.

12. If you are *still* not fed up with Ackermann's function $ack \in \mathbb{N}^2 \rightarrow \mathbb{N}$, show that the λ-term $\mathsf{ack} \triangleq \lambda x.\, x\,(\lambda f\, y.\, y\, f\,(f\,\underline{1}))\,\mathsf{Succ}$ represents *ack* (where Succ is as on slide 152).

13. Let I be the λ-term $\lambda x.\, x$. Show that $\underline{n}\mathsf{I} =_\beta \mathsf{I}$ holds for every Church numeral $\underline{n}$. Now consider

    $$\mathsf{B} \triangleq \lambda f\, g\, x.\, g\, x\, \mathsf{I}\,(f\,(g\, x))$$

    Assuming the fact about normal order reduction mentioned on slide 145, show that if partial functions $f, g \in \mathbb{N} \rightharpoonup \mathbb{N}$ are represented by closed λ-terms F and G respectively, then their composition $(f \circ g)(x) \equiv f(g(x))$ is represented by $\mathsf{B}\, F\, G$. Now try Tripos question 2005.5.12.