Computation Theory Example Sheet 1

Lecture 1

Exercise 1 What are decision problems?

Exercise 2 Define the *Halting problem*. Explain the informal argument of why the Halting problem is undecidable.

Lecture 2

Exercise 3

(a) Define a *register machine (RM)*. (See **[2010P6Q3 (a)]** or **[2007P3Q7 (a)(i)]**)

(b) Define a *register machine configuration*. (See **[2009P6Q3 (a)]** or **[1999P3Q9 (b)]**)

(c) Define a *register machine computation* (See **[2010P6Q3 (a)]** or **[1999P3Q9 (a)]**). What do we mean when we say that the execution of an RM is *deterministic*?

(d) What are the two ways that an RM *halts*?

(e) How can you modify a program to turn all erroneous halts into proper halts?

Exercise 4 [RM graphical representation] Define the graphical representation for the RM programs.

Exercise 5

(a) Define a *partial function*.

(b) Why is the relation between initial and final register contents of a RM a partial function?

(c) Define a *total function*.

(d) Show that a total function is always a partial function.

(e) Define the notations $f(x)\downarrow$, $f(x)\uparrow$, $X \to Y$ and $X \rightharpoonup Y$.

(f) Give an example of an RM program that is a total function and an RM program that is partial (but not total).

Exercise 6 [RM Computable]

(a) What does it mean for a function to be *RM computable*? (See **[2013P6Q3 (b)]** or **[2010P6Q3 (b)]** or **[2007P3Q7 (b)(i)]** or **[2005P3Q7 (c)]**)

(b) What is the *IO convention*?

Exercise 7 In this exercise, you will investigate a simple register machine emulator.

(a) In the "rm.zip" file execute the "Examples.java" file and see the execution for an add and a copy program.

(b) Write a similar program for multiplying two integers and execute it. (Use the `debug` flag to output the configuration sequence and the flag `maxIter` to set an upper bound on the computation steps).

Exercise 8 (optional) In this exercise, you can experiment and implement various extensions to the RM emulator (of course you can create your own from scratch).

(a) Add an instruction that adds two register values and stores the result in the third one. (This is mostly to understand how the current emulator works, you will not need this for subsequent steps)

(b) Create a function that takes a program and a mapping for the registers (e.g. $\{R_1 \to R_2, R_2 \to R_4, R_5 \to R_3\}$) and returns a new program where operations are performed on the mapped registers.

(c) Write a function that concatenates various programs together, so that one executes after the other. Note: You need to change the labels and also replace HALT instructions.

(d) Use the previous two operations to create a program that uses copy (defining it once) and add.

(e) Write a function that takes three programs M_1, M_2 and M_3 and returns a program that implements if M_1 then M_2 else M_3. (You need to define its semantics)

(f) Write a function that takes two programs M_1 and M_2 and returns a program that implements while M_1 do M_2.

(g) Write a function that takes a program M and three registers and implements for $R_i = R_1$ to R_2 do M.

(h) Write a program that computes the Fibonacci numbers using the high-level constructs.

(i) (optional - more of Compiler Design) Implement a parser for reading RM programs and converting them to abstract syntax trees (which are just lists in this case).

Exercise 9 [High-level constructs] Show that the following high-level constructs can be implemented using RMs.

(a) (sequential composition) If M_1 and M_2 are programs, then there exists a program M_3 that is equivalent to first executing M_1 and then M_2.

(b) (if-then-else statements) If M_1, M_2 and M_3 are programs, then there exists a program M_4 that is equivalent to **if** M_1 **then** M_2 **else** M_3. (You need to define the exact semantics for if)

(c) (while-do) If M_1 and M_2 are programs, then there exists a progam M_3 that is equivalent to **while** M_1 **do** M_2. (Again, you need to define the exact semantics for while)

(d) (optional) Similarly, define a program that is equivalent to a for-loop.

(e) Give a high-level argument for why any function computable using a high-level programming language is computable using a RM. Is the other direction true?

Exercise 10 [Projection] Show that the *first projection* function $p_1^2 : \mathbb{N}^2 \to \mathbb{N}$, where $p_1^2(x, y) \triangleq x$, is RM computable. Write the RM program and create a diagram for this. Similarly, argue that the *second projection* function $p_2^2(x, y) \triangleq y$ is RM computable.

