P

Lecture Notes on

Computation Theory

for the

Computer Science Tripos IB

Andrew M. Pitts
University of Cambridge
Computer Laboratory

First edition 1995.
Revised 2002, 2003, 2005, 2006, 2007, 2008.

Contents

Learning Guide

These notes are designed to accompany 12 lectures on computation theory for Parts IB and II(G) of the Computer Science Tripos and the Diploma in Computer Science. The aim of this course is to introduce several apparently different formalisations of the informal notion of algorithm; to show that they are equivalent; and to use them to demonstrate that there are uncomputable functions and algorithmically undecidable problems. At the end of the course you should:

- be familiar with the register machine and Turing machine models of computability
- understand the notion of coding programs as data, and of a universal machine
- be able to use diagonalisation to prove the undecidability of the Halting Problem
- understand the mathematical notion of partial recursive function and its relationship to computability
- be able to develop simple mathematical arguments to show that particular sets are not recursively enumerable.

The prerequisites for taking this course are the Part IA courses *Discrete Mathematics* and *Regular Languages and Finite Automata*.

This *Computation Theory* course contains some material that **everyone who calls themselves a computer scientist should know**. It is also a prerequisite for the Part IB course on *Complexity Theory*.

Exercises and Tripos Questions

A course on Computation Theory has been offered for many years and so there are many past Tripos questions to try. See ⟨`www.cl.cam.ac.uk/tripos/t-ComputationTheory.html`⟩ for a list of them; all the questions from the last five years are relevant to the present course. 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 page 24.
   (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 $\mathrm{Pfn}(\mathbb{N}, \mathbb{N})$) computed by the register machine with code e (cf. page 52). 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 $f \in \mathrm{Fun}(\mathbb{N}, \mathbb{N})$ 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 if S_2 is. (Cf. page 58-1.)

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 page 68, give the register machine program implementing
$$(S, T, D) := \delta(S, T)$$
as described on page 71. [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. Work out explicit descriptions that demonstrate that the functions defined on pages 97 and 98 (predecessor, truncated subtraction, conditional and bounded summation) are all primitive recursive.

10. Define
$$div(x, y) = \begin{cases} \text{integer part of } x/y & \text{if } y > 0 \\ 0 & \text{otherwise} \end{cases}$$
(see page 109). Show that div is primitive recursive. [Hint: use Example 7, page 98); cf. page 122.]

11. Recall the definition of Ackermann's function ack from page 112. Sketch how to build a register machine M that computes $ack(x, y)$ in $R0$ when started with x in $R1$ and y 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.]

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

13. Try Tripos question 1997.4.8.

Recommended books

- Hopcroft, J.E., Motwani, R. & Ullman, J.D. (2001). *Introduction to Automata Theory, Languages and Computation, Second Edition*. Addison-Wesley.

- Cutland, N.J. (1980) *Computability. An introduction to recursive function theory*. CUP.

- Davis, M.D., Sigal, R. & Wyuker E.J. (1994). *Computability, Complexity and Languages*, 2nd edition. Academic Press.

- Sudkamp, T.A. (1995). *Languages and Machines*, 2nd edition. Addison-Wesley.

- Jones, N.D. (1997). *Computability and Complexity*. MIT Press.

Note

The material in these notes has been drawn (by hand!) from several different sources, including the books mentioned above, previous versions of this course by the author and by others, and similar courses at some other universities. Any errors are of course all the author's own work. A list of corrections will be available from the course web page (follow links from ⟨`www.cl.cam.ac.uk/Teaching/`⟩). Please take time to fill out a lecture(r) appraisal form via the URL ⟨`https://lecture-feedback.cl.cam.ac.uk/feedback`⟩.

Andrew Pitts
⟨`amp12@cl.cam.ac.uk`⟩