Exercise 11 [Constant] Show that the *constant* function $c : \mathbb{N} \to \mathbb{N}$, where $c(x) \triangleq n$ for fixed $n \in \mathbb{N}$ is RM computable. Write the RM program and create a diagram for this.

Exercise 12 [Addition] Show that the *addition* function $\text{add} : \mathbb{N}^2 \to \mathbb{N}$, where $\text{add}(x, y) \triangleq x + y$ is RM computable. Write the RM program and create a diagram for this.

Exercise 13 [Multiplication] Show that the *multiplication* function $\text{mult} : \mathbb{N}^2 \to \mathbb{N}$, where $\text{mult}(x, y) \triangleq x \cdot y$ is RM computable. Write the RM program and create a diagram for this. (See **[2007P3Q7 (b)(ii)]** for variant)

Exercise 14 [Max] Show that the *max* function $\max : \mathbb{N}^2 \to \mathbb{N}$ is RM computable. Write the RM program and create a diagram for this. (See **[2018P6Q6 (b)(ii)]**)

Exercise 15 [Truncated subtraction] Show that the *truncated subtraction* function $\text{tsub} : \mathbb{N}^2 \to \mathbb{N}$,

where

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

is RM computable. Write the RM program and create a diagram for this. (See **[2010P6Q3 (e)]** for variant)

Exercise 16 [Comp] Show that the *comp* function $\text{comp} : \mathbb{N}^2 \to \mathbb{N}$, where

$$\text{comp}(x, y) \triangleq \begin{cases} 0 & \text{if } x \leq y \\ 1 & \text{otherwise} \end{cases}$$

is RM computable. Write the RM program and create a diagram for this. (See **[2018P6Q6 (b)(iii)]**)

Exercise 17 [Undef] Show that the *undef* function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is RM computable. (See **[2013P6Q3 (c) (i)]**)

Exercise 18 [Integer division] Show that the *integer division* function $\text{div} : \mathbb{N}^2 \to \mathbb{N}$, where

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

is RM computable. Write the RM program and create a diagram for this.

Exercise 19 [Mod] Show that the *mod* (basically $[\cdot]$. as defined in Discrete Maths) function $\text{mod} : \mathbb{N}^2 \to \mathbb{N}$, where

$$\text{mod}(x, y) \triangleq \text{tsub}(x, y \cdot (\text{div}(x, y))$$

is RM computable. Outline the construction of an RM program that computes this function.

Exercise 20 [Binary Exponential] Show that the *binary exponential* function $e : \mathbb{N} \to \mathbb{N}$, where

$$e(x) \triangleq 2^x$$

is RM computable. Write the RM program and create a diagram for this. (See **[2013P6Q3 (c) (iii)]** for variant)

Exercise 21 [Exponentiation] Show that the *exponentiation* function $\text{pow} : \mathbb{N}^2 \to \mathbb{N}$, where

$$\text{pow}(x, y) \triangleq x^y$$

is RM computable. Outline the construction of an RM program that computes this function.

Exercise 22 [Binary Logarithm] Show that the *binary logarithm* function $\log_2 : \mathbb{N} \to \mathbb{N}$, where

$$\log_2(x) \triangleq \begin{cases} \text{largest } y \text{ such that } 2^y \leq x & \text{if } x > 0 \\ 0 & \text{if } x = 0 \end{cases}$$

is RM computable. Write the RM program and create a diagram for this.

Exercise 23 [Fibonacci Numbers] Show that the *Fibonacci number* function $\mathrm{Fib} : \mathbb{N} \to \mathbb{N}$, where

$$\mathrm{Fib}(n) \triangleq \begin{cases} \mathrm{Fib}(n-1) + \mathrm{Fib}(n-2) & \text{if } n \geq 2 \\ n & \text{otherwise} \end{cases}$$

is RM computable. Outline the construction of an RM program that computes this function.

Exercise 24 [Boolean logic] Show that the binary logic functions **and**, **or** and **xor** are computable.

Exercise 25 [Reverse engineer the program 1] Attempt **[2010P6Q3 (d)]**.

Exercise 26 [Reverse engineer the program 2] Attempt **[1999P3Q9 (c)]** (x' means x^+).

Lecture 3

Exercise 27 [Numerical codings of pairs]

(a) Define $\langle\langle x, y\rangle\rangle$.

(b) Why is this a bijection between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N} \setminus \{0\}$?

(c) Show that the encoding and decoding functions are computable. (See **[2017P6Q3 (a)(i),(ii)]**)

(d) Define $\langle x, y\rangle$.

(e) Why is this a bijection between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$?

(f) Show that the encoding and decoding functions are computable.

Exercise 28 [Reverse engineer the program 3] Attempt **[2006P3Q7 (a)]**.

Exercise 29 [Numerical codings of lists]

(a) Show how a list is encoded using the numerical codings of pairs.

(b) Is this encoding bijective?

(c) Show that the encoding and decoding are computable.

(d) (optional) Show how you could obtain an injective mapping for OCaml datatypes.

(e) (optional) Show how to implement the for-each construct.

Exercise 30 [Instruction encodings]

(a) Explain how RM instructions are encoded.

(b) How can these be decoded?

(c) Show that both the encoding and decoding functions are computable.

(d) Is this encoding a bijection?

Exercise 31 [Program encodings]

(a) How is an RM program encoded?

(b) Why did the lecturer choose to have errorneous halts as well as proper halts?

Exercise 32

(a) Attempt **[2014P6Q3 (a), (b)]**.

(b) Read the statements for **[2011P6Q3 (a)]** and **[1996P3Q9 (a)]**.

Exercise 33 [Program encoding in the emulator] (optional)

(a) Write a function for the RM emulator that takes a program and returns the index of the program.

(b) Write a function that takes the index of a program and returns the program.

(c) Choose an index of your preference and check if the program terminates. (be careful with erroneous halts)

(d) See also **[1999P3Q9 (d)]**, **[1996P3Q9 (a)]** or **[1995P3Q9 (a)]**.

Exercise 34 [Counting programs] Attempt the following subquestions from **[2007P3Q7 (b)(iii)]**:

(a) Explain why there are only countably many computable functions from $\mathbb{N} \to \mathbb{N}$.

(b) Deduce that there exists a partial function from $\mathbb{N} \rightharpoonup \mathbb{N}$ that is not computable. (Any standard results you use about countable and uncountable sets should be clearly stated, but need not be proved.)

(c) If a partial function f from $\mathbb{N} \rightharpoonup \mathbb{N}$ is computable, how many different register machine programs are there that compute f?

Lecture 4

Exercise 35 [Universal RM]

(a) Define what is the *universal RM*. (See **[2019P6Q5 (a)]**, **[2013P6Q3 (a)]**, **[2007P3Q7 (a) (ii)]** or **[1995P3Q9 (b)]**)

(b) Describe the high level steps for creating a universal machine. (See **[1999P3Q9 (e)]**)

(c) (optional) Implement the universal RM in the emulator. Having implemented the program concatenation and the function that permutes the registers used by a program, will make your code much simpler.

Exercise 36 [Number of computation steps] Explain how you would use the universal register machine construction to create RMs for the following:

(a) Show that the function

$$\ell(e,x) = \begin{cases} \text{number of steps in the computation of } e(x) & \text{if } e(x) \text{ halts} \\ \text{undef} & \text{otherwise} \end{cases}$$

is computable. (See **[2013P6Q3 (c) (iv)]**)

(b) Given two programs e_1 and e_2, show how to create a program e, such that

$$e(x) = \begin{cases} \text{undef} & \text{if } \ell(e_1,x) = \ell(e_2,x) = \infty \\ e_1(x) & \text{if } \ell(e_1,x) \leq \ell(e_2,x) \\ e_2(x) & \text{otherwise} \end{cases}$$

.

(c) Given a program e and a natural t, write a function

$$\text{stops}(e,x,t) = \begin{cases} 1 & \text{if } \ell(e,x) \leq t \\ 0 & \text{otherwise} \end{cases}$$

